
Matteo Baldoni, Tran Cao Son
M. Birna van Riemsdijk, Michael Winikoff (eds.)

# Declarative Agent Languages and Technologies

*Fifth International Workshop, DALT 2007*
*Honolulu, Hawai'i, May 14th, 2007*

*Workshop Notes*

---

DALT 2007 Home Page:
`http://www.di.unito.it/~baldoni/DALT-2007/`

# Preface

The workshop on Declarative Agent Languages and Technologies (DALT), in its *fifth edition* this year, is a well-established forum for researchers interested in sharing their experiences in combining declarative and formal approaches with engineering and technology aspects of agents and multiagent systems. Building complex agent systems calls for models and technologies that ensure predictability, allow for the verification of properties, and guarantee flexibility. Developing technologies that can satisfy these requirements still poses an important and difficult challenge. Here, declarative approaches have the potential of offering solutions that satisfy the needs for both specifying and developing multiagent systems. Moreover, they are gaining more and more attention in important application areas such as the semantic web, web services, security, and electronic contracting.

DALT 2007 is being held as a satellite workshop of AAMAS 2007, the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, in Honolulu, Hawai'i. Following the success of DALT 2003 in Melbourne (LNAI 2990), DALT 2004 in New York (LNAI 3476), DALT 2005 in Utrecht (LNAI 3904), and DALT 2006 in Hakodate (LNAI 4327), DALT will again aim at providing a discussion forum to both (i) support the transfer of declarative paradigms and techniques to the broader community of agent researchers and practitioners, and (ii) to bring the issue of designing complex agent systems to the attention of researchers working on declarative languages and technologies.

This volume containts the eleven papers that have been selected by the Programme Committee for presentation at the workshop. Each paper received at least three reviews in order to supply the authors with a rich feedback that could stimulate the research as well as foster the discussion.

We would like to thank all authors for their contributions, the members of the Steering Committee for the precious suggestions and support, and the members of the Programme Committee for the excellent work during the reviewing phase.

March 26th, 2007

Matteo Baldoni
Tran Cao Son
M. Birna van Riemsdijk
Michael Winikoff

## Workshop Organisers

| | |
|---|---|
| Matteo Baldoni | University of Torino, Italy |
| Tran Cao Son | New Mexico State University, USA |
| M. Birna van Riemsdijk | Ludwig-Maximilians-Universitaet Muenchen, Germany |
| Michael Winikoff | RMIT University, Australia |

## Programme Committee

| | |
|---|---|
| Marco Alberti | University of Ferrara, Italy |
| Natasha Alechina | University of Nottingham, UK |
| Grigoris Antoniou | University of Crete, Greece |
| Matteo Baldoni | University of Torino, Italy, *Co-chair* |
| Cristina Baroglio | University of Torino, Italy |
| Rafael Bordini | University of Durham, UK |
| Keith Clark | Imperial College London, UK |
| Ulle Endriss | University of Amsterdam, The Netherlands |
| Benjamin Hirsch | Technical University Berlin, Germany |
| Shinichi Honiden | National Institute of Informatics, Japan |
| John Lloyd | Australian National University, Australia |
| Viviana Mascardi | University of Genova, Italy |
| John-Jules Ch. Meyer | Utrecht University, The Netherlands |
| Enrico Pontelli | New Mexico State University, USA |
| Birna van Riemsdijk | Ludwig-Maximilians-Universitaet Muenchen, Germany, *Co-chair* |
| Munindar Singh | North Carolina State University, USA |
| Tran Cao Son | New Mexico State University, USA, *Co-chair* |
| Chiaki Sakama | Wakayama University, Japan |
| Wamberto Vasconcelos | University of Aberdeen, UK |
| Mirko Viroli | University of Bologna, Italy |
| Michael Winikoff | RMIT University, Melbourne, Australia, *Co-chair* |

## Steering Committee

| | |
|---|---|
| João Leite | New University of Lisbon, Portugal |
| Andrea Omicini | University of Bologna-Cesena, Italy |
| Leon Sterling | University of Melbourne, Australia |
| Paolo Torroni | University of Bologna, Italy |
| Pınar Yolum | Bogazici University, Turkey |

## Additional Reviewers

| | | |
|---|---|---|
| Sebastian Sardina | Martin Caminada | Nirmit Desai |
| Berndt Farwer | | Yasuyuki Tahara |

## Sponsoring Institutions

Matteo Baldoni has partially been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net), and it has also been supported by MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national project.

M. Birna van Riemsdijk has partially supported by the project SENSORIA, IST-2005-016004.

# Table of Contents

# Integrating Agent Models and Dynamical Systems

Tibor Bosse, Alexei Sharpanskykh, and Jan Treur

Vrije Universiteit Amsterdam, Department of Artificial Intelligence,
De Boelelaan 1081a, 1081 HV, The Netherlands
{tbosse, sharp, treur}@cs.vu.nl
http://www.cs.vu.nl/~{tbosse, sharp, treur}

**Abstract.** Agent-based modelling approaches are usually based on logical languages, whereas in many areas dynamical system models based on differential equations are used. This paper shows how to model complex agent systems, integrating quantitative, numerical and qualitative, logical aspects, and how to combine logical and mathematical analysis methods.

## 1 Introduction

Existing models for complex systems are often based on quantitative, numerical methods such as Dynamical Systems Theory (DST) [23], and more in particular, differential equations. Such approaches often use numerical variables to describe global aspects of the system and how they affect each other over time; for example, how the number of predators affects the number of preys. An advantage of such numerical approaches is that numerical approximation methods and software environments are available for simulation.

The relatively new agent-based modelling approaches to complex systems take into account the local perspective of a possibly large number of separate agents and their specific behaviours in a system; for example, the different individual predator agents and prey agents. These approaches are usually based on qualitative, logical languages. An advantage of such logical approaches is that they allow (automated) logical analysis of the relationships between different parts of a model, for example relationships between global properties of the (multi-agent) system as a whole and local properties of the basic mechanisms within (agents of) the system. Moreover, by means of logic-based approaches, declarative models of complex systems can be specified using knowledge representation languages that are close to the natural language. An advantage of such declarative models is that they can be considered and analysed at a high abstract level. Furthermore, automated support (e.g., programming tools) is provided for manipulation and redesign of models.

Complex systems, for example organisms in biology or organisations in the socio-economic area, often involve both qualitative aspects and quantitative aspects. In particular, in the area of Cognitive Science, the lower-level cognitive processes of agents (e.g., sensory or motor processing) are often modelled using DST-based approaches. Furthermore, at the global level the dynamics of the environment, in which agents are situated, is often described by continuous models (i.e., models based

on differential equations); e.g., dynamic models of markets, or natural environmental oscillations. Yet agent-based (logical) languages are often used for describing high-level cognitive processes of agents (e.g., processes related to reasoning) and agent interaction with the environment (e.g., agent actions, execution of tasks).

It is not easy to integrate both types of approaches in one modelling method. On the one hand, it is difficult to incorporate logical aspects in differential equations. For example, qualitative behaviour of an agent that depends on whether the value of a variable is below or above a threshold is difficult to describe by differential equations. On the other hand, quantitative methods based on differential equations are not usable in the context of most logical, agent-based modelling languages, as these languages are not able to handle real numbers and calculations.

This paper shows an integrative approach to simulate and analyse complex systems, integrating quantitative, numerical and qualitative, logical aspects within one expressive temporal specification language. Some initial ideas behind the simulation approach proposed in this paper were described in [6]. The current paper elaborates upon these ideas by proposing more extensive means to design precise, stable, and computationally effective simulation models for hybrid systems (i.e., comprising both quantitative and qualitative aspects). Furthermore, it proposes techniques for analysis of hybrid systems, which were not previously considered elsewhere. The developed simulation and analysis techniques are supported by dedicated tools.

In Section 2, this language (called LEADSTO) is described in detail, and is applied to solve an example differential equation. In Section 3, it is shown how LEADSTO can solve a system of differential equations (for the case of the classical Predator-Prey model), and how it can combine quantitative and qualitative aspects within the same model. Section 4 demonstrates how existing methods for approximation (such as the Runge-Kutta methods) can be incorporated into LEADSTO, and Section 5 shows how existing methods for simulation with dynamic step size can be incorporated. Section 6 demonstrates how interlevel relationships can be established between dynamics of basic mechanisms (described in LEADSTO) and global dynamics of a process (described in a super-language of LEADSTO). Finally, Section 7 is a discussion.

## 2 Modelling Dynamics in LEADSTO

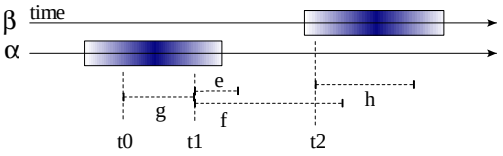
Dynamics can be modelled in different forms. Based on the area within Mathematics called calculus, the Dynamical Systems Theory [23] advocates to model dynamics by continuous state variables and changes of their values over time, which is also assumed continuous. In particular, systems of differential or difference equations are used. This may work well in applications where the world states are modelled in a quantitative manner by real-valued state variables. The world's dynamics in such application show continuous changes in these state variables that can be modelled by mathematical relationships between real-valued variables. However, not for all applications dynamics can be modelled in a quantitative manner as required for DST. Sometimes qualitative changes form an essential aspect of the dynamics of a process. For example, to model the dynamics of reasoning processes usually a quantitative approach will not work. In such processes states are characterised by qualitative state

properties, and changes by transitions between such states. For such applications often qualitative, discrete modelling approaches are advocated, such as variants of modal temporal logic, e.g. [20]. However, using such non-quantitative methods, the more precise timing relations are lost too. For the LEADSTO language described in this paper, the choice has been made to consider the timeline as continuous, described by real values, but for state properties both quantitative and qualitative variants can be used. The approach subsumes approaches based on simulation of differential or difference equations, and discrete qualitative modelling approaches. In addition, the approach makes it possible to combines both types of modelling within one model. For example, it is possible to model the exact (real-valued) time interval for which some qualitative property holds. Moreover, the relationships between states over time are described by either logical or mathematical means, or a combination thereof. This will be explained in more detail in Section 2.1. As an illustration, in Section 2.2 it will be shown how the logistic model for population growth in resource-bounded environments [4] can be modelled and simulated in LEADSTO.

### 2.1 The LEADSTO Language

Dynamics is considered as evolution of states over time. The notion of state as used here is characterised on the basis of an ontology defining a set of properties that do or do not hold at a certain point in time. For a given (order-sorted predicate logic) ontology Ont, the propositional language signature consisting of all *state ground atoms* (or *atomic state properties*) based on Ont is denoted by APROP(Ont). The *state properties* based on a certain ontology Ont are formalised by the propositions that can be made (using conjunction, negation, disjunction, implication) from the ground atoms. A *state* S is an indication of which atomic state properties are true and which are false, i.e., a mapping S: APROP(Ont) $\rightarrow$ {true, false}.

To specify simulation models a temporal language has been developed. This language (the LEADSTO language [7]) enables to model direct temporal dependencies between two state properties in successive states, also called *dynamic properties*. A specification of dynamic properties in LEADSTO format has as advantages that it is executable and that it can often easily be depicted graphically. The format is defined as follows. Let $\alpha$ and $\beta$ be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h non-negative real numbers. In the LEADSTO language the notation $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$ (also see Fig. 1), means:




*If state property $\alpha$ holds for a certain time interval with duration g, then after some delay (between e and f) state property $\beta$ will hold for a certain time interval of length h.*

**Fig. 1.** Timing relationships for LEADSTO expressions.

An example dynamic property that uses the LEADSTO format defined above is the following: "observes(agent_A, food_present) $\rightarrow\!\!\!\!\rightarrow_{2, 3, 1, 1.5}$ beliefs(agent_A, food_present)".

Informally, this example expresses the fact that, if agent A observes that food is present during 1 time unit, then after a delay between 2 and 3 time units, agent A will belief that food is present during 1.5 time units. In addition, within the LEADSTO language it is possible to use sorts, variables over sorts, real numbers, and mathematical operations, such as in "has_value(x, v) $\rightarrow\!\!\!\!\rightarrow_{e, f, g, h}$ has_value(x, v*0.25)". Next, a *trace* or *trajectory* $\gamma$ over a state ontology Ont is a time-indexed sequence of states over Ont (where the time frame is formalised by the real numbers). A LEADSTO expression $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$, holds for a trace $\gamma$ if:

$\forall t1\ [\forall t\ [t1-g \leq t < t1 \Rightarrow \alpha \text{ holds in } \gamma \text{ at time } t\ ] \Rightarrow \exists d\ [e \leq d \leq f\ \&\ \forall t'\ [t1+d \leq t' < t1+d+h \Rightarrow \beta \text{ holds in } \gamma \text{ at time } t'\ ]$

To specify the fact that a certain event (i.e., a state property) holds at every state (time point) within a certain time interval a predicate holds_during_interval(event, t1, t2) is introduced. Here event is some state property, t1 is the beginning of the interval and t2 is the end of the interval.

An important use of the LEADSTO language is as a specification language for simulation models. As indicated above, on the one hand LEADSTO expressions can be considered as logical expressions with a declarative, temporal semantics, showing what it means that they hold in a given trace. On the other hand they can be used to specify basic mechanisms of a process and to generate traces, similar to Executable Temporal Logic [3]. More details on the semantics of the LEADSTO language can be found in [7].

### 2.2 Solving the Initial Value Problem in LEADSTO: Euler's method

Often behavioural models in the Dynamical Systems Theory are specified by systems of differential equations with given initial conditions for continuous variables and functions. A problem of finding solutions to such equations is known as an initial value problem in the mathematical analysis. One of the approaches for solving this problem is based on discretisation, i.e., replacing a continuous problem by a discrete one, whose solution is known to approximate that of the continuous problem. For this methods of numerical analysis are usually used [22]. The simplest approach for finding approximations of functional solutions for ordinary differential equations is provided by Euler's method. Euler's method for solving a differential equation of the form $dy/dt = f(y)$ with the initial condition $y(t_0)=y_0$ comprises the difference equation derived from a Taylor series:

$$y(t) = \sum_{n=0}^{\infty} \frac{y^{(n)}(t_0)}{n!} * (t - t_0)^n ,$$

where only the first member is taken into account: $y_{i+1}=y_i+h* f(y_i)$, where $i \geq 0$ is the step number and $h>0$ is the integration step size. This equation can be modelled in the LEADSTO language in the following way:

- Each integration step corresponds to a state, in which an intermediate value of y is calculated.
- The difference equation is modelled by a transition rule to the successive state in the LEADSTO format.
- The duration of an interval between states is defined by a step size h.

Thus, for the considered case the LEADSTO simulation model comprises the rule:

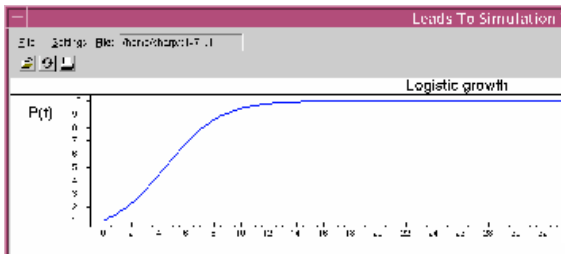
has_value(y, v1) $\rightarrow\!\!\!\!\rightarrow_{0,\,0,\,h,\,h}$ has_value(y, v1+h* f(v1))

The initial value for the function y is specified by the following LEADSTO rule:

holds_during_interval(has_value(y, $y_0$), 0, h)

By performing a simulation of the obtained model in the LEADSTO environment an approximate functional solution to the differential equation can be found.

To illustrate the proposed simulation-based approach based on Euler's method in LEADSTO, the logistic growth model or the Verhulst model [4] which is often used to describe the population growth in resource-bounded environments, is considered: dP/dt = r*P(1-P/K), where P is the population size at time point t; r and K are some constants. This model corresponds to the following LEADSTO simulation model: has_value(y, v1) $\rightarrow\!\!\!\!\rightarrow_{0,\,0,\,h,\,h}$ has_value(y, v1+ h*r* v1*(1-v1/K)).




The result of simulation of this model in the LEADSTO environment with the parameters r=0.5 and K=10 and the initial value P(0)=1 is given in Figure 2.

**Fig. 2.** Logistic growth function modelled in LEADSTO with parameters r=0.5, K=10, P(0)=1.

## 3 Modelling the Predator-Prey Model in LEADSTO

The proposed simulation-based approach can be applied for solving a system of ordinary differential equations. In order to illustrate this, the classical Lotka-Volterra model (also known as a Predator-Prey model) [21] is considered. The Lotka-Volterra model describes interactions between two species in an ecosystem, a predator and a prey. The model consists of two equations: the first one describes how the prey population changes and the second one describes how the predator population changes. If x(t) and y(t) represent the number of preys and predators respectively, that are alive in the system at time t, then the Lotka-Volterra model is defined by: dx/dt = a*x - b*x*y ; dy/dt = c*b*x*y - e*y where the parameters are defined by: a is the per capita birth rate of the prey, b is a per capita attack rate, c is the conversion efficiency of consumed prey into new predators, and e is the rate at which predators die in the absence of prey. To solve this system, numerical methods derived from a Taylor series up to some order can be used. In the following section it will be shown how Euler's (first-order rough) method can be used for creating a LEADSTO simulation model for finding the approximate solutions for the Predator-Prey problem. After that, in Section 3.2 it will be demonstrated how the generated LEADSTO simulation model can be extended by introducing qualitative behavioural aspects in the standard predator-prey model.

### 3.1 The LEADSTO language

Using the technique described in Section 2.2, the Lotka-Volterra model is translated into a LEADSTO simulation model as follows:

has_value(x, v1) $\wedge$ has_value(y, v2) $\rightarrow_{0, 0, h, h}$ has_value(x, v1+h*(a*v1-b*v1*v2))
has_value(x, v1) $\wedge$ has_value(y, v2) $\rightarrow_{0, 0, h, h}$ has_value(y, v2+h*(c*b*v1*v2-e*v2))

The initial values for variables and functions are specified as for the general case. Although Euler's method offers a stable solution to a stable initial value problem, a choice of initial values can significantly influence the model's behaviour. More specifically, the population size of both species will oscillate if perturbed away from the equilibrium. The amplitude of the oscillation depends on how far the initial values of x and y depart from the equilibrium point. The equilibrium point for the considered model is defined by the values x=e/(c*b) and y=a/b. For example, for the parameter settings a=1.5, b=0.2, c=0.1 and e=0.5 the equilibrium is defined by x=25 and y=7.5. Yet a slight deviation from the equilibrium point in the initial values ($x_0$=25, $y_0$=8) results in the oscillated (limit cycle) behaviour.

### 3.2 Extending the Standard Predator-Prey Model with Qualitative Aspects

In this section, an extension of the standard predator-prey model is considered, with some qualitative aspects of behaviour. Assume that the population size of both predators and preys within a certain eco-system is externally monitored and controlled by humans. Furthermore, both prey and predator species in this eco-system are also consumed by humans. A control policy comprises a number of intervention rules that ensure the viability of both species. Among such rules could be following:

- in order to keep a prey species from extinction, a number of predators should be controlled to stay within a certain range (defined by pred_min and pred_max);
- if a number of a prey species falls below a fixed minimum (prey_min), a number of predators should be also enforced to the prescribed minimum (pred_min);
- if the size of the prey population is greater than a certain prescribed bound (prey_max), then the size of the prey species can be reduced by a certain number prey_quota (cf. a quota for fish catch).

These qualitative rules can be encoded into the LEADSTO simulation model for the standard predator-prey case by adding new dynamic properties and changing the existing ones in the following way:

has_value(x, v1) $\wedge$ has_value(y, v2) $\wedge$ v1< prey_max $\rightarrow_{0, 0, h, h}$ has_value(x, v1+h*(a*v1-b*v1*v2))
has_value(x, v1) $\wedge$ has_value(y, v2) $\wedge$ v1 $\geq$ prey_max $\rightarrow_{0, 0, h, h}$
has_value(x, v1+h*(a*v1-b*v1*v2) - prey_quota)
has_value(x, v1) $\wedge$ has_value(y, v2) $\wedge$ v1 $\geq$ prey_min $\wedge$ v2 < pred_max $\rightarrow_{0, 0, h, h}$
has_value(y, v2+h* (c*b*v1*v2-e*v2))
has_value(x, v1) $\wedge$ has_value(y, v2) $\wedge$ v2 $\geq$ pred_max $\rightarrow_{0, 0, h, h}$ has_value(y, pred_min)
has_value(x, v1) $\wedge$ has_value(y, v2) $\wedge$ v1 < prey_min $\rightarrow_{0, 0, h, h}$ has_value(y, pred_min)

The result of simulation of this model using Euler's method with the parameter settings: a=4; b=0.2, c=0.1, e=8, pred_min=10, pred_max=30, prey_min=40, prey_max=100, prey_quota=20, $x_0$=90, $y_0$=10 is given in Fig. 3. More examples of the LEADSTO simulation models combining quantitative and qualitative aspects of behaviour can be

found in [5] and [6]. In [6], a LEADSTO model for classical conditioning is presented, based on Machado's differential equation model [18]. This model integrates quantitative aspects such as levels of preparation with qualitative aspects such as the occurrences of certain stimuli. In [5], a LEADSTO model for eating regulation disorders is presented. This model integrates quantitative aspects such as a person's weight with qualitative aspects such as the decision to eat.

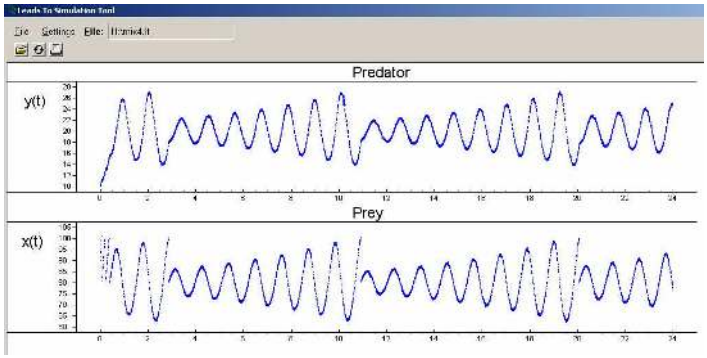



**Fig. 3.** Simulation results for the Lotka-Volterra model combined some qualitative aspects.

## 4 Simulating the Predator-Prey Model by the Runge-Kutta Method

As shown in [22], within Euler's method the local error at each step (of size $h$) is $O(h^2)$, while the accumulated error is $O(h)$. However, the accumulated error grows exponentially as the integration step size increases. Therefore, in situations in which precision of a solution is required, high order numerical methods are used. For the purpose of illustration of high-order numerical approaches the fourth-order Runge-Kutta method is considered. This method is derived from a Taylor expansion up to the fourth order. It is known to be very accurate (the accumulated error is $O(h^4)$) and stable for a wide range of problems. The Runge-Kutta method for solving a differential equation of the form $dx/dt = f(t, x)$ is described by the following formulae:

$x_{i+1} = x_i + h/6 *(k_1 + 2*k_2 + 2*k_3 + k_4)$,

where $i \geq 0$ is the step number, $h>0$ is the integration step size, and

$k_1 = f(t_i, x_i)$, $k_2 = f(t_i + h/2, x_i + h/2 *k_1)$, $k_3 = f(t_i + h/2, x_i + h/2 *k_2)$, $k_4 = f(t_i + h, x_i + h* k_3)$.

Now, using the Runge-Kutta method, the classical Lotka-Volterra model considered in the previous section is described in the LEADSTO format as follows:

$\text{has\_value}(x, v1) \wedge \text{has\_value}(y, v2) \rightarrow_{0, 0, h, h} \text{has\_value}(x, v1 + h/6 *(k_{11} + 2*k_{12} + 2*k_{13} + k_{14}))$
$\text{has\_value}(x, v1) \wedge \text{has\_value}(y, v2) \rightarrow_{0, 0, h, h} \text{has\_value}(y, v2 + h/6 *(k_{21} + 2*k_{22} + 2*k_{23} + k_{24}))$,

where:

$k_{11} = a*v1-b*v1*v2$, $k_{21} = c*b*v1*v2 - e*v2$, $k_{12} = a*(v1 + h/2 *k_{11}) - b*(v1 + h/2 *k_{11})*(v2 + h/2 *k_{21})$, $k_{22} = c*b*(v1 + h/2 *k_{11})*(v2 + h/2 *k_{21}) - e*(v2 + h/2 *k_{21})$, $k_{13} = a*(v1 + h/2 *k_{12}) - b*(v1 + h/2 *k_{12})*(v2 + h/2 *k_{22})$, $k_{23} = c*b*(v1 + h/2 *k_{12})*(v2 + h/2 *k_{22}) - e*(v2 + h/2 *k_{22})$, $k_{14} = a*( v1 + h *k_{13}) - b*(v1 + h *k_{13})*(v2 + h *k_{23})$, $k_{24} = c*b*(v1 + h *k_{13})*(v2 + h *k_{23}) - e*(v2 + h *k_{23})$.

## 5 Simulation with Dynamic Step Size

Although for most cases the Runge-Kutta method with a small step size provides accurate approximations of required functions, this method can still be computationally expensive and, in some cases, inaccurate. In order to achieve a higher accuracy together with minimum computational efforts, methods that allow the dynamic (adaptive) regulation of an integration step size are used. This section shows how such methods can be incorporated in LEADSTO.

To illustrate the use of methods for dynamic step size control, the biochemical model of [13], summarised in Table 1, is considered.

**Table. 1.** Glycolysis model by [13].

| Variables | Moiety conservation | Rate equations |
|---|---|---|
| W: Fructose 6-phosphate<br>X : phosphoenolpyruvate<br>Y : pyruvate<br>N1 : ATP; N2 : ADP; N3 : AMP | N1[t] + N2[t] + N3 = 20 | Vxy = 343*N2[t]*X[t]/((0.17 + N2[t])*(0.2 + X[t]))<br>Vak = -(432.9*N3*N1[t] - 133*N2[t]^2)<br>Vatpase = 3.2076*N1[t]<br>Vpdc = 53.1328*Y[t]/(0.3 + Y[t]) (*10.0*Y[t]*)<br>Vpfk = 45.4327*W^2/(0.021*(1 + 0.15*N1[t]^2/N3^2 + W^2)) |
| Differential equations<br>X'[t] == 2*Vpfk - Vxy<br>Y'[t] == Vxy - Vpdc<br>N1'[t] == Vxy + Vak - Vatpase<br>N2'[t] == -Vxy - 2*Vak + Vatpase | Initial conditions<br>N1[0] == 10<br>N2[0] == 9<br>Y[0] == 0<br>X[0] == 0 | |
| | Fixed metabolites<br>W = 0.0001; Z = 0 | |

This model describes the process of glycolysis in *Saccharomyces cerevisiae*, a specific species of yeast. This model is interesting to study, because the concentrations of some of the substances involved (in particular ATP and ADP) are changing at a variable rate: sometimes these concentrations change rapidly, and sometimes they change very slowly. Using the technique described in Section 2.2 (based on Euler's method), this model can be translated to the following LEADSTO simulation model:

has_value(x, v1) ∧ has_value(y, v2) ∧ has_value(n1, v3) ∧ has_value(n2, v4) →$_{0, 0, h, h}$
has_value(x, v1+ (2* (45.4327*w^2/ (0.021* (1+0.15*v3^2/ (20-v3-v4)^2+w^2)))-343*v4*v1/ ((0.17+v4)* (0.2+v1)))*h)

has_value(x, v1) ∧ has_value(y, v2) ∧ has_value(n1, v3) ∧ has_value(n2, v4) →$_{0, 0, h, h}$
has_value(y, v2+ (343*v4*v1/ ((0.17+v4)* (0.2+v1))-53.1328*v2/ (0.3+v2))*h)

has_value(x, v1) ∧ has_value(y, v2) ∧ has_value(n1, v3) ∧ has_value(n2, v4) →$_{0, 0, h, h}$
has_value(n1, v3+ (343*v4*v1/ ((0.17+v4)* (0.2+v1))+ (- (432.9* (20-v3-v4)*v3-133*v4^2))-3.2076*v3)*h)

has_value(x, v1) ∧ has_value(y, v2) ∧ has_value(n1, v3) ∧ has_value(n2, v4) →$_{0, 0, h, h}$
has_value(n2, v4+ (-343*v4*v1/ ((0.17+v4)* (0.2+v1))-2* (- (432.9* (20-v3-v4)*v3-133*v4^2))+3.2076*v3)*h)

The simulation results of this model (with a static step size of 0.00001) are shown in Fig. 4. Here the curves for N1 and N2 are initially very steep, but become flat after a while. As demonstrated by Figure 4, for the first part of the simulation, it is necessary to pick a small step size in order to obtain accurate results. However, to reduce computational efforts, for the second part a bigger step size is desirable. To this end, a number of methods exist that allow the dynamic adaptation of the step size in a simulation. Generally, these approaches are based on the fact that the algorithm

signals information about its own truncation error. The most straightforward (and most often used) technique for this is *step doubling* and *step halving*, see, e.g. [Gear 1971]. The idea of step doubling is that, whenever a new simulation step should be performed, the algorithm compares the result of applying the current step twice with the result of applying the double step (i.e., the current step * 2) once. If the difference between both solutions is smaller than a certain threshold ε, then the double step is selected. Otherwise, the algorithm determines whether step halving can be applied: it compares the result of applying the current step once with the result of applying the half step (i.e., the current step * 0.5) twice. If the difference between both solutions is smaller than ε, then the current step is selected. Otherwise, the half step is selected.

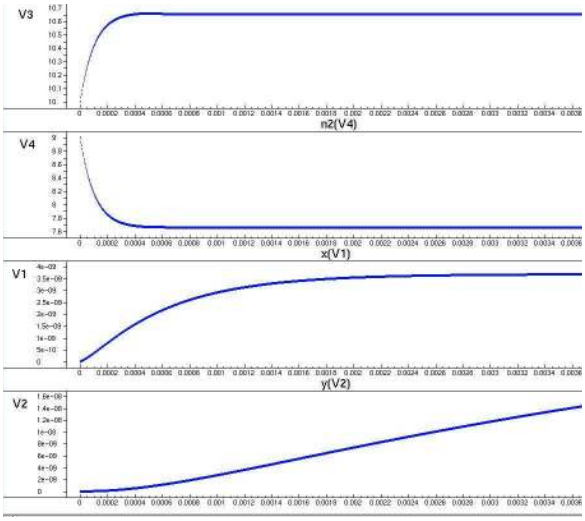



**Fig. 4.** Simulation results of applying Euler's method to [13]'s glycolysis model.

Since its format allows the modeller to include qualitative aspects, it is not difficult to incorporate step doubling and step halving into LEADSTO. To illustrate this, consider the general LEADSTO rule shown in Section 2.2 for solving a differential equation of the form dy/dt = f(y) using Euler's method:

has_value(y, v1) $\twoheadrightarrow_{0,\,0,\,h,\,h}$ has_value(y, v1+h* f(v1))

Adding step doubling and step halving to this rule yields the following three rules:

step(h) ∧ has_value(y, v1) ∧ |( v1+2h* f(v1)) - ((v1+h* f(v1))+h* f(v1+h* f(v1)))| ≤ ε
$\rightarrow_{0,\,0,\,2h,\,2h}$ has_value(y, v1+2h* f(v1)) ∧ step(2h)

step(h) ∧ has_value(y, v1) ∧ |( v1+2h* f(v1)) - ((v1+h* f(v1))+h* f(v1+h* f(v1)))| > ε ∧
|( v1+h* f(v1)) - ((v1+0.5h* f(v1))+0.5h* f(v1+0.5h* f(v1)))| ≤ ε
$\rightarrow_{0,\,0,\,h,\,h}$ has_value(y, v1+h* f(v1)) ∧ step(h)

step(h) ∧ has_value(y, v1) ∧ |( v1+h* f(v1)) - ((v1+0.5h* f(v1))+0.5h* f(v1+0.5h* f(v1)))| ≤ ε
$\rightarrow_{0,\,0,\,0.5h,\,0.5h}$ has_value(y, v1+0.5h* f(v1)) ∧ step(0.5h)

Besides step doubling, many other techniques exist in the literature for dynamically controlling the step size in quantitative simulations. Among these are several

techniques that are especially aimed at the Runge-Kutta methods, see, e.g., [24], Chapter 16 for an overview. Although it is possible to incorporate such techniques into LEADSTO, they are not addressed here because of space limitations.

## 6 Analysis In Terms of Local-Global Relations

Within the area of agent-based modelling, one of the means to address complexity is by modelling processes at different levels, from the global level of the process as a whole, to the local level of basic elements and their mechanisms. At each of these levels dynamic properties can be specified, and by interlevel relations they can be logically related to each other; e.g., [14], [27]. These relationships can provide an explanation of properties of a process as a whole in terms of properties of its local elements and mechanisms. Such analyses can be done by hand, but also software tools are available to automatically verify the dynamic properties and their interlevel relations. To specify the dynamic properties at different levels and their interlevel relations, a more expressive language is needed than simulation languages based on causal relationships, such as LEADSTO. The reason for this is that, although the latter types of languages are well suited to express the basic mechanisms of a process, for specifying global properties of a process it is often necessary to formulate complex relationships between states at different time points. To this end, the formal language TTL has been introduced as a super-language of LEADSTO; cf. [8]. It is based on order-sorted predicate logic and, therefore, inherits the standard semantics of this variant of predicate logic. That is, the semantics of TTL is defined in a standard way, by interpretation of sorts, constants, functions and predicates, and variable assignments. Furthermore, TTL allows representing numbers and arithmetical functions. Therefore, most methods used in Calculus are expressible in TTL, including methods based on derivatives and differential equations. In this section, first (in Section 6.1) it is shown how to incorporate differential equations in the predicate-logical language TTL that is used for analysis. Next, in Section 6.2 a number of global dynamic properties are identified, and it is shown how they can be expressed in TTL. In Section 6.3 a number of local dynamic properties are identified and expressed in TTL. Finally, Section 6.4 discusses how the global properties can be logically related to local properties such that a local property implies the global property.

### 6.1 The LEADSTO language

As mentioned earlier, traditionally, analysis of dynamical systems is often performed using mathematical techniques such as the Dynamical Systems Theory. The question may arise whether or not such modelling techniques can be expressed in the Temporal Trace Language TTL. In this section it is shown how modelling techniques used in the Dynamical Systems approach, such as difference and differential equations, can be represented in TTL. First the discrete case is considered. As an example consider again the logistic growth model: $dP/dt = r*P(1-P/K)$. This equation can be expressed in TTL on the basis of a discrete time frame (e.g., the natural numbers) in a straightforward manner:

$\forall t\ \forall v$ state($\gamma$ , t) $\models$ has_value(P, v) $\Rightarrow$ state($\gamma$ , t+1) $\models$ has_value(P, v + h • r • v • (1 - v/K))

The traces $\gamma$ satisfying the above dynamic property are the solutions of the difference equation. However, it is also possible to use the dense time frame of the real numbers, and to express the differential equation directly. To this end, the following relation is introduced, expressing that x = dy/dt:

is_diff_of($\gamma$, x, y) :
$\forall$t,w $\forall\varepsilon>0\ \exists\delta>0\ \forall$t',v,v' [ 0 < dist(t',t) < $\delta$ & state($\gamma$, t) $\models$ has_value(x, w) &
state($\gamma$, t) $\models$ has_value(y, v) & state($\gamma$, t') $\models$ has_value(y, v') $\Rightarrow$ dist((v'-v)/(t'-t),w) < $\varepsilon$ ]

where $\gamma$ is the trace that describes the change of values of x and y over time, dist(u,v) is defined as the absolute value of the difference, i.e. u-v if this is $\geq$ 0, and v-u otherwise. Using this, the differential equation can be expressed by is_diff_of($\gamma$ , r • P (1 - P/K), P).

The traces $\gamma$ for which this statement is true are (or include) solutions for the differential equation. Models consisting of combinations of difference or differential equations can be expressed in a similar manner. This shows how modelling constructs often used in DST can be expressed in TTL. Thus, TTL on the one hand subsumes modelling languages based on differential equations, but on the other hand enables the modeller to express more qualitative, logical concepts as well.

### 6.2 Mathematical Analysis in TTL: Global Dynamic Properties

Within Dynamical Systems Theory and Calculus, also for global properties of a process more specific analysis methods are known. Examples of such analysis methods include mathematical methods to determine equilibrium points, the behaviour around equilibrium points, and the existence of limit cycles [10]. Suppose a set of differential equations is given, for example a predator prey model: dx/dt = f(x, y) ; dy/dt = g(x, y), where f(x, y) and g(x, y) are arithmetical expressions in x and y. Within TTL the following abbreviation is introduced as a definable predicate:

point($\gamma$, t, x, v, y, w) $\Leftrightarrow$ state($\gamma$, t) |= has_value(x, v) $\wedge$ has_value(y, w)

Using this predicate, the following global properties can for example be specified:

*Monotonicity*
monotic_increase_after($\gamma$, t, x) $\Leftrightarrow$
$\forall$t1, t2 [ t $\leq$ t1 < t2 & point($\gamma$, t1, x, v1, y, w1) & point($\gamma$, t2, x, v2, y, w2) $\Rightarrow$ v1<v2 ]

*Bounded*
upward_bounded_after_by($\gamma$, t, M) $\Leftrightarrow\ \forall$t1 [ t $\leq$ t1 & point($\gamma$, t1, x, v1, y, w1) $\Rightarrow$ v1$\leq$M ]

*Equilibrium points*
These are points in the (x, y) plane for which, when they are reached by a solution, the state stays at this point in the plane for all future time points. This can be expressed as a global dynamic property in TTL as follows:
has_equilibrium($\gamma$, x, v, y, w) $\Leftrightarrow\ \forall$t1 [ point($\gamma$, t1, x, v, y, w) $\Rightarrow\ \forall$t2$\geq$t1 point($\gamma$, t2, x, v, y, w) ]
occurring_equilibrium($\gamma$, x, v, y, w) $\Leftrightarrow\ \exists$t point($\gamma$, t, x, v, y, w) & has_equilibrium($\gamma$, x, v, y, w)

*Behaviour Around an Equilibrium*
attracting($\gamma$, x, v, y, w, $\varepsilon$0) $\Leftrightarrow$ has_equilibrium($\gamma$, x, v, y, w) &
$\varepsilon$0>0 $\wedge\ \forall$t [ point($\gamma$, t, x, v1, y, w1) $\wedge$ dist(v1, w1, v, w) < $\varepsilon$0 $\Rightarrow$
$\forall\varepsilon>0\ \exists$t1$\geq$t $\forall$t2$\geq$t1 [ point($\gamma$, t2, x, v2, y, w2) $\Rightarrow$ dist(v2, w2, v, w) < $\varepsilon$ ] ]

Here, dist(v1, w1, v2, w2) denotes the distance between the points (v1, w1) and (v2, w2) in the (x, y) plane.

*Limit cycle*
A limit cycle is a set S in the x, y plane such that
$\forall t, v, w$ point($\gamma$, t, x, v, y, w) & (v, w) $\in$ S $\Rightarrow$ $\forall t' \geq t, v', w'$ [ point($\gamma$, t', x, v', y, w') $\Rightarrow$ (v', w') $\in$ S ]

In specific cases the set can be expressed in an implicit manner by a logical and/or algebraic formula, e.g., an equation, or in an explicit manner by a parameterisation. For these cases it can be logically expressed that a set S is a limit cycle.

(1) When S is defined in an implicit manner by a formula $\varphi(v, w)$ with $S = \{ (v, w) \mid \varphi(v, w) \}$, then it is defined that S is a limit cycle as follows:

$\forall t, v, w$ point($\gamma$, t, x, v, y, w) & $\varphi(v, w) \Rightarrow$ $\forall t' \geq t, v', w'$ [ point($\gamma$, t', x, v', y, w') $\Rightarrow \varphi(v', w')$ ]

E.g., when S is a circle defined by a formula of the form $S = \{ (v, w) \mid v^2 + w^2 = r^2 \}$

(2) When a set S in the plane is parameterised by two functions $c1, c2: [0, 1] \rightarrow \Re$, i.e., $S = \{ (c1(u), c2(u)) \mid u \in [0, 1] \}$, then S is a limit cycle if

$\forall t, u$ point($\gamma$, t, c1(u), c2(u)) $\Rightarrow \forall t' \geq t \exists u'$ point($\gamma$, t', c1(u'), c2(u'))

An example of a parameterising for S in the shape of a circle is as follows:

$c1(u) = r \cos 2\pi u$, $c2(u) = r \sin 2\pi u$

In many cases, however, the set S cannot be expressed explicitly in the form of an equation or an explicitly defined parameterisation. What still can be done often is to establish the existence of a limit cycle within a certain area, based on the Poincaré-Bendixson Theorem [16].

### 6.3 Mathematical Analysis in TTL: Local Dynamic Properties

The global dynamic properties described above can also be addressed from a local perspective. For example, the property of monotonicity (which was expressed above for a whole trace after a certain time point t), can also be expressed for a certain interval (with duration d) around t, as shown below.

*Local monotonicity property*
monotic_increase_around($\gamma$, t, x, d) $\Leftrightarrow$
$\forall t1, t2$ [ $t-d \leq t1 < t < t2 \leq t+d$ & point($\gamma$, t1, x, v1, y, w1) & point($\gamma$, t2, x, v2, y, w2) $\Rightarrow$ $v1 < v2$ ]

In terms of f and g:
monotic_increase_around($\gamma$, t, x, d) $\Leftrightarrow$ point($\gamma$, t, x, v1, y, w1) $\Rightarrow$ $f(v1, w1) > 0$

*Local bounding property*
upward_bounding_around($\gamma$, t, M, $\delta$, d) $\Leftrightarrow$
[ point($\gamma$, t, x, v1, y, w1) $\Rightarrow \forall t'$ [ $t \leq t' \leq t+d$ & point($\gamma$, t', x, v2, y, w2) $\Rightarrow M-v2 \geq (1-\delta)*(M-v1)$ ]

In terms of f and g from the equations $dx/dt = f(x, y)$ and $dy/dt = g(x, y)$:
upward_bounding_around($\gamma$, t, M, $\delta$, d) $\Leftrightarrow$ point($\gamma$, t, x, v1, y, w1) $\Rightarrow$ $f(v1, w1) \leq \delta/d\ (M - v1)$

*Local equilibrium property*
From the local perspective of the underlying mechanism, equilibrium points are those points for which $dx/dt = dy/dt = 0$, i.e., in terms of f and g for this case $f(x, y) = g(x, y) = 0$.
equilibrium_state(v, w) $\Leftrightarrow$ $f(v, w) = 0$ & $g(v, w) = 0$

*Local property for behaviour around an equilibrium*:
attracting(γ, x, v, y, w, δ, ε0, d) ⇔ has_equilibrium(γ, x, v, y, w) &
ε0>0 ∧ 0< δ <1 ∧ d≥0 ∧ ∀t [ point(γ, t, x, v1, y, w1) ∧ dist(v1, w1, v, w) < ε0 ⇒
∀t' [ t+d≤t'≤t+2d & point(γ, t', x, v2, y, w2) ⇒ dist(v2, w2, v, w) < δ*dist(v1, w1, v, w) ] ]

In terms of f and g, this can be expressed by relationships for the eigen values of the matrix of derivatives of f and g.

*Local limit cycle property*
Let a set S in the plane be parameterised by two explicitly given functions c1, c2: [0, 1] → ℜ, i.e., S = { (c1(u), c2(u)) | u ∈ [0, 1] }, and d1(u) = dc1(u)/du, d2(u) = dc2(u)/du. Then S is a limit cycle if:
∀t, u point(γ, t, c1(u), c2(u)) ⇒ d1(u)*g(c1(u), c2(u)) = f(c1(u), c2(u))*d2(u)

### 6.4 Logical Relations between Local and Global Properties

The properties of local and global level can be logically related to each other by general interlevel relations, for example, the following ones:

| | |
|---|---|
| ∃d>0 ∀t'≥t monotic_increase_around(γ, t', x, d) | ⇒ monotic_increase_after(γ, t, x) |
| ∃d>0, δ>0 ∀t'≥t upward_bounding_around(γ, t, M, δ, d) | ⇒ upward_bounded_after_by(γ, t, M) |
| ∀t [ state(γ, t) \|= equilibrium_state(v, w) | ⇒ has_equilibrium(γ, x, v, y, w) |
| ∃d>0, δ>0 attracting(γ, x, v, y, w, δ, ε0, d) | ⇒ attracting(γ, x, v, y, w, ε0) |

These interlevel relations are general properties of dynamic systems, as explained, e.g., in [10]. Full proofs for these relations fall outside the scope of this paper. However, to make them a bit more plausible, the following sketch is given. The first interlevel relation involving monotonicity can be based on induction on the number of d-intervals of the time axis between two given time points t1 and t2. The second interlevel relation, involving boundedness is based on the fact that local bounding implies that in any d-interval, if the value at the start of the interval is below M, then it will remain below M in that interval. The third interlevel relation, on equilibrium points, is based on the fact that if at no time point the value changes, then at all time points after this value is reached, the value will be the same. For the fourth interlevel relation, notice that local attractiveness implies that for any d-interval the distance of the value to the equilibrium value at the end point is less than δ times the value at the starting point. By induction over the number of d-intervals the limit definition as used for the global property can be obtained.

## 7 Discussion

The LEADSTO approach discussed in this paper provides means to simulate models of dynamic systems that combine both quantitative and qualitative aspects. A dynamic system, as it is used here, is a system, which is characterised by states and transitions between these states. As such, dynamic systems as considered in [23], which are described by differential equations, constitute a subclass of the dynamic systems considered in this paper. Systems that incorporate both continuous

components and discrete components are sometimes called *hybrid systems*. Hybrid systems are studied in both computer science [9], [19] and control engineering [17]. They incorporate both continuous components, whose dynamics is described by differential equations and discrete components, which are often represented by finite-state automata. Both continuous and discrete dynamics of components influence each other. In particular, the input to the continuous dynamics is the result of some function of the discrete state of a system; whereas the input of the discrete dynamics is determined by the value of the continuous state. In the control engineering area, hybrid systems are often considered as switching systems that represent continuous-time systems with isolated and often simplified discrete switching events. Yet in computer science the main interest in hybrid systems lies in investigating aspects of the discrete behaviour, while the continuous dynamics is often kept simple.

Our LEADSTO approach provides as much place for modelling the continuous constituent of a system, as for modelling the discrete one. In contrast to many studies on hybrid systems in computer science (e.g., [25]), in which a state of a system is described by assignment of values to variables, in the proposed approach a state of a system is defined using a rich ontological basis (i.e., typed constants, variables, functions and predicates). This provides better possibilities for conceptualising and formalising different kinds of systems (including those from natural domains). Furthermore, by applying numerical methods for approximation of the continuous behaviour of a system, all variables in a generated model become discrete and are treated equally as finite-state transition system variables. Therefore, it is not needed to specify so-called *control points* [19], at which values of continuous variables are checked and necessary transitions or changes in a mode of a system's functioning are made. Moreover, using TTL, a super-language of LEADSTO, dynamical systems can be analysed by applying formalised standard techniques from mathematical calculus.

Since LEADSTO has a state-based semantics and allows a high ontological expressivity for defining state properties, many action-based languages (*A*, *B*, *C* [12], *L* [2] and their extensions) can be represented in (or mapped to) the LEADSTO format. In particular, trajectories that define the world evolution in action languages correspond to traces in LEADSTO, fluents evaluated in each state can be represented by state properties, and transitions between states due to actions can be specified by LEADSTO rules that contain the corresponding actions within the antecedents. Furthermore, to represent actions, observations, and goals of agents and facts about the world, the state ontology of LEADSTO includes corresponding sorts, functions and predicates. LEADSTO allows representing both static and dynamic laws as they are defined in [12], and non-deterministic actions with probabilities. To represent and reason about temporal aspects of actions, LEADSTO includes the sort TIME, which is a set of linearly ordered time points.

The expressions of query languages used to reason about actions [2], [12] can be represented in TTL, of which LEADSTO is a sublanguage. TTL formulae can express causality relations of query languages by implications and may include references to multiple states (e.g., histories of temporally ordered sequences of states). Using a dedicated tool [TTL], TTL formulae can be automatically checked on traces (or trajectories) that represent the temporal development of agent systems.

Concerning other related work, in [26], a logic-based approach to simulation-based modelling of ecological systems is introduced. Using this approach, continuous

dynamic processes in ecological systems are conceptualised by system dynamics models (i.e., sets of compartments with flows between them). For formalising these models and performing simulations, the logical programming language Prolog is used. In contrast to this, the LEADSTO approach provides a more abstract (or high-level) logic-based language for knowledge representation.

Also within the area of cognitive modelling, the idea to combine qualitative and quantitative aspects within one modelling approach is not uncommon. A number of architectures have been developed in that area, e.g., ACT-R [1] and SOAR [15]. Such cognitive architectures basically consist of a number of different modules that reflect specific parts of cognition, such as memory, rule-based processes, and communication. They have in common with LEADSTO that they are hybrid approaches, supporting both qualitative (or *symbolic*) and quantitative (or *subsymbolic*) structures. However, in LEADSTO these qualitative and quantitative concepts can be combined within the same expressions, whereas in ACT-R and SOAR separate modules exist to express them. In these cognitive architectures, often the role of the subsymbolic processes is to control the symbolic processes. For example, the subsymbolic part of ACT-R is represented by a large set of parallel processes that can be summarised by a number of mathematical equations, whereas its symbolic part is fulfilled by a production system. Here, the subsymbolic equations control many of the symbolic processes. For instance, if multiple production rules in ACT-R's symbolic part are candidates to be executed, a subsymbolic utility equation may estimate the relative cost and benefit associated with each rule and select the rule with the highest utility for execution.

Accuracy and efficiency of simulation results for hybrid systems provided by the proposed approach to a great extend depend on the choice of a numerical approximation method. Although the proposed approach does not prescribe usage of any specific approximation method (even the most powerful of them can be modelled in LEADSTO), for most of the cases the fourth-order Runge-Kutta method can be recommended, especially when the highest level of precision is not required. For simulating system models, for which high precision is demanded, higher-order numerical methods with an adaptive step size can be applied.

## References


1. Anderson, J.R., Lebiere, C. The atomic components of thought. Lawrence Erlbaum Associates, Mahwah, NJ (1998)
2. Baral, C., Gelfond, M., Provetti, A. Representing Actions: Laws, Observation and Hypothesis. Journal of Logic Programming, 31(1-3) (1997) 201-243
3. Barringer, H., Fisher, M., Gabbay, D., Owens, R., Reynolds, M. The Imperative Future: Principles of Executable Temporal Logic, Research Studies Press Ltd. and John Wiley & Sons (1996)
4. Boccara, N. Modeling Complex Systems. Graduate Texts in Contemporary Physics series, Springer-Verlag (2004)
5. Bosse, T., Delfos, M.F., Jonker, C.M., Treur, J. Modelling Adaptive Dynamical Systems to analyse Eating Regulation Disorders. Simulation Journal: Transactions of the Society for Modeling and Simulation International, 82 (2006) 159-171

6. Bosse, T., Jonker, C.M., Los, S.A., Torre, L. van der, Treur, J. Formalisation and Analysis of the Temporal Dynamics of Conditioning. In: Mueller, J.P. and Zambonelli, F. (eds.), Proceedings of the Sixth International Workshop on Agent-Oriented Software Engineering, AOSE'05 (2005) 157-168
7. Bosse, T., Jonker, C.M., Meij, L. van der, Treur, J. LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. In: Eymann, T. et al. (eds.), Proc. MATES'05. LNAI 3550. Springer Verlag (2005) 165-178. Extended version in: International Journal of Artificial Intelligence Tools. To appear, 2007
8. Bosse, T., Jonker, C.M., Meij, L. van der, Sharpanskykh, A., Treur, J. Specification and Verification of Dynamics in Cognitive Agent Models. In: Nishida, T. (ed.), Proceedings of the Sixth International Conference on Intelligent Agent Technology, IAT'06. IEEE Computer Society Press (2006) 247-254
9. Davoren, J.M., Nerode, A. Logics for Hybrid Systems. In Proceedings of the IEEE, 88 (7) (2000) 985-1010
10. Edwards, C.H., Penney, D. L. Calculus with Analytic Geometry. Prentice Hall, London, 5th edition (1998)
11. Gear, C.W. Numerical Initial Value Problems in Ordinary Differential Equations. Englewood Cliffs, NJ: Prentice-Hall (1971)
12. Gelfond, M., Lifschitz, V. Action languages, Electronic Transactions on AI, 3(16) (1998)
13. Hynne F, Dano S, Sorensen PG., Full-scale model of glycolysis in Saccharomyces cerevisiae. Biophys. Chem., 94 (1-2) (2001) 121-63
14. Jonker, C.M., Treur, J. Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. International Journal of Cooperative Information Systems 11 (2002) 51-92.
15. Laird, J.E., Newell, A., and Rosenbloom, P.S. Soar: an architecture for general intelligence. Artificial Intelligence 33 (1) (1987) 1-64.
16. Lefschetz, S. Differential equations: geometric theory. Dover Publications (2005)
17. Liberzon, D., Morse, A. S. Basic problems in stability and design of switched systems, IEEE Control Systems Magazine 19 (5) (1999) 59-70
18. Machado, A. Learning the Temporal Dynamics of Behaviour. Psychological Review, vol. 104 (1997) 241-265
19. Manna, Z., Pnueli, A. Verifying Hybrid Systems. In Hybrid Systems, LNCS 736, Springer-Verlag, (1993) 4-35
20. Meyer, J.J.Ch., Treur, J. (volume eds.). Agent-based Defeasible Control in Dynamic Environments. Series in Defeasible Reasoning and Uncertainty Management Systems (D. Gabbay and Ph. Smets, series eds.) vol. 7, Kluwer Academic Publishers (2002)
21. Morin P.J. Community Ecology. Blackwell Publishing, USA (1999)
22. Pearson, C.E.. Numerical Methods in Engineering and Science. CRC Press (1986)
23. Port, R.F., Gelder, T. van (eds.). Mind as Motion: Explorations in the Dynamics of Cognition. MIT Press, Cambridge, Mass (1995)
24. Press, W. H., Teukolsky, S. A., Vetterling,W. T., Flannery, B. P. Numerical recipes in C: the art of scientific computing. Cambridge university press, second edition (1992)
25. Rajeev, A., Henzinger, T.A., and Wong-Toi, H. Symbolic analysis of hybrid systems. In Proceedings of the 36th Annual Conference on Decision and Control (CDC), IEEE Press (1997) 702-707
26. Robertson, D., Bundy, A., Muetzelfeldt, R., Haggith, M., Ushold, M. Eco-Logic: Logic-Based Approaches to Ecological Modelling. MIT Press, Cambridge, Mass (1991)
27. Sharpanskykh, A., Treur, J. Verifying Interlevel Relations within Multi-Agent Systems. In: Brewka, G., Coradeschi, S., Perini, A., and Traverso, P. (eds.), Proc. of the 17th European Conference on Artificial Intelligence, ECAI'06, IOS Press (2006) 290-294

# On the Complexity Monotonicity Thesis for Environment, Behaviour and Cognition

Tibor Bosse, Alexei Sharpanskykh, and Jan Treur

Vrije Universiteit Amsterdam, Department of Artificial Intelligence,
De Boelelaan 1081a, 1081 HV, The Netherlands
{tbosse, sharp, treur}@cs.vu.nl
http://www.cs.vu.nl/~{tbosse, sharp, treur}

**Abstract.** Development of more complex cognitive systems during evolution is sometimes viewed in relation to environmental complexity. In more detail, growth of complexity during evolution can be considered for the dynamics of externally observable behaviour of agents, for their internal cognitive systems, and for the environment. This paper explores temporal complexity for these three aspects, and their mutual dependencies. A number of example scenarios have been formalised in a declarative temporal language, and the complexity of the structure of the different formalisations was measured. Thus, some empirical evidence was provided for the thesis that for more complex environments, more complex behaviour and more complex mental capabilities are needed.

## 1 Introduction

Behaviour of agents (both living organisms and artificial (software or hardware) agents) can occur in different types and complexities, varying from very simple behaviour to more sophisticated forms. Depending on the complexity of the externally observable behaviour, the internal mental representations and capabilities required to generate the behaviour also show a large variety in complexity. From an evolutionary viewpoint, for example, Wilson [16], p. 187 and Darwin [3], p. 163 point out how the development of behaviour relates to the development of more complex cognitive capabilities. Godfrey-Smith [4], p. 3 assumes a relationship between the complexity of the environment and the development of mental representations and capabilities. He formulates the main theme of his book in condensed form as follows: 'The function of cognition is to enable the agent to deal with environmental complexity' (the *Environmental Complexity Thesis*). In this paper, this thesis is refined as follows:

- the more complex the environment, the more sophisticated is the behaviour required to deal with this environment,
- the more sophisticated the behaviour, the more complex are the mental representations and capabilities needed

This refined thesis will be called the *Complexity Monotonicity Thesis*. The idea is that to deal with the physical environment, the evolution process has generated and still generates a variety of organisms that show new forms of behaviour. These new forms of behaviour are the result of new architectures of organisms, including

cognitive systems with mental representations and capabilities of various degrees of complexity. The occurrence of such more complex architectures for organisms and the induced more complex behaviour itself increases the complexity of the environment during the evolution process. New organisms that have to deal with the behaviour of such already occurring organisms live in a more complex environment, and therefore need more complex behaviour to deal with this environment, (to be) realised by an architecture with again more complex mental capabilities. In particular, more complex environments often ask for taking into account more complex histories, which requires more complex internal cognitive representations and dynamics, by which more complex behaviour is generated.

This perspective generates a number of questions. First, how can the Complexity Monotonicity Thesis be formalised, and in particular how can the 'more complex' relation be formalised for (1) the environment, (2) externally observable agent behaviour and (3) internal cognitive dynamics? Second, connecting the three items, how to formalise (a) when does a behaviour fit an environment: which types of externally observable behaviours are sufficient to cope with which types of environments, and (b) when does a cognitive system generate a certain behaviour: which types of internal cognitive dynamics are sufficient to generate which types of externally observable agent behaviour?

In this paper these questions are addressed from a dynamics perspective, and formalised by a declarative temporal logical approach. Four cases of an environment, suitable behaviour and realising cognitive system are described, with an increasing complexity over the cases. Next, for each case, complexity of the dynamics of environment, externally observable agent behaviour and internal cognitive system are formalised in terms of structure of the formalised temporal specifications describing them, thus answering (1) to (3). Moreover, (a) and (b) are addressed by establishing formalised logical (entailment) relations between the respective temporal specifications. By comparing the four cases with respect to complexity, the Complexity Monotonicity Thesis is tested.

## 2 Evolutionary Perspective

The environment imposes certain requirements that an agent's behaviour needs to satisfy; these requirements change due to changing environmental circumstances. The general pattern is as follows. Suppose a certain goal G for an agent (e.g., sufficient food uptake over time) is reached under certain environmental conditions ES1 (Environmental Specification 1), due to its Behavioural Specification BS1, realised by its internal (architecture) CS1 (Cognitive Specification 1). In other words, the behavioural properties BS1 are sufficient to guarantee G under environmental conditions ES1, formally ES1 & BS1 $\Rightarrow$ G, and the internal dynamics CS1 are sufficient to guarantee BS1, formally CS1 $\Rightarrow$ BS1. In other environmental circumstances, described by environmental specification ES2 (for example, more complex) the old circumstances ES1 may no longer hold, so that the goal G may no longer be reached by behavioural properties BS1. An environmental change from ES1 to ES2 may entail that behaviour BS1 becomes insufficient. It has to be replaced by new behavioural

properties BS2 (also more complex) which express how under environment ES2 goal G can be achieved, i.e., ES2 & BS2 $\Rightarrow$ G.

Thus, a population is challenged to realise such behaviour BS2 by changing its internal architecture and its dynamics, and as a consequence fulfill goal G again. This challenge expresses a redesign problem: the given architecture of the agent as described by CS1 (which entails the old behavioural specification BS1) is insufficient to entail the new behavioural requirements BS2 imposed by the new environmental circumstances ES2; the evolution process has to redesign the architecture into one with internal dynamics described by some CS2 (also more complex), with CS2 $\Rightarrow$ BS2, to realise the new requirements on behaviour.

Based on these ideas, the Complexity Monotonicity Thesis can be formalised in the following manner. Suppose $< E_1, B_1, C_1 >$ and $< E_2, B_2, C_2 >$ are triples of environment, behaviour and cognitive system, respectively, such that the behaviours $B_i$ are adequate for the respective environment $E_i$ and realised by the cognitive system $C_i$. Then the Complexity Monotonicity Thesis states that

$$E_1 \leq_c E_2 \Rightarrow B_1 \leq_c B_2 \quad \& \quad B_1 \leq_c B_2 \Rightarrow C_1 \leq_c C_2$$

Here $\leq_c$ is a partial ordering in complexity, where $X \leq_c Y$ indicates that Y is more complex than X. A special case is when the complexity ordering is assumed to be a total ordering where for every two elements X, Y either $X \leq_c Y$ or $Y \leq_c X$ (i.e., they are comparable), and when some complexity measure cm is available, assigning degrees of complexity to environments, behaviours and cognitive systems, such that

$$X \leq_c Y \Leftrightarrow cm(X) \leq cm(Y)$$

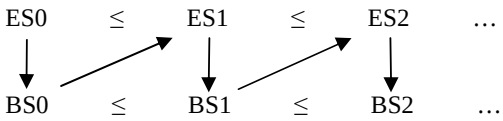
where $\leq$ is the standard ordering relation on (real or natural) numbers. In this case the Complexity Monotonicity Thesis can be reformulated as

$$cm(E_1) \leq cm(E_2) \Rightarrow cm(B_1) \leq cm(B_2) \quad \&$$
$$cm(B_1) \leq cm(B_2) \Rightarrow cm(C_1) \leq cm(C_2)$$

The Temporal Complexity Monotonicity Thesis can be used to explain increase of complexity during evolution in the following manner. Make the following assumption on Addition of Environmental Complexity by Adaptation, as described above:

- adaptation of a species to an environment adds complexity to this environment

Suppose an initial environment is described by ES0, and the adapted species by BS0. Then this transforms ES0 into a more complex environmental description ES1. Based on ES1, the adapted species will have description BS1. As ES1 is more complex than ES0, by the Complexity Monotonicity Thesis it follows that this BS1 is more complex than BS0: ES0 $\leq$ ES1 $\Rightarrow$ BS0 $\leq$ BS1. Therefore BS1 again adds complexity to the environment, leading to ES2, which is more complex than ES1, et cetera[1]:




This argument shows that the increase of complexity during evolution can be related to and explained by two assumptions: the Complexity Monotonicity Thesis,

[1] Note that this argument can also be applied to multiple species at the same time, i.e., species A increases the complexity of the environment, which causes another species B to adapt to this more complex environment.

and the Addition of Environmental Complexity by Adaptation assumption. This paper focuses on the former assumption.

## 3 Variations in Behaviour and Environment

To evaluate the approach put forward, a number of cases of increasing complexity are analysed, starting from very simple *stimulus-response behaviour* solely depending on stimuli the agent gets as input at a given point in time. This can be described by a very simple temporal structure: direct associations between the input state at one time point and the (behavioural) output state at a next time point. A next class of behaviours, with slightly higher complexity, analysed is *delayed response behaviour*: behaviour that not only depends on the current stimuli, but also may depend on input of the agent in the past. This pattern of behaviour cannot be described by direct functional associations between one input state and one output state; it increases temporal complexity compared to stimulus-response behaviour. For this case, the description relating input states and output states necessarily needs a reference to inputs received in the past. Viewed from an internal perspective, to describe mental capabilities generating such a behaviour, often it is assumed that it involves a memory in the form of an internal model of the world state. Elements of this world state model mediate between the agent's input and output states.

Other types of behaviour go beyond the types of reactive behaviour sketched above. For example, behaviour that depends in a more indirect manner on the agent's input in the present or in the past. Observed from the outside, this behaviour seems to come from within the agent itself, since no direct relation to current inputs is recognised. It may suggest that the agent is motivated by itself or acts in a goal-directed manner. For a study in *goal-directed behaviour* and foraging, see, for example, [5]. Goal-directed behaviour to search for invisible food is a next case of behaviour analysed. In this case the temporal description of the externally observable behavioural dynamics may become still more complex, as it has to take into account more complex temporal relations to (more) events in the past, such as the positions already visited during a search process. Also the internal dynamics may become more complex. To describe mental capabilities generating such a type of behaviour from an internal perspective, a mental state property *goal* can be used. A goal may depend on a history of inputs. Finally, a fourth class of behaviour analysed, which also goes beyond reactive behaviour, is *learning behaviour* (e.g., conditioning). In this case, depending on its history comprising a (possibly large) number of events, the agent's externally observable behaviour is tuned. As this history of events may relate to several time points during the learning process, this again adds temporal complexity to the specifications of the behaviour and of the internal dynamics.

To analyse these four different types of behaviour in more detail, four cases of a food supplying environment are considered in which suitable food gathering behaviours are needed. These cases are chosen in such a way that they correspond to the types of behaviour mentioned above. For example, in case 1 it is expected that stimulus-response behaviour is sufficient to cope with the environment, whilst in case 2, 3 and 4, respectively, delayed response behaviour, goal-directed behaviour, and

learning behaviour is needed). The basic setup is inspired by experimental literature in animal behaviour such as [6], [14], [15]. The world consists of a number of positions which have distances to each other. The agent can walk over these positions. Time is partitioned in fixed periods (days) of a duration of d time units (hours). Every day the environment generates food at certain positions, but this food may or may not be visible, accessible and persistent at given points in time. The four different types of environment with increasing temporal complexity considered are:

(1) Food is always visible and accessible. It persists until it is taken.
(2) Food is visible at least at one point in time and accessible at least at one later time point. It persists until it is taken.
(3) Food either is visible at least at one point in time and accessible at least at one later time point, or it is invisible and accessible the whole day. It persists until it is taken.
(4) One of the following cases holds:
    a) Food is visible at least at one point in time and accessible at least at one later time point. It persists until it is taken.
    b) Food is invisible and accessible the whole day. It persists until it is taken.
    c) Food pieces can disappear, and new pieces can appear, possibly at different positions. For every position where food appears, there are at least three different pieces in one day. Each piece that is present is visible. Each position is accessible at least after the second food piece disappeared.

Note that there is an accumulating effect in the increase of complexity of these types of environment. For example, the behaviour of environment (3) is described as the disjunction of the behaviour of environment (2) and another type of behaviour. For this reason, it is expected that agents that survive in environment n will also survive in environment n-1.

## 4 Modelling Approach

To express formal specifications for environmental, behavioural and cognitive dynamics for agents, the Temporal Trace Language (TTL, see [2]) is used. This language is a variant of order-sorted predicate logic. In dynamic property expressions, TTL allows explicit references to time points and traces. If a is a state property, then, for example state($\gamma$, t, input(agent)) |= a denotes that this state property holds in trace $\gamma$ at time point t in the input state of the agent. Here, a *trace* (or trajectory) is defined as a time-indexed sequence of states, where time points can be expressed, for example, by real or integer values. If these states are input states, such a trace is called an *input trace*. Similarly for an *output trace*. Moreover, an *input-output correlation* is defined as a binary relation C : Input_traces x Output_traces between the set of possible input traces and the set of possible output traces.

In the following sections, the four variations in behaviour and environment as introduced above are investigated in more detail. For formalising dynamic properties in TTL that will be used to specify these cases, the following state properties are used:

| | |
|---|---|
| at(o, p) | object o is at position p |
| visible(sp) | an object occurring in the state property sp is visible (e.g. as it is not covered by a large object) |
| accessible(p) | position p is accessible (e.g. because there is no enemy at the position) |
| distance(p1, p2, i) | the distance between positions p1 and p2 is i |
| max_dist | a constant indicating the maximum distance the agent can travel in one step |
| observed(sp) | the agent observes state property sp |
| performing_action(a) | the agent performs action a |

For example, a property that describes stimulus-response behaviour of an agent that goes to food, observed in the past can be expressed and formalised as follows:

At any point in time t,
if the agent observes itself at position p
and it observes an amount of food x at position p'
and position p' is accessible
then at the next time point after t the agent will go to position p'

Formalisation:
$\forall t\ \forall x\ \forall p\ \forall p'$
$[\ state(\gamma, t, input(agent)) \models observed(at(agent, p)) \wedge observed(at(food(x), p')) \wedge$
$observed(accessible(p'))\ \Rightarrow\ state(\gamma, t+1, output(agent)) \models performing\_action(goto(p'))\ ]$

# 5 Behavioural Cases

Using the introduced approach to formalise dynamic properties, the four variations in behaviour and environment are addressed in this section: stimulus-response, delayed-response, goal-directed, and learning behaviour.

## 5.1 Stimulus-Response Behaviour

As a first, most simple type of behaviour, stimulus-response behaviour is analysed in more detail. For this and the following cases of behaviour the following basis properties EP1-EP5 are used to describe the behaviour of the environment. They are specified both in a structured semi-formal temporal language, and in the formal temporal language TTL. Additionally, for every case specific properties of the environment will be specified.

*Environmental properties*

**EP1 Sufficient food within reach**
At the beginning of every day n (d is the duration of a day), the agent is positioned at a position p, and a sufficient amount x of food (c is the minimum) is provided at some position p' within reachable distance from p.
$\forall n\ \exists p\ \exists p'\ \exists x\ \exists i\ x>c\ \&\ i \leq max\_dist\ \&$
$state(\gamma, n*d, environment) \models at(agent, p) \wedge at(food(x), p') \wedge distance(p, p', i)$

**EP2 Complete observability**
If the agent is at position p, and a(p, p') is a visible state property involving p and a position p' within reachable distance, then this is observed by the agent. This property is to be applied to food, distance, accessibility, agent position, and the absence of these.
$\forall t\ \forall x\ \forall p\ \forall p'\ \forall i$
$[[\ i \leq max\_dist\ \&\ state(\gamma, t, environment) \models at(agent, p) \wedge a(p, p') \wedge visible(a(p, p')) \wedge$
$distance(p, p', i)\ ] \Rightarrow state(\gamma, t, input(agent)) \models observed(a(p, p'))]$

**EP3 Guaranteed effect of movement**
At any point in time t, if the agent goes to position p, then it will be at position p.
$\forall t\ \forall p\ state(\gamma, t, output(agent)) \models performing\_action(goto(p))$
$\Rightarrow\ state(\gamma, t+1, environment) \models at(agent, p)$

**EP4 Guaranteed effect of eating**
At any point in time t, if the agent takes food and the amount of food is sufficient for the agent then the agent will be well fed
$\forall t$ [[$\forall x$ state($\gamma$, t, output(agent))|= performing_action(take(food(x))) & $x \geq c$]
$\Rightarrow$ state($\gamma$, t+1, environment) |= agent_well_fed ]

**EP5 Reachability of environment**
The distances between all positions p in the agent's territory are smaller than max_dist. Here, p and p' are variables over the type TERRITORY_POSITION, which is a subtype of POSITION.
$\forall t \forall p \forall p' \forall I$ state($\gamma$, t, environment) |= distance(p, p', i) $\Rightarrow i \leq$ max_dist

The following environmental properties hold for the stimulus-response case and some of the other cases considered.

**EP6 Food persistence**
Food persists until taken by the agent.
$\forall t1 \forall t2 \forall x \forall p$ [ t1<t2 & state($\gamma$, t1, environment) |= at(food(x), p) &
[ $\forall t$ $t1 \leq t \leq t2 \Rightarrow$ state($\gamma$, t, output(agent)) |= not(performing_action(take(food(x)))) ]
$\Rightarrow$ state($\gamma$, t2, environment) |= at(food(x), p) ]

**EP7 Food on one position**
Per day, food only appears on one position.
$\forall n \forall x \forall p \forall p' \forall t$ state($\gamma$, n*d, environment) |= at(food(x), p) &
state($\gamma$, t, environment) |= at(food(x), p') & $n*d < t \leq (n+1)*d \Rightarrow p = p'$

**EP8 Complete accessibility**
Each position is accessible for the agent (i.e., never blocked by enemies).
$\forall t \forall p$ state($\gamma$, t, environment) |= accessible(p)

**EP9 Complete visibility**
All state properties a(p, p') that are true, are visible (which means that they will be observed by agents that are close enough, according to EP2). This property is to be applied to food, distance, accessibility, agent position, and the absence of these.
$\forall t \forall p \forall p'$ state($\gamma$, t, environment) |= a(p, p') $\Rightarrow$ state($\gamma$, t, environment(agent)) |= visible(a(p, p'))

Note that the property of an agent being well fed is assumed to be a state property of the environment, since it refers to the agent's body state.

For the case of stimulus-response behaviour the environment is characterised by the following conjunction ES1 of a subset of the environmental properties given above:

ES1 $\equiv$ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP8 & EP9

*Behavioural Properties*
The agent's stimulus-response behaviour is characterised by the following behavioural properties.

**BP1 Going to observed food**
At any point in time t, if the agent observes itself at position p and it observes no food at position p and it observes that an amount of food x is present at position p' and it observes that position p' is accessible and it observes that position p' is within reachable distance then it will go to position p'.
$\forall t \forall x \forall p \forall p'$ [ [ state($\gamma$, t, input(agent)) |= observed(at(agent, p)) $\wedge$ observed(not(at(food(x), p))) $\wedge$ observed(at(food(x), p')) $\wedge$ observed(accessible(p')) $\wedge$ observed(distance(p, p', i)) & $i \leq$max_dist ]
$\Rightarrow$ state($\gamma$, t+1, output(agent)) |= performing_action(goto(p')) ]

**BP2 Food uptake**
At any point in time t, if the agent observes itself at position p and the agent observes food at p then it will take the food
$\forall t \forall x \forall p$ [ [ state($\gamma$, t, input(agent)) |= observed(at(agent, p)) $\wedge$ observed(at(food(x), p))]
$\Rightarrow$ state($\gamma$, t+1, output(agent))|= performing_action(take(food(x))) ]

**Vitality property VP**
The animal gets sufficient food within any given day.
$\forall n \exists t1$ [ $n*d \leq t1 \leq (n+1)*d$ & state($\gamma$, t1, environment) |= agent_well_fed ]

*Logical relations*
Given the dynamic properties specified above, the *environmental* and *behavioural* specifications (in short, ES1 and BS1) for case 1 (stimulus-response behaviour) are as follows:

ES1 ≡ EP1 & EP2 & EP3 & EP4 & EP5 &EP6 & EP7 & EP8 & EP9
BS1 ≡ BP1 & BP2

Given these specifications, the question is whether they are logically related in the sense that this behaviour is adequate for this environment, i.e., whether indeed the following implication holds:

BS1 & ES1 $\Rightarrow$ VP

To automatically check such implications between dynamic properties at different levels, model checking techniques can be used. To this end, first the dynamic properties should be converted from TTL format to a finite state transition format. This can be done using an automated procedure, as described in [11]. After that, for checking the implications between the converted properties, the model checker SMV is appropriate (see URL: http://www.cs.cmu.edu/~modelcheck/smv.html; see also [8]). SMV has been used to verify (and confirm) the above implication, as well as a number of other implications shown in this paper.

Concerning the relation between the specification of the *cognitive* and the *behavioural* dynamics: in this case CS1 = BS1. Thus, CS1 $\Rightarrow$ BS1 also holds.

### 5.2 Delayed Response Behaviour

In delayed response behaviour, previous observations may have led to maintenance of some form of memory of the world state: a model or representation of the (current) world state (for short, *world state model*). This form of memory can be used at any point in time as an additional source (in addition to the direct observations). In that case, at a given time point the same input of stimuli can lead to different behavioural output, since the world state models based on observations in the past can be different. This makes that agent behaviours do not fit in the setting of an input-output correlation based on a direct functional association between (current) input states and output states. Viewed from an external viewpoint, this type of behaviour, which just like stimulus-response behaviour occurs quite often in nature, is just a bit more complex than stimulus-response behaviour, in the sense that it adds complexity to the temporal dimension by referring not only to current observations but also to observations that took place in the past.

This leads to the question what kind of complexity in the environment is coped with this kind of behaviour that is not coped with by stimulus-response behaviour. An answer on this question can be found in a type of environment with aspects which are important for the animal (e.g., food or predators), and which cannot be completely observed all the time; e.g., food or predators are sometimes hidden by other objects:

*Environmental properties*

For this case the environment described sometimes shows the food, but not always as in the previous case. It is characterised by the following conjunction ES2 of a subset of the environmental properties given above, extended with the properties EP10, EP11 and EP12 given below:

ES2 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP10 & EP11 & EP12

**EP10 Temporary visibility of food**
Per day, all food that is present is visible for at least one time point, and is accessible for at least one later time point[2].

**EP11 Complete visibility of non-food**
All state properties that are true, except the presence of food, are visible. Thus, this property is applied to distance, accessibility, and agent position.

**EP12 Complete local observability of food**
For all time points, if the agent is at the position p with food then the agent observes the food (no matter if it is visible, e.g., by smell)

*Behavioural properties*

Next, dynamic properties are identified that characterise the input-output correlation of delayed response behaviour, observed from an external viewpoint. Such a dynamic property has a temporal nature; it can refer to the agent's input and output in the present, the past and/or the future. In semi-formal and formal notation, for the case considered, the input-output correlation for delayed response behaviour can be characterised by:

**BP3 Going to food observed in the past**
At any point in time t, if the agent observes itself at position p and it observes no food at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and at some earlier point in time t1 the agent observed that an amount of food x was present at position p' and at every point in time t2 after t1 up to t, the agent did not observe that no food was present at p' then at the next time point after t the agent will go to position p'

$\forall t\ \forall x\ \forall i\ \forall p\ \forall p'$
$[\ [\ state(\gamma, t, input(agent)) \models observed(at(agent, p)) \wedge observed(not(at(food(x), p))) \wedge$
$observed(accessible(p')) \wedge observed(distance(p, p', i)) \ \&\ i \leq max\_dist\ ] \ \&$
$\exists t1 < t\ [state(\gamma, t1, input(agent)) \models observed(at(food(x), p')) \ \&$
$\forall t2\ [t \geq t2 > t1 \Rightarrow state(\gamma, t2, input(agent)) \models not(observed(not(at(food(x), p'))))]]$
$\Rightarrow state(\gamma, t+1, output(agent)) \models performing\_action(goto(p'))\ ]$

*Cognitive properties*

Since the external characterisations of delayed response behaviour refer to the agent's input in the past, it is assumed that internally the agent maintains past observations by means of persisting internal state properties, i.e., some form of memory. These persisting state properties are sometimes called *beliefs*. For the example case, it is assumed that an internal state property b1(p) is available, with the following dynamics:

**CP1 Belief formation on food presence**
At any point in time t, if the agent observes that food is present at position p then internal state property b1(p) will hold (i.e., a belief that food is present at p)

[2] Formal expressions for all properties can be found in the Appendix at http://www.cs.vu.nl/~tbosse/complexity.

**CP2 Belief b1 persistence**
At any point in time t, if internal state property b1(p) holds and the agent does not observe the absence of food at position p then at the next time point internal state property b1(p) still holds

**CP3 Going to food believed present**
At any point in time t, if the agent observes itself at position p and it observes no food at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and $p \neq p'$ and internal state property b1(p') holds then the agent will go to position p'

*Logical relations*

ES2 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP10 & EP11 & EP12
BS2 ≡ BP2 & BP3
CS2 ≡ BP2 & CP1 & CP2 & CP3
BS2 & ES2 ⇒ VP
CS2 ⇒ BS2

### 5.3 Goal-Directed Behaviour

A next, more complex type of behaviour considered is goal-directed behaviour. This behaviour is able to cope with environments where visibility can be more limited than in the environments considered before.

*Environmental properties*
For this case the environment is characterised by the following expression ES3 based on a subset of the environmental properties given earlier, extended with property EP13, given below:

ES3 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP11 & EP12 &
(EP10 OR (EP8 & EP13))

**EP13 Complete invisibility of food**
Food is always invisible for the agent (e.g., always covered), unless the agent is at the same position as the food.

*Behavioural properties*
The agent's behaviour exploring positions in order to discover food is characterised by the following behavioural property:

**BP4 Searching for food**
At any point in time t, if the agent observes itself at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and it did not visit position p' yet and p' is the position closest to p which the agent did not visit and it did not observe any food at all yet then at the next time point after t the agent will go to position p'

∀t ∀p ∀p'
state(γ, t, input(agent)) |= observed(at(agent, p)) ∧ observed(accessible(p')) ∧
observed(distance(p, p', i)) & i≤max_dist &
not [∃t' t'<t & state(γ, t', input(agent)) |= observed_at(agent, p') ] &
∀p'' [[not [∃t' t'<t & state(γ, t', input(agent)) |= observed_at(agent, p'') ]]
⇒ ∃d1 ∃d2 state(γ, t, input(agent)) |= observed(distance(p, p', d1)) ∧
observed(distance(p, p'', d2)) & d1<d2 ] &
not [∃t' ∃p'' ∃x t'≤t & state(γ, t', input(agent)) |= observed(at(food(x), p'')) ]
⇒ state(γ, t+1, output(agent)) |= performing_action(goto(p'))

*Cognitive properties*

To describe the internal cognitive process generating this type of behaviour, the mental state property *goal* is used. In particular, for the case addressed here, when the agent has no beliefs about the presence of food, it will generate the goal to find food. If it has this goal, it will pro-actively search for food in unexplored positions. This is characterised by the following dynamic properties:

**CP4 Goal formation**
At any point in time t, if the agent does not believe that food is present at any position p then it will have the goal to find food

**CP5 Non-goal formation**
At any point in time t, if the agent believes that food is present at position p then it will not have the goal to find food

**CP6 Belief formation on visited position**
At any point in time t, if the agent observes itself at position p then internal state property b2(p) will hold (i.e., the belief that it visited p)

**CP7 Belief b2 persistence**
At any point in time t, if internal state property b2(p) holds then at the next time point internal state property b2(p) still holds

**CP8 Belief formation on distances**
At any point in time t, if the agent observes that the distance between position p and p' is d then internal state property belief(p, p', d) will hold

**CP9 Belief persistence on distances**
At any point in time t, if internal state property belief(p, p', d) holds then at the next time point internal state property belief(p, p', d) still holds

**CP10 Going to closest position**
At any point in time t, if the agent observes itself at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and it has the goal to find food and it believes it did not visit p' yet and p' is the position closest to p of which the agent believes it did not visit it then at the next time point after t the agent will go to position p'

*Logical relations*

ES3 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & EP6 & EP7 & EP11 & EP12 &
(EP10 OR (EP8 & EP13))
BS3 ≡ BP2 & BP3 & BP4
CS3 ≡ BP2 & CP1 & CP2 & CP3 & CP4 & CP5 & CP6 & CP7 & CP8 & CP9 & CP10
BS3 & ES3 ⇒ VP
CS3 ⇒ BS3

### 5.4 Learning Behaviour

A final class of behaviour analysed is learning behaviour. In this case, depending on its history comprising a (possibly large) number of events, the agent's externally observable behaviour is tuned to the environment's dynamics. In the case addressed here, in contrast to the earlier cases, the environment has no guaranteed persistence of food for all positions. Instead, at certain positions food may come and go (e.g., because it is eaten by competitors). The agent has to learn that, when food often

appears (and disappears) at a certain position, then this is an interesting position to be, because food may re-appear at that position (but soon disappear again).

*Environmental properties*

For this case the environment is characterised by the following expression ES4 based on a subset of the environmental properties given earlier, extended with property EP14, given below.

ES4 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & ((EP6 & EP7 & EP10 & EP11 & EP12)
OR (EP6 & EP7 & EP8 & EP11 & EP12 & EP13) OR (EP9 & EP14))

**EP14 Food reoccurrence**

Every piece of food disappears and reappears at least 2 times per day, of which at least after the second disappearance its position will be accessible.

*Behavioural properties*

The agent's behaviour for this case should take into account which positions show reoccurence of food. The following behavioural property characterises this.

**BP5 Being at useful positions**

At any point in time t, if the agent observes itself at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and for all positions p'' that the agent observed food in the past, the agent later observed that the food disappeared and at some earlier point in time t1 the agent observed that food was present at position p' and after that at time point t2 before t the agent observed no food present at position p' and after that at time point t3 before t the agent again observed the presence of food at position p' and after that at a time point t4 before t the agent again observed no food present at position p' and p' is the closest reachable position for which the above four conditions hold then at the next time point after t the agent will go to position p'

∀t ∀p ∀p' ∀x
state(γ, t, input(agent)) |= observed(at(agent, p)) ∧
observed(accessible(p')) ∧ observed(distance(p, p', i)) & i≤max_dist &
∀t' ∀p'' ∀x' [t'<t & state(γ, t', input(agent)) |= observed(at(food(x'), p''))
⇒ ∃t'' t'<t''≤t &
state(γ, t'', input(agent)) |= observed(not(at(food(x'), p'')))]
& ∃t1 ∃t2 ∃t3 ∃t4 [ t1<t2<t3<t4<t &
state(γ, t1, input(agent)) |= observed(at(food(x), p')) &
state(γ, t2, input(agent)) |= observed(not(at(food(x), p'))) &
state(γ, t3, input(agent)) |= observed(at(food(x), p')) &
state(γ, t4, input(agent)) |= observed(not(at(food(x), p'))) ]
& ∀p'' [ ∃t1 ∃t2 ∃t3 ∃t4 [ t1<t2<t3<t4 &
state(γ, t1, input(agent)) |= observed(at(food(x), p'')) &
state(γ, t2, input(agent)) |= observed(not(at(food(x), p''))) &
state(γ, t3, input(agent)) |= observed(at(food(x), p'')) &
state(γ, t4, input(agent)) |= observed(not(at(food(x), p''))) ] ⇒
∃d1 ∃d2
state(γ, t, input(agent)) |= observed(distance(p, p', d1)) ∧
observed(distance(p, p'', d2)) & d1<d2 ]
⇒ state(γ, t+1, output(agent)) |= performing_action(goto(p'))

*Cognitive properties*

The internal cognitive dynamics has to take into account longer histories of positions and food (re)appearing there. This is realised by representations that are built up for more complex world properties, in particular, not properties of single states but of histories of states of the world. For example, at a certain time point, it has to be represented that for a certain position in the past food has appeared twice and in

between disappeared. The state properties b3(p, q) play the role of representations of world histories on food (re)occurrence.

**CP11 Initial mental state**
At the beginning of every day n, for all positions p, internal state property b3(p, 0) holds (i.e. a belief that there is no food at p)

**CP12 Belief update on food presence**
At any point in time t, for $q \in \{0,2\}$, if internal state property b3(p, q) holds and the agent observes food at position p then internal state property b3b(p, q+1) will hold

**CP13 Belief update on food absence**
At any point in time t, for $q \in \{1,3\}$, if internal state property b3(p,q) holds and the agent observes no food at position p then internal state property b3(p,q+1) will hold

**CP14 Belief b3 persistence**
At any point in time t, for all q, if internal state property b3(p,q) holds then at the next time point internal state property b3(p,q) still holds

**CP15 Going to interesting position**
At any point in time t, if the agent observes itself at position p and it observes that position p' is accessible and it observes that position p' is within reachable distance and it has the goal to find food and p' is the position closest to p of which the agent believes that it is an attractive position then at the next time point after t the agent will go to position p'

Here, b3(p,4) represents the belief that food was twice present at p, and subsequently disappeared (in other words, a belief that p is an attractive position, since food might show up again). Note that, although the mechanism described here is quite different from, e.g., machine learning, this type of behaviour nevertheless can be qualified as learning behaviour. The reason for this is that the behaviour can be split into two distinct phases: one in which nothing was learned, and one in which the agent has learned which positions are useful by maintaining a history of previous observations.

*Logical relations*

ES4 ≡ EP1 & EP2 & EP3 & EP4 & EP5 & ((EP6 & EP7 & EP10 & EP11 & EP12)
OR (EP6 & EP7 & EP8 & EP11 & EP12 & EP13) OR (EP9 & EP14))
BS4 ≡ BP2 & BP3 & BP4 & BP5
CS4 ≡ BP2 & CP1 & CP2 & CP3 & CP4 & CP5 & CP6 & CP7 & CP8 & CP9 & CP10 &
CP11 & CP12 & CP13 & CP14 & CP15
BS4 & ES4 $\Rightarrow$ VP
CS4 $\Rightarrow$ BS4

## 6 Formalisation of Temporal Complexity

The Complexity Monotonicity Thesis discussed earlier involves environmental, behavioural and cognitive dynamics of living systems. In Section 2 it was shown that based on a given complexity measure cm this thesis can be formalised by:

$$cm(E_1) \leq cm(E_2) \Rightarrow cm(B_1) \leq cm(B_2) \;\&$$
$$cm(B_1) \leq cm(B_2) \Rightarrow cm(C_1) \leq cm(C_2)$$

What remains is the existence or choice of the complexity measure function cm. To measure degrees of complexity for the three aspects considered, a temporal perspective is chosen: complexity in terms of the temporal relationships describing

them. For example, if references have to be made to a larger number of events that happened at different time points in the past, the temporal complexity is higher. The temporal relationships have been formalised in the temporal language TTL based on predicate logic. This translates the question how to measure complexity to the question how to define complexity of syntactical expressions in such a language. In the literature an approach is available to define complexity of expressions in predicate logic in general by defining a function that assigns a *size* to every expression [7]. To measure complexity, this approach was adopted and specialised to the case of the temporal language TTL. Roughly spoken, the complexity (or size) of an expression is (recursively) calculated as the sum of the complexities of its components plus 1 for the composing operator. In more details it runs as follows.

Similarly to the standard predicate logic, predicates in the TTL are defined as relations on terms. The size of a TTL-term t is a positive natural number s(t) recursively defined as follows:

(1) $s(x)=1$, for all variables x.

(2) $s(c)=1$, for all constant symbols c.

(3) $s(f(t1,\ldots, t_n))= s(t1) + \ldots + s(t_n) + 1$, for all function symbols f.

For example, the size of the term observed(not(at(food(x), p))) from the property BP1 (see the Appendix) is equal to 6.

Furthermore, the size of a TTL-formula $\psi$ is a positive natural number $s(\psi)$ recursively defined as follows:

(1) $s(p(t_1,\ldots, t_n))= s(t_1) + \ldots + s(t_n) +1$, for all predicate symbols p.

(2) $s(\neg\varphi)=s((\forall x)\ \varphi)= s((\exists x)\ \varphi) = s(\varphi)+1$, for all TTL-formulae $\varphi$ and variables x.

(3) $s(\varphi\&\chi) = s(\varphi|\chi) = s(\varphi\Rightarrow\chi) = s(\varphi)+ s(\chi)+1$, for all TTL-formulae $\varphi$, $\chi$.

In this way, for example, the complexity of behavioural property BP1 amounts to 53, and the complexity of behavioural property BP2 is 32. As a result, the complexity of the complete behavioural specification for the stimulus-response case (which is determined by BP1 & BP2) is 85.

Using this formalisation of a complexity measure as the size function defined above, the complexity measures for environmental, internal cognitive, and behavioural dynamics for the considered cases of stimulus-response, delayed response, goal-directed and learning behaviours have been determined. Table 1 provides the results (see the Appendix for all properties).

**Table 1.** Temporal complexity of environmental, behavioural and cognitive dynamics.

| Case | Environmental dynamics | Behavioural dynamics | Cognitive dynamics |
|---|---|---|---|
| Stimulus-response | 262 | 85 | 85 |
| Delayed response | 345 | 119 | 152 |
| Goal-directed | 387 | 234 | 352 |
| Learning | 661 | 476 | 562 |

The data given in Table 1 confirm the Complexity Monotonicity Thesis put forward in this paper, that the more complex the environmental dynamics, the more complex the types of behaviour an agent needs to deal with the environmental complexity, and the more complex the behaviour, the more complex the internal cognitive dynamics.

## 7 Discussion

In this paper, the temporal complexity of environmental, behavioural, and cognitive dynamics, and their mutual dependencies, were explored. As a refinement of Godfrey-Smith's Environmental Complexity Thesis [4], the Complexity Monotonicity Thesis was formulated: for more complex environments, more complex behaviours are needed, and more complex behaviours need more complex internal cognitive dynamics. A number of example scenarios were formalised in a temporal language, and the complexity of these formalisations was measured. Complexity of environment, behaviour and cognition was taken as temporal complexity of dynamics of these three aspects, and the formalisation of the measurement of this temporal complexity was based on the complexity of the syntactic expressions to characterise these dynamics in a predicate logic language, as known from, e.g., [7]. The outcome of this approach is that the results support the Complexity Monotonicity Thesis.

Obviously, the results as reported in this paper are no generic proof for the correctness of the Complexity Monotonicity Thesis. Instead, the paper should rather be seen as a case study in which the thesis was tested positively. However, the approach taken for this test was not completely arbitrary: the used complexity measure is one of the standard approaches to measure complexity of syntactical expressions [7]. Moreover, the formal specifications were constructed very carefully, to ensure that no shorter specifications exist that are equivalent. Although no formal proof is given that the used specifications are indeed the shortest possible ones, the construction of these specifications has been an iterative process in which multiple authors have participated. To represent the specifications, the language TTL was just used as a vehicle. Various similar temporal languages could have been used instead, but we predict that this would not significantly influence the results.

Nevertheless, there are a number of alternative possibilities for measuring complexity that might in fact influence the results. Among these is the option to use complexity measures from information theory based on the amount of entropy of a system, such as [1]. In future work, such alternatives will be considered as well. Another challenging direction for future work is the possibility to establish a uniform approach for specification of dynamic properties for environment, behaviour, and cognition. Such an approach may, for example, prescribe a limited number of predefined concepts that can be used within the dynamic properties.

Another issue that is worth some discussion is the fact that the Complexity Monotonicity Thesis can also be considered in isolation of Godfrey-Smith's Environmental Complexity Thesis. Although it was used as a source of inspiration to explore for the more refined Complexity Monotonicity Thesis, the Environmental Complexity Thesis as such was not investigated in this paper. Doing this, again from an agent-based modelling perspective, is another direction for future work. To this end, techniques from the area of Artificial Life may be exploited, e.g., to perform social simulations and observe whether more complex agents evolve in a way that supports the Environmental Complexity Thesis.

In [4], in particular in Chapters 7 and 8, mathematical models are discussed to support the Environmental Complexity Thesis, following, among others [9] and [12]. These models are made at an abstract level, abstracting from the temporal dimension of the behaviour and the underlying cognitive architectures and processes. Therefore,

the more detailed temporal complexity as addressed in this paper is not covered. Based on the model considered, Godfrey-Smith [4] concludes that the flexibility to accommodate behaviour to environmental conditions, as offered by cognition, is favoured when the environment shows (i) unpredictability in distal conditions of importance to the agent, and (ii) predictability in the links between (observable) proximal and distal. This conclusion has been confirmed to a large extent by the formal analysis described in this paper. Comparable claims on the evolutionary development of learning capabilities in animals are made in work such as [13] and [10]. According to these authors, learning is an adaptation to environmental change. All these are conclusions at a global level, compared to the more detailed types of temporal complexity considered in our paper, where cognitive processes and behaviour extend over time, and their complexity can be measured in a more detailed manner as temporal complexity of their dynamics.

## References


1. Berlinger, E. (1980). An information theory based complexity measure. In Proceedings of the Natural Computer Conference, pp. 773-779.
2. Bosse, T., Jonker, C.M., Meij, L. van der, Sharpanskykh, A., and Treur, J. (2006). Specification and Verification of Dynamics in Cognitive Agent Models. In: *Proceedings of the Sixth International Conference on Intelligent Agent Technology, IAT'06*. IEEE Computer Society Press, 2006, pp. 247-254.
3. Darwin, C. (1871). *The Descent of Man*. John Murray, London.
4. Godfrey-Smith, P., (1996). *Complexity and the Function of Mind in Nature*. Cambridge University Press.
5. Hills, T.T. (2006). Animal Foraging and the Evolution of Goal-Directed Cognition. *Cognitive Science*, vol. 30, pp. 3-41.
6. Hunter, W.S. (1912). The delayed reaction in animals. *Behavioral Monographs*, 2, 1912, pp. 1-85
7. Huth, M. and Ryan, M. (2000). Logic in Computer Science: Modelling and reasoning about computer systems, Cambridge University Press.
8. McMillan, K.L. (1993). Symbolic Model Checking: An Approach to the State Explosion Problem. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1992. Published by Kluwer Academic Publishers, 1993.
9. Moran, N. (1992). The evolutionary maintenance of alternative phenotypes. *American Naturalist*, vol. 139, pp. 971-989.
10. Plotkin, H. C. and Odling-Smee, F. J. (1979). Learning, Change and Evolution. *Advances in the Study of Behaviour 10*, pp. 1-41.
11. Sharpanskykh, A., Treur, J. (2006). Verifying Interlevel Relations within Multi-Agent Systems. In: *Proceedings of the 17th European Conference on Artificial Intelligence, ECAI'06*. IOS Press, 2006, pp. 290-294.
12. Sober, E. (1994). The adaptive advantage of learning versus a priori prejustice. In: *From a Biological Point of View*. Cambridge University Press, Cambridge.
13. Stephens, D. (1991). Change, regularity and value in evolution of animal learning. *Behavioral Ecology*, vol. 2, pp. 77-89.
14. Tinklepaugh, O.L. (1932). Multiple delayed reaction with chimpanzees and monkeys. *Journal of Comparative Psychology*, 13, 1932, pp. 207-243.
15. Vauclair, J. (1996). *Animal Cognition.* Harvard Univerity Press, Cambridge, MA.
16. Wilson, O. (1992). *The Diversity of Life*. Harvard University Press, Cambridge, MA.

# Component-Based Standardisation of Agent Communication

Frank Guerin and Wamberto Vasconcelos

Dept. of Computing Science, Univ. of Aberdeen, Aberdeen AB24 3UE, UK
{fguerin, wvasconc}@csd.abdn.ac.uk

**Abstract.** We address the problem of standardising the semantics of agent communication. The diversity of existing approaches suggests that no single agent communication language can satisfactorily cater for all scenarios. However, standardising the way in which different languages are specified is a viable alternative. We describe a standard meta-language in which the rules of an arbitrary institution can be specified. In this way different agent communication languages can be given a common grounding. From this starting point, we describe a component based approach to standardisation, whereby a standard can develop by adding component sets of rules; for example to handle various classes of dialogs and normative relations. This approach is illustrated by example. Eventually we envisage different agent institutions publishing a specification of their rules by simply specifying the subset of standard components in use in that institution. Agents implementing the meta-language can then interoperate between institutions by downloading appropriate components.

## 1 Introduction

We are interested in facilitating interoperability for agents interacting with different institutions on the Internet. For example, consider a personal agent of a professor who is invited to participate in a conference (say to give a keynote address and chair a session). The personal agent may connect with the conference site and enter a collaborative dialogue with the agents of the various other speakers, and the conference organiser, in order to arrange the schedule of events. Subsequently the agent will connect to various online travel sites to procure airline tickets and accommodation, most likely by means of some auction mechanism. Finally the agent may discover that an airline ticket it has bought does not conform to what was advertised, thus it may seek compensation, lodging an appeal with some arbitration site, and bringing evidence to support the claim. Each of these interactions occurs in a different institution; the requirements for the agent communication language (ACL) in each institution are quite different. Yet, would it be possible to provide a standard language which ecompasses all requirements?

Past attempts to standardise agent communication [8, 15, 7][1] have managed to standardise certain syntactic or pragmatic aspects, but fared poorly when it comes to the issue of the semantics of the communication language. In practice, implementers who claim to be using a particular standard ACL tend to ignore those aspects of the standard that pose difficulties for their implementation

[1] Note that the FIPA'97 specification is cited here because the communication language semantics has not changed since then.

(often the formal semantics are ignored); additionally they often create ad hoc extensions when none of the constructs of the standard quite fits their needs. Effectively they invent their own custom dialect, which will not be understood by any other system [22]. Given the diverse needs of different domains, it is probably not feasible to come up with a single standard ACL which will cater for the needs all possible agent systems. Furthermore, a standard ACL would be rigid, precluding the possibility of agents negotiating the semantics of their own custom ACL on the fly, for a particular purpose. The ACL would seem to be the wrong level to standardise at; instead, it would seem appropriate to have a standard way of specifying semantics, to allow developers (or agents themselves) to create their own languages in a standard and structured way. Our proposal is to create a standard meta-language which would allow different interaction languages to be defined for different domains.

The core language, on which developers will build, must be sufficiently expressive to allow any reasonable language to be specified. For this purpose we identify a class of agent communication languages which are universal in the sense that they could be used to simulate any other agent communication language which is computable. We specify one such language and demonstrate its generality by showing how it allows the specification of institutions in which agents can change the rules of the institution itself. With this core in place, we envisage a standard evolving gradually by adding "components", by which we mean a set of rules to govern a certain type of interaction (i.e. a component encodes a protocol), or to make available useful abstractions (e.g. a component provides rules for normative relations and further components can use these to specify high-level protocols). In this way we can give developers the flexibility to define their own components, and publish the specifications, so that others can develop further components, and agents, to work with that language. It is hoped that this could bring together the efforts of the community as similar efforts have done in other programming languages. A further advantage of the component based approach is that all agents in a society do not necessarily need to support the same components. Some agents may be less sophisticated than others and may support simple reactive protocol components, while other more sophisticated agents may be able to use components which allow them to express their intentions and desires, with a well defined meaning. When agents wish to communicate they would firstly discover which components they support in common, and then they can determine the level at which they can interact. This ability to implement lightweight versions of an agent communication language is one of the desiderata for agent communication languages outlined by Mayfield *et al.* [19].

In this paper we will illustrate the proposed approach with some examples. We must stress that we are not advocating that the components described in this paper be adopted as a standard; we merely provide simple examples to demonstrate the feasibility of the component-based approach. This paper will focus exclusively on the semantic issues as these have proved to be the most problematic for the standardisation of agent communication. We therefore ignore all pragmatic issues, e.g. how to find the name of an agent who provides some service, authentication, registration, capability definition and facilitation [16]. We assume an agent platform which can take care of all these issues. Pragmatic

issues are of course important, but they would require a full treatment in their own right.

This paper is organised as follows. Section 2 looks at the most general framework within which all practical ACLs could be specified. Section 3 defines an agent communication language which allows unlimited extensions, and so forms the base component which we later build on. Section 4 adds a component for normative relations. Section 5 discusses how protocols can be added in general, and adds an auction protocol. Section 6 describes a temporal logic component we have added. Section 7 discusses related work and Section 8 concludes with a look to the future.

## 2 Definition of an Agent Communication Language

In this section we want to define what an ACL is in the most general terms, and to have a formal framework which captures the space of possible ACLs. Following Singh's seminal work on social commitments [22, 23], there does seem to be a consensus in the community that the semantics of communication for open systems should be based on social phenomena rather than agents' private mental states [25, 9, 4, 5]. We follow this social approach and we consider that all "reasonable" languages for use in an open system must be of the social type. We do not restrict ourselves to commitments: we allow arbitrary social facts[2].

We define an ACL by specifying an institution. The existence of a communication language presumes the existence of an institution, which defines the meaning of language. Institutions are man-made social structures created to regulate the activities of a group of people engaged in some enterprise. They may be created deliberately, as is the case for formal organisations, or they may be created by conventions evolving over time, as is the case for human culture. Institutions regulate the activities of their members by inventing *institutional facts*, a term due to Searle [21]. Some institutional facts take the form of *rules* while others merely describe a *state of affairs* in the institution. *Rules* describe how institutional facts can be created or modified. An example of an institutional fact of the *state of affairs* type is having the title "doctor"; examples of institutional facts which are rules are the rules in a University which describe how the title can be awarded and by whom. The *rule* type of facts can be used to provide a relationship between the real physical world and the institution; rules can have preconditions which depend on the physical world and/or on other institutional facts. For example, the submission of a thesis physically bound in a specified format is a necessary precondition to the awarding of the title "doctor"; the passing of an examination (a purely institutional fact) is another precondition.

Rules relating to the physical world describe how events or states of the world (typically the actions of members) bring about changes in the institutional facts. The classic example of this is where an utterance by a member of an institution can bring about an institutional fact, for example the naming of a ship: "I hereby name this ship the Queen Elizabeth." [3]. It is not possible for institutional

[2] This means that we are not precluded from representing mental states that have been publicly expressed by an agent [12], the difference betweeen this and earlier mentalistic semantics [8, 15, 7] is that we do not require that agents actually hold the mental states which they have publicly expressed.

facts to bring about changes in the physical world because the institution itself, being a collection of intangible institutional facts, cannot directly effect any physical change in the world.[3] Institutions may describe their rules in a form which specifies physical effects in the world, but such rules are not strictly true because the physical effects are not guaranteed to happen; the only way in which the institution can influence the actions of its members is through the threat of further institutional facts being created. For example a legal institution may prescribe a term of imprisonment as the consequence of an institutional fact, but it cannot directly bring about the imprisonment of a member; instead it can state that a policeman should use physical force to bring the member to prison, and the policeman can be threatened with the loss of his job if he does not. The rule prescribing imprisonment can be reformulated as a rule which states that if the policeman does not imprison the member by physical force or otherwise, then the policeman loses his job. Thus all the rules relating to the physical world take the form of descriptions of how events or states of the world bring about changes in the institutional facts.

A further point to note is that the institutional facts being modified by a rule could be rules themselves. Many institutions do modify their rules over time; a legal institution may allow arguments about the rules by which argumentation should take place. This is accommodated by the framework described above, because a rule can modify an institutional fact, and that institutional fact could be another rule.

If we assume that any relevant change in the world's state can be translated into an event, then we can say (without loss of generality) that the instutional facts change only in response to events in the world (we do not allow rules to refer to states of the world). In a typical agent system we rely on the agent platform to handle the generation of events. Typical events include messages being transmitted, timer events and possibly other non communicative actions of agents or events such as agent death. Let $E$ be the set of possible events and let $\mathcal{F}$ be the set of possible institutional facts. Let *update* be a function which updates the institutional facts in response to an event; $update : E \times 2^{\mathcal{F}} \rightarrow 2^{\mathcal{F}}$. Now in an institution $\mathcal{I}$, it is the institutional rules $R$ which indirectly define this update function. The institution interprets the rules in order to define the update function, let the interpreter function be $I$, where $I$ maps $R$ to some update function. An institution $\mathcal{I}$ can then be fully specified by specifying the interpreter $I$ and the facts $F \subset \mathcal{F}$. $F$ is itself composed of the *rule* type of facts $R$ and the *state of affairs* type of facts $A$, so $F = \langle R, A \rangle$. Therefore institution $\mathcal{I}$ can be represented by a tuple $\langle I, F \rangle$. The $F$ component fully describes the facts and rules which currently hold in the institution.

This gives us the most general view of agent communication languages; by specifying the tuple $\langle I, F \rangle$ we can specify any ACL. It describes how institutional

[3] Institutions may indirectly affect the physical world if agents of the institution take physical actions in response to institutional facts. We consider a bank balance to be an institutional fact; it happens that banks have implemented physical agents which act on this institutional fact and dispense money. Neverless, the bank balance itself is not a physical fact. Likewise, if certain institutional facts are valued or feared by agents, then they will act in response to them (hence the institutional facts affect the physical world only through the agents).

facts $F$ change in response to events as the multi-agent system runs. Given an institution described by $\langle I, F_0 \rangle$ at some instant, and a subsequent sequence of events $e_1, e_2, e_3 \ldots$, we can calculate the description of the institutional facts after each event, obtaining a sequence of facts descriptions: $F_0, F_1, F_2, \ldots$, where each $F_i$ is related to $F_{i-1}$ as follows: $F_i = update_{i-1}(e_i, F_{i-1})$ where $update_{i-1} = I(R_{i-1})$ (and $F_i = \langle R_i, A_i \rangle$ for all $i$). Interpreter $I$ remains fixed throughout runs.

### 2.1 A Universal Agent Communication Machine

The rule interpreter $I$ specified above is the immutable part of an institution. The choice of $I$ can place limits on what is possible with that institution, or give it universal expressive power. Just as a universal Turing machine can simulate the behaviour of any turing machine, we can have an analogous universal agent communication machine.

**Definition 1.** *A universal agent communication machine is a machine which can simulate the behaviour of any computable agent communication language.*

By "simulate" here we mean that (given an appropriate set of input rules) it could generate the same sequence of institutional facts in response to the same sequence of events. In fact a universal Turing machine is a universal agent communication machine. The input $R$ to the machine produces the function *update*. Any update function that is computable can be produced in this way. Any Turing complete programming language can be used to implement a universal agent communication machine.

## 3 Specifying Extensible Languages

Given a universal agent communication machine it is possible to specify an ACL which has *universal expressive power*, in the following sense.

**Definition 2.** *An agent communication language is said to have universal expressive power if the agents using it can transform its rules so that it simulates the behaviour of any computable agent communication language.*

Given a language defined by an institutional specification $\mathcal{I} = \langle I, \langle R, A \rangle \rangle$ (as described above), if $I$ is a universal agent communication machine, then the language will have universal expressive power if the rules $R$ allow messages sent (i.e. events) to program the machine in a Turing complete way.[4] Languages with universal power are of particular interest because they allow unlimited extension. It is our thesis that a minimal language with universal expressive power is an appropriate basis for standardising agent communication; i.e. the specification of the programming language and core code can be agreed upon and published. Such a choice of standard does not restrict agents to the rules given because it can provide a mechanism through which agents can change the rules at runtime; this can allow agents to introduce new protocols at runtime, for example. Such protocols could come from trusted libraries, or could be generated by the agents on the fly for the scenario at hand. If necessary, agents could also have a phase of negotiation before deciding on accepting some new rules.

[4] This expressiveness implies undecidability, but if desired one can specify a restricted and decidable language on top of this, by restricting the agents' ability to modify rules, as described in the sequel.

```
1  interpretEvent(F,Event,NewF):-
2     F=[Rules,Asserts],
3     Event=..EventAsList,
4     append(EventAsList,[F,NewF],NewEventAsList),
5     Pred=..NewEventAsList,
6     copy_term(Rules,Rules2),
7     member([_|[Pred|Tail]],Rules2),
8     callPred(Tail,Rules).

9  callPred([],_).
10 callPred([HeadPred|Tail],Rules):-
11    copy_term(Rules,Rules2),
12    member([_|[HeadPred|NestTail]],Rules2),
13    callPred(NestTail,Rules),
14    callPred(Tail,Rules).
15 callPred([HeadPred|Tail],Rules):-
16    call(HeadPred),
17    callPred(Tail,Rules).
```

Fig. 1: Extensible Communication Language in Prolog

We define one such language in Fig. 1. We make use of Prolog as the logic programming paradigm is particularly appropriate for agent communication; there is also evidence that Prolog already enjoys considerable popularity in the agent communication semantics community [1, 20, 14]. The `interpretEvent`/3 predicate takes as input the current set of facts `F` and an event `Event`, and generates the new set of facts `NewF`. In line *3* the event is converted from its predicate form to a list form, so that line *4* can append the old and new facts variables to it. In line *5* the event is converted back from list form to its predicate form. The next step will be to match the head of the event with the appropriate rule in `Rules` (this corresponds to $R$ in the formal model); however, we do not want to change the rule itself by unifying its variables, this is why we make a clean copy of it in line *6* before doing the matching in line *7*, via the `member`/2 predicate. Now that the body of the rule (`Tail`) has been retrieved, we can invoke it in line *8* via `callPred/2`. Lines *9* to *17* define the recursive `callPred/2` predicate. Line *10* handles the case where the rule body to be executed invokes another rule within `Rules`, in which case `callPred/2` is called to handle it. Line *15* handles the case where the rule body to be executed invokes a built in Prolog predicate, in which case it is called directly via `call/1`. It is important that `interpretEvent`/3 forces the event to use a rule from `Rules` (i.e. it checks that the rule is a member of `Rules` before passing control to `callPred/2`) so that agents are unable to directly invoke Prolog predicates with their messages; their messages are interpreted first. Without this precaution our interpreter would not truly have universal expressive power, as it would always accept Prolog predicates, which could be used to reprogram it; hence it would be impossible to define a language which restricted the possible things which events could change.

Rules stored in $R$ are written in the form of lists, with an index number at the head of each rule. A Prolog clause of the form "`pred1(A,B):-pred2(A), pred3(B)`" becomes "`[1,pred1(A,B),pred2(A),pred3(B)]`". This corresponds to the Horn clause $pred2(A) \wedge pred3(B) \rightarrow pred1(A, B)$. Some sample rules are:

```
[  [ 1, addRule(Rule,[R1,A1],[NewR1,A1]),
       append(R1,[Rule],NewR1)                ],
   [ 2, deleteRule(Index,[R2,A2],[NewR2,A2]),
       delete(R2,[Index|_],NewR2)             ]
]
```

Let the above program be called *prog* and the interpreter $I = \langle prog, Prolog \rangle$. Let the assertions $A$ be initially empty and rules $R$ be the two rules above.

**Theorem 1.** *The ACL specified by institution* $\langle I, \langle R, A \rangle \rangle$ *has universal expressive power.*

The truth of this follows from that fact that Prolog is Turing-complete, and `addRule` can be used to add arbitrary predicates, and can therefore give subsequent events access to the underlying Prolog language (or restrict their access). Despite the ease with which this can be done, to our knowledge this is the first example of such an ACL. We propose that an ACL such as this would form the core component of a standard. This is only the first step of standardisation however. Standards will also need to define libraries and tools which will make the base machine more usable.

Let us briefly illustrate how we can begin to use the above ACL. The following is an example of an event:

```
addrule([3,assert(Fact,[R,A],[R,[Fact|A]])])
```

After interpreting this event, the rules $R$ will be updated so that subsequent `assert` events cause the addition of an element to the assertions $A$. For example, a subsequent event `assert(alive(agent1))` would add `alive(agent1)` to $A$. Note that this is invoking our rule 3 and not Prolog's built-in `assert`/1 predicate. At this point we will avoid giving an extended example of the kind of interaction we can capture. Instead we want to show the component based approach to standardisation, so we will eventually illustrate only a very simple auction protocol, but we will build it upon some useful components.

We now add some basic "housekeeping" rules. We will have a *timer* predicate in $A$, which records the current time, e.g. *timer(524)*. We will assume that our agent platform generates timer events at regular intervals. Whenever a timer event happens we want to update the clock and execute a set of housekeeping rules. These rules perform housekeeping checks, for example to see if an agent has failed to meet a deadline. The following rule (in $R$) handles the timer event:

```
[ 3, timer(Time,[R,A],[NewR,NewA]),
     replace(A,timer(Time),UpdatedA),
     housekeeping([R,UpdatedA],[NewR,NewA])
]
```

Here we have assumed the existence of a *replace* predicate which replaces a named predicate in $A$ with a new version. The initial *housekeeping* predicate simply preserves the institutional facts $F$; subsequent components will modify the predicate, adding their own rules.

It is desirable to add another layer for the interpretation of agent communications. We create a *speechAct* rule for this purpose. Agents communicate by sending messages (events) of the form *speechAct(sender, receiver, performative,*

*content)*. We must rely on the platform's message handling layer to discard any messages where the *sender* parameter is falsified; there is no way to do this once the event is processed by the interpreter. We also rely on the platform to distribute the message to the intended recipients. The message event is then handled by our *speechAct* rule. With this in place we protect the lower level operations from direct access by the agents. We do not want agents to be able to directly invoke the timer event or the rule changing events; however, if desired, we can still create speech acts which allow the modification of certain rules in $R$ by suitably empowered agents. Now the *speechAct* predicate becomes particularly useful to gather together all those operations which need to be done during the processing of any message (e.g. check roles, permissions and empowerments). This is described in Section 4.

It is worth noting that the update rule we have been describing needs to have access to all events in the system in order to build a complete picture of the social facts. If each agent is applying the update rule using only the events they have observed, they only build a view of the institutional facts from their individual perspectives. If agents apply the rules on limited information in this way, it is entirely possible that the institutional facts from the perspective of two different agents may have contradictory assertions. This is not a problem, so long as the developer bears this in mind when designing components (protocols for example). Specifications of norms should not create "unfair" rules, for example creating an obligation for an agent to do something, and leaving the agent unaware of the existence of the obligation. In most practical systems which we envisage, there will be no need for any agent to maintain this global view and indeed in a large system it might not be feasible to maintain it; it will be sufficient for each agent to maintain an individual perspective, which coincides with the perspective of other agents for any interactions they share.

Obviously we need to be particularly careful if we allow agents to change the rules $R$, lest conversational participants have contradictory beliefs about the meanings of the messages they are exchanging. At least two solutions can be envisaged: either all members of the institution need to be informed of any change, or a subgroup can decide to set up a virtual organisation (having new communicative actions and corresponding rules which apply only within that virtual organisation, the old ones still applying outside).

## 4 Normative Relations Component

Various different notions are employed by institutions to describe their rules; Sergot distinguishes between notions of power, permission and practical possibility [13]. Power is the ability of a member to bring about changes in the institutional facts; i.e. for each event which changes $F$ we can describe which members of the institution can effect those changes. For convenience it is common to define roles and define the power of a role. This is because the occupants of roles often change, while the powers associated with the role do not.

Permission can be used to describe those actions which ought or ought not to be taken. Permission is distinct from power because a member may be empowered to do something even though he is not permitted; in this case: if he does it then it counts, i.e. it creates the institutional fact. For example, an examiner could award a student a pass on submission which falls short of the required

standards as set out by the institution. In this case the examiner's action is not permitted but still counts as a pass under the rules of the institution; the examiner may be subject to some sanction if the abuse is discovered, but this may not necessarily revoke the fact that the student has passed.

The notion of permission leads to its dual: obligation; obligation is equivalent to "not permitted not to". Obligation can be captured by a rule which specifies a sanction if an action is not done. Because we will be testing agents' compliance over finite models, we must always specify a time-limit for obligations. It is no good for an agent to promise something and deliver "eventually", if there is no upper bound on the time taken.

Practical possibility is another distinct notion which some institutions may need to represent explicitly. For example, suppose there is a rule defining the sanction to be placed on a member in the case of failing to fulfil an obligation, there may be a need to exempt the case where the member was physically incapable of fulfilling the obligation at the time. Thus there could be institutional facts to represent the physical capabilities of each agent; i.e. a rule will define the events in the physical world which *count as* the agent being recognised by the institution as being capable or incapacitated. We do not implement practical possibility however.

### 4.1 Implementing Norms

The normative relations we implement are defined by predicates stored in the assertions $A$. Relations can apply to agents directly or via *roles*; an agent occupies one or more roles (also stored in the assertions $A$). There are four types of normative predicate: *power, permitted, obliged* and *sanction*. Sanctions are defined for actions which agents should not do. Permitted or obliged actions are treated as exemptions to these sanctions, i.e. the sanction applies unless the agent was permitted or obliged. Power and permission have arity 3: the first parameter is the agent name (or role name), the second is the performative of the speech act he is empowered/permitted to do, and the third is a condition. For example

```
power(bidder,bid,[Content=[C],C>47])
```

means that any agent in the role of bidder is empowered to send a *bid* speech act provided it complies with the following conditions: the content of the act must be a list containing a single number whose value is greater than 47. If the condition is the empty list then it is always true. Sanctions and obligations add a further (fourth) parameter, which is the "sanction code". Following [20] we will associate a 3-figure "sanction code" with each norm violation (similar to the error codes used in the Internet protocol HTTP), in our case higher numbers are used for more egregious violations. The sanction codes gathered by each agent as it commits offences are merely recorded in a list. The use of codes is just a convenient way to record sanctions without yet dealing with them; we would require a separate component to impose some form of punishment. Finally the obligation adds a fifth parameter which is the deadline by which the specified speech act must be sent.

The algorithm shown in Fig. 2 is added to the *speechAct* rule to handle the normative relations, it effectively defines an operational semantics for the normative relations. With this implementation we make obligation imply permission

**algorithm** HANDLE-NORMS

1. Input: a speech act with *Sender, Receiver, Performative, Content*
2. Check if there is an obligation which requires that *Sender* (or one of the roles he occupies) send this speech act. If so remove the obligation from $A$ and jump to 5.
3. Check if there is a sanction for *Sender* (or one of the roles he occupies) sending this speech act: If not, go to the next step; If so,
   - check if there is a permission for *Sender* (or one of the roles he occupies) to send this speech act: If so, go to the next step; If not, apply the specified sanction.
4. Check if *Sender* (or one of the roles he occupies) is empowered to send this speech act: If not, discard the act and exit this algorithm.
5. Process the act as normal.

Fig. 2: Algorithm to Handle Normative Relations

and power. It is in this algorithm that roles are consulted to retrieve the names of the agents occupying the roles; e.g. when checking if an agent who has just sent a message is obliged (and hence permitted), the algorithm will consult the facts to see what roles the sending agent occupies. We also need to add the following to the *housekeeping* rule (recall that the housekeeping rule is invoked on every timer event):

– For each obligation check if it has timed out. If so, apply the sanction to the agent (or all agents occupying the obliged role) and remove the obligation from $A$.

Note that we are assuming that the existence of a *speechAct* rule is an agreed standard across component developers, so that any new components can add checks and guards to this rule.

## 5 Protocol Components

Protocols are additional components of the ACL, they are each encoded via their own rules in $R$. Each protocol has a unique name and may be represented by a number of clauses in $R$. Protocols essentially determine what actions are to be taken next, given the current state and an event that happens. They do this by consulting the current state and modifying the normative relations according to the event that has just happened. Agents initiate protocols by using the special speech act *initProtocol*; the *speechAct* predicate passes control to the protocol on initiation. A protocol initiates a "sub-conversation" within the institution. All the assertions describing the protocol's state of execution are gathered together as an indexed list within $A$. In order to ensure the index is unique, the initiator will compose an index by concatenating his name with a positive integer which increases with each new protocol he initiates. Subsequently all speech acts indexed with the protocol's identifier will be processed by the protocol's rules (instead of the standard rules which process speech acts that are not part of any protocol). Normative relations defined within the protocol's "space" in $A$ only apply to messages that are part of that protocol. Timer events are processed by all protocols running at any time. Agents are free to enter

multiple parallel protocols, each being a separate sub-conversation. Sending a *exitProtocol* message terminates the protocol and removes its assertions from $A$.

### 5.1 Example Protocol: Auction

The Vickrey auction protocol below is expected to be invoked by a speech act with content [*Index,Protocol,Item,OffersOver,ClosingTime*]. These then become variables accessible to the initiator clause of the protocol rule, along with the initiator of the protocol and the list of receivers. Each clause has access to the variables *Sndr* and *Rcvr* from the event that invoked the clause (we cannot use the names *Sender* and *Receiver* as these are used by content checking conditions). The Prolog-style pseudocode below describes a series of clauses, one to handle each speech act that can happen during the execution of the protocol. To keep the presentation concise we have avoided presenting the example in real Prolog code.

```
initiator:
 add role(Sndr,auctioneer)
 for each Rcvr add role(Rcvr,bidder)
 add power(bidder,bid,[Content>OffersOver])
 add permitted(bidder,bid,[Receiver=R,
               role(R,auctioneer),Content>OffersOver])
 add sanction(bidder,bid,[],100)
 retrieve global.timer(Time)
 add timeout(closingTime+Time)
 add item(Item)
 add high1(0)
 add high2(0)

if bid([auctioneer,NewBid])
 retrieve high1(High1)
 retrieve high2(High2)
 if NewBid>High1 then replace winner(_) with winner(Sndr)
                      replace high1(_) with high1(NewBid)
 else if NewBid>High2 then replace high2(_) with high2(NewBid)

if timer(Time)
 retrieve timeout(T)
 if Time>T then
  retrieve high1(High1)
  retrieve high2(High2)
  NewTime = Time+50
  if High1=High2 then
   obliged(auctioneer,exitProtocol,[Receiver=bidder],101,250)
  else
   remove power(bidder,bid,_)
   add power(auctioneer,inform,[])
   retrieve winner(Winner)
   add obliged(auctioneer,inform,
              [Receiver=Winner,Content=[won,High2]],103,NewTime)
   add obliged(auctioneer,exitProtocol,
               [Receiver=[bidder]],101,NewTime)
```

```
if inform([won,Price]) then
 retrieve global.timer(Time)
 retrieve winner(Winner)
 retrieve item(Item)
 NewTime = Time + 150
 add global.obliged(auctioneer,inform,
    [Receiver=bank,Content=[transfer,Item,Winner]],102,NewTime)
 add global.obliged(Winner,inform,[Receiver=bank,
    Content=[credit,Price,auctioneer]],102,NewTime)
```

Note that the final clause creates obligations which are to persist after the protocol's termination (this is the meaning of `add global...`). When this is done the agent's name is put in the obligation instead of the role name. This is because the role will cease to exist on termination of the protocol (i.e. the fact asserting it is within the indexed list of facts for that protocol, and hence will be deleted when the protocol terminates), whereas we want the agent to still be obliged to pay even after the auction is finished.

### 5.2 Auction Animation

The initiating speech act is

```
speechAct(alice, [bob,claire], initProtocol,
            [alice1,auction,IPRowner,47,200])
```

Here *initProtocol* is the performative and *auction* is the protocol to be initiated. This starts a new conversation state, having its assertions as an indexed list within $A$. The index is *alice1*. Subsequent messages which are part of this protocol execution must be tagged with this index at the head of their content list. After this the following assertions hold within the indexed list

```
role(alice,auctioneer)
role(bob,bidder)
role(claire,bidder)
power(bidder,bid,[Content=[C],C>47])
power(auctioneer,exitProtocol,[Receiver=[bob,claire]])
permitted(bidder,bid,[Receiver=R,role(R,auctioneer),Content=[C],>47])
sanction(bidder,bid,[],100)
```

The next speech act is a bid by *bob*:

```
speechAct(bob, alice, bid, [alice1,53])
```

The only effect of this is to add a predicate recording this as the highest bid. Bidders still retain the power and permission to revise their bids. Next we have *claire* bidding 51, which adds a predicate recording the second highest bid. Then the timeout event happens. This results in the power to bid being revoked. Agents are still permitted to bid, but it has no effect. We now have the following norms for the auctioneer:

```
power(auctioneer,inform,[])
obliged(auctioneer,inform,[Receiver=bob,Content=[won,51]],103,250)
obliged(auctioneer,exitProtocol,[Receiver=[bob,claire]],101,250)
```

Note that the auctioneer is empowered to inform anything to the bidders; whatever he says, it counts. However, he is obliged to announce the winner and losers as expected in a Vickrey auction. The auctioneer's next messages are

```
speechAct(alice, bob, inform, [alice1,won,52])
speechAct(alice, [bob, claire], exitProtocol,[alice1])
```

This terminates the protocol and generates two obligations which exist in the "root" of $A$, i.e. not in the sublist indexed by *alice1*.

```
obliged(alice,inform,[Receiver=bank,
         Content=[transfer,IPRowner,bob]],102,400)
obliged(bob,inform,[Receiver=bank,Content=[credit,52,alice]],102,400)
```

Note that *alice* has overcharged *bob*. Without any third party monitoring, there is no way for him to know. However, we could imagine a subsequent dialog where *claire* reveals her bid to him and he lodges a complaint with an arbitration authority. If the evidence is deemed to be sufficient, the protocol specification can be consulted again to determine the appropriate sanction, i.e. that sanction 103 should be enforced on *alice*.

## 6 Temporal Logic Component

Our *obliged* predicate only allows us to specify that an action must be done before some future time. We have also added a temporal logic component which allows us to express more complex conditions. For example we can specify that an agent is obliged to ensure that a certain condition holds true, where the condition is expressed in temporal logic, but ultimately refers to the truth values of predicates in $A$. We are interested in making these kinds of normative specifications for the behaviour of agents, and then testing their compliance with the specification by observing their behaviour at runtime (by observing finite runs of the system). This is the same type of testing as described by Venkatraman and Singh [24]; i.e. given an observed run of the system we can determine if the agents have complied so far, but not if they will comply in other circumstances. We have found that the standard temporal logic operator $\Diamond$ (at some time in the future) is not very useful for our specifications. The linear temporal formula $\Diamond p$ promises that $p$ will eventually true, but there is no way to falsify it in a finite model; i.e. if we require that an agent perform some action eventually, it is not possible to be non compliant in a finite model. Hence this formula becomes meaningless when referring to agent behaviour in a finite observed sequence. A typical type of formula we need to specify is that some condition must hold continuously before a deadline. This can be done with the $\mathcal{U}$ (until) operator. The formula $p\mathcal{U}q$ means that $p$ must hold continuously until $q$ becomes true, and $q$ should eventually become true (this second part is of course redundant in our finite sequences). We include Boolean connectives $\neg, \wedge$ and $\vee$ in the language. The $\Box$ operator (now and at all future times) is not included in our language because $\Box p$ is the same as $p\mathcal{U}\mathit{false}$ in a finite sequence. Despite the fact that our temporal logic only has one temporal operator, it is still quite expressive, as nestings of $\mathcal{U}$ can be used, as well as the Boolean connectives.

Using this simple language we have constructed a model checking component which keeps track of the temporal formulae which an agent is obliged to keep

true; i.e. the checker "carries forward" the pending formulae and checks each new state as timer events are processed. This allows the formulae to be used in normative specifications, and sanctions to be triggered automatically when the formulae are falsified. We use the method of particle tableau from [18] to check the formulae. This allows an efficient incremental construction of the relevant portion of the tableaux. Unfortunately the algorithm cannot be properly described in the space available.

## 7 Related Work

There are few recent works which address standardising agent communication semantics. It appears that the effort has been abandoned since the attempts of FIPA and KQML [8, 15, 7] in the 90's. However, in terms of technical ideas, there are some recent proposals which are moving in directions similar to what we propose.

In [10] it is demonstrated that a system of production rules can be used to implement many agent institutions that had originally been specified with very diverse formalisms. This is similar to our proposal as it is given a common computational grounding to proposals which were previously hard to compare. It also shows that if we are considering computational implementations of agent communication, then one simple language will be sufficient to implement whatever diverse notions we choose to employ to govern the agents.

In [20] the possibility of agents modifying the rules of the institution is mentioned; it is stated that this would require "interpretable code or some form of dynamic compilation". In [2] the event calculus formalism has been implemented to animate a specification of a rule governed agent society, but it is also stated that features of the underlying programming language could be made accessible to complement the event calculus formalism; this comes closer to the flavour of our proposal. In [11] normative relations are implemented in the Jess production rule system. The authors mention the possibility of "societal change", where societies may "evolve over time by altering, eliminating or incorporating rules". This societal change facility is not actually implemented in [11], but the authors do specify norms in a computationally grounded language based on observable phenomena.

In [6] there is a proposed development methodology which is similar to the "component based" aspect of our approach; generic protocols are specified, and then *transformers* can be applied to them to capture variations of the protocol for specific contexts. The work of [26] advocates the need for tools to assist developers in protocol design, while also showing how protocols can be built on the social commitments approach to agent communication semantics; this type of tool support and structured development is exactly what we expect will be needed to take our standardisation approach forward.

## 8 Conclusions and Future Work

There are two requirements which should be fulfilled as a precondition to making a standard for agent communication which has a reasonable prospect of actually being adopted. One is the expressive power to allow developers to do what they want, and the second is the ease of use (for which tools are required). The first

aspect is easy, as we have shown, the second will take more effort. Even with the few components we described, we can see already that programming moves to a higher level as we add more components. We expect that standardisation will need to proceed by means of evolving libraries and tools which make the agent developers job easier. In this process the role of a standards body would be to accredit components and publish them, and to standardise the form of their documentation.

We are currently experimenting with our temporal logic component which model checks temporal logic formulae, and plan to extend the expressiveness of its language. Argumentation is yet more interesting as it typically requires the use of nonmonotonic logics: an agent may undercut another agent's argument, and so force a conclusion to be retracted. Here we would code the rules defining acceptability of an argument. The ability of a meta-interpreter to specify a depth limit on proofs is particularly useful for this purpose; in order to have a common consensus on what arguments are accepted we need to specify the limits on the resource bounded reasoning [17]. Argumentation also introduces the possibility of negotiating changes to the rules of the institution itself. There will also no doubt be considerable interest in developing components for various logics such as the C+ action language (which is gaining popularity [2, 6]) and various modal logics.

Eventually it is hoped that different electronic institutions could publish the components which comprise their communication language in a machine readable format, so that a roaming agent could come and join the institution without needing to be programmed to use that particular language in advance. This is an ambitious goal, as the agent would need not to just know the rules, but also its strategy for participation. However, if we restrict our attention to certain types of dialog, and their variants (e.g. auctions) then it does seem feasible.

## References


1. R. Agerri and E. Alonso. Semantics and Pragmatics for Agent Communication. volume 3808 of *LNAI*. Springer-Verlag, 2005.
2. A. Artikis, M. Sergot, and J. V. Pitt. Specifying Norm-Governed Computational Societies. Technical Report 06-5, Dept. of Computing, Imperial College, London, UK, 2005.
3. J. L. Austin. *How To Do Things With Words*. Oxford University Press, 1962.
4. J. Bentahar, B. Moulin, J.-J. C. Meyer, and B. Chaib-Draa. A Computational Model for Conversation Policies for Agent Communication. volume 3487 of *LNAI*. Springer-Verlag, 2005.
5. B. Chaib-Draa, M.-A. Labrie, M. Bergeron, and P. Pasquier. An Agent Communication Language Based on Dialogue Games and Sustained by Social Commitments. *Autonomous Agents and Multi-Agent Systems*, 13(1):61–95, 2006.
6. A. K. Chopra and M. P. Singh. Contextualizing Commitment Protocols. In *Procs. 5th. Int'l Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Hakodate, Japan, May 2006. ACM Press.
7. P. R. Cohen and H. J. Levesque. Communicative Actions for Artificial Agents. In *Int'l Conf. on MASs*, pages 65–72, Mass., USA, 1995. MIT Press.
8. FIPA. [FIPA OC00003] FIPA 97 Part 2 Version 2.0: Agent Communication Language Specification. In *Website of the Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa2000.tar.gz*, 1997.

9. N. Fornara, F. Vigano, and M. Colombetti. Agent communication and artificial institutions. *Autonomous Agents and Multi-Agent Systems*, online; DOI: 10.1007/s10458-006-0017-8, 2006.
10. A. Garcia-Camino, J. Rodriguez-Aguilar, C. Sierra, and W. Vasconcelos. A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions. *SIGEcomm Exchanges*, 5(5), 2006.
11. A. Garcia-Camino, J.-A. Rodriguez-Aguilar, and P. Noriega. Implementing Norms in Electronic Institutions. In *Procs. 4th Int'l Conf. on Autonomous Agents & Multiagent Systems (AAMAS)*, Utrecht, The Netherlands, 2005. ACM Press.
12. F. Guerin and J. V. Pitt. A semantic framework for specifying agent communication languages. In *Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 395–396. IEEE Computer Society, Los Alamitos, California, 2000.
13. A. J. I. Jones and M. J. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.
14. Y. Labrou. *Semantics for an agent communication language*. PhD thesis, Baltimore, MD: University of Maryland Graduate School, 1996.
15. Y. Labrou and T. Finin. A semantics approach for kqml – a general purpose communication language for software agents. In *Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 447–455, 1994.
16. Y. Labrou, T. Finin, and Y. Peng. The current landscape of agent communication languages, 1999.
17. R. P. Loui. Process and policy: Resource-bounded nondemonstrative reasoning. *Computational Intelligence*, 14(1):1, 1998.
18. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems (Safety), vol. 2*. Springer-Verlag, New York, 1995.
19. J. Mayfield, Y. Labrou, and T. Finin. Desiderata for agent communication languages. In *Proceedings of the AAAI Symposium on Information Gathering from Heterogeneous, Distributed Environments, AAAI-95 Spring Symposium*, pages 347–360. Stanford University, Stanford, CA, 1995.
20. J. Pitt, L. Kamara, M. Sergot, and A. Artikis. Formalization of a voting protocol for virtual organizations. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'05), Utrecht, July 2005*. ACM Press, 2005.
21. J. R. Searle. What is a speech act ? In *Philosophy of Language. edited by A.P. Martinich, Third edition*. 1996, Oxford University Press, 1965.
22. M. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.
23. M. Singh. A social semantics for agent communication languages. In *IJCAI Workshop on Agent Communication Languages, Springer-Verlag, Berlin.*, 2000.
24. M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 2(3):217–236, 1999.
25. M. Verdicchio and M. Colombetti. A logical model of social commitment for agent communication. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems table of contents Melbourne, Australia*, pages 528 – 535, 2003.
26. P. Yolum. Towards design tools for protocol development. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 99–105, New York, NY, USA, 2005. ACM Press.

# Satisfying Maintenance Goals

Koen V. Hindriks[1] and M. Birna van Riemsdijk[2]

[1] EEMCS, Delft University of Technology, Delft, The Netherlands
[2] LMU, Munich, Germany

**Abstract.** A rational agent derives its choice of action from its beliefs and goals. Goals can be distinguished into achievement goals and maintenance goals. The aim of this paper is to define a mechanism which ensures the satisfaction of maintenance goals. We argue that such a mechanism requires the agent to look ahead, in order to make sure that the execution of actions does not lead to a violation of a maintenance goal. That is, maintenance goals may constrain the agent in choosing its actions. We propose a formal semantics of maintenance goals based on the notion of lookahead, and analyze the semantics by proving some properties. Additionally, we discuss the issue of achievement goal revision, in case the maintenance goals are so restrictive that all courses of action for satisfying achievement goals will lead to a violation of maintenance goals.

## 1 Introduction

The research presented in this paper concerns the role of *maintenance goals* in the selection of actions by a rational agent. A rational agent aims at satisfying its goals, which may include both achievement goals as well as maintenance goals. Achievement goals define states that are to be achieved, whereas maintenance goals define states that must remain true.

The distinction between achievement and maintenance goals is common in the literature about rational agents. However, whereas various proposals for computational semantics and programming frameworks that include achievement goals are available [3, 5, 11, 14, 17, 18], maintenance goals have received less attention [3, 4, 6]. In this paper we investigate a semantics for maintenance goals. Our aim is to define a mechanism which ensures the satisfaction of maintenance goals that can be integrated into various agent programming languages.

Achievement goals in agent programming frameworks are typically used in combination with rules that express which action or plan an agent may execute in certain circumstances in order to achieve a particular achievement goal. In such a setting, achievement goals thus trigger the execution of a course of action. A maintenance goal can have a similar role in agent programming frameworks, in the sense that it can trigger an agent to perform actions in order to ensure that a maintenance goal is not violated, or to take action to reestablish the maintenance goal if it is violated.

Implementing maintenance goals using conditions to trigger the execution of actions, however, is not sufficient to guarantee that maintenance goals are

not violated. In order to prevent the violation of a maintenance goal, an agent may sometimes have to *refrain* from performing an action that the agent would otherwise have selected, e.g., to satisfy one of its achievement goals [6]. A comprehensive framework for maintenance goals should thus not only incorporate an action selection mechanism based on triggering conditions, but should also take into account the *constraining* role of maintenance goals. As we will show, a selection mechanism that is based on this constraining role can also be used to actively ensure that a maintenance goal is not violated.

We argue that taking into account the constraining role of maintenance goals requires some kind of *lookahead* mechanism, which allows the agent to determine whether certain actions or plans it would like to perform might lead to the violation of one of its maintenance goals. The main aim of this paper is to investigate how the semantics of maintenance goals can be formally defined through such a lookahead mechanism. We analyze the semantics formally by proving several properties. It is important to note that one advantage of giving a formal semantics, is the fact that one can formally prove certain properties. This is important in order to get a clear understanding of the phenomenon under investigation. Besides providing a formal semantics and analysis of maintenance goals, we discuss the issue of achievement goal revision, in case the maintenance goals are so restrictive that all plans for satisfying achievement goals will lead to a violation of maintenance goals.

The paper is organized as follows. In Section 2, a motivating example is introduced to illustrate the main ideas throughout the paper. Our investigations are carried out in the context of the agent programming language GOAL [5], which is briefly introduced in Section 3. The results presented, however, are general and can be integrated into any agent framework. Section 4 formally defines a lookahead mechanism that ensures selected actions do not violate the agent's maintenance goals. The look-ahead mechanism introduced, however, may overconstrain an agent's decision procedure. In Section 5 this problem is discussed and a revision procedure is suggested to resolve it. Section 6 concludes the paper, outlines some directions for future work, and discusses related work.

## 2 Motivating Example: A Carrier Agent

In this section, a simple scenario involving a carrier agent is presented in order to illustrate the role of maintenance goals in action selection and the reasoning we believe is involved in avoiding violation of maintenance goals.

### 2.1 The Basic Scenario

The setting is as follows. Consider an agent who wants to bring parcels from some location A to a location B, using its truck. The distance between A and B is too large to make it without refueling, and so, in order not to end up without gas, the agent needs to stop every once in a while to refuel. The fact that the agent does not want to end up without gas, can be modeled as a maintenance

goal.[3] This maintenance goal *constrains* the actions of the agent, as it is not supposed to drive on in order to fulfil its goal of delivering the parcels, if driving on would cause it to run out of gas.

The action of driving is an action that the agent can take in order to fulfil its achievement goal of delivering the parcels. Other actions that the agent has at its disposal, may be used to *actively* ensure that the agent's maintenance goals are not violated. In the example scenario, the action of refueling can be viewed as an action of this kind (although, in this example, not violating the maintenance goal is also instrumental for achieving the achievement goal). Maintenance goals thus on the one hand constrain the agent's actions, but may also induce the agent to take preventive actions to make sure maintenance goals are not violated.

An essential reasoning mechanism in order to ensure that the agent does not take actions that would violate the agent's maintenance goals is a *lookahead* mechanism. In the example scenario, the agent should reason about the distance to the next gas station and the amount of fuel it has left, in order to make sure it does not end up without fuel between two gas stations. That is, it should in one way or another, reason about the consequences of possible future sequences of actions in order to be able to choose those actions that will not lead to a violation of maintenance goals at some point in the future.

### 2.2 Conflicts between Achievement and Maintenance Goals

In this simple scenario so far, there is no conflict between the agent's maintenance goals and achievement goals. It is perfectly possible for the agent to deliver its parcels without running out of gas, as long as it refuels in time. It may, however, sometimes be the case that conflicts between achievement goals and maintenance goals arise, in the sense that in order to achieve an achievement goal, the agent will have to violate a maintenance goal.

In the example scenario, such a conflict may arise if the agent has the additional maintenance goal of making sure that the weight of truck load stays below a certain threshold. Assuming that the total weight of the parcels exceeds this threshold, and assuming that the agent cannot drive back and forth between A and B (e.g., because the agent has loaned the truck and has to return it after arriving at location B), there is a conflict between the achievement goal of bringing all the parcels from A to B, and not overloading the truck.

Such a situation of conflict may result in the agent not doing anything anymore at a certain point. That is, it may be the case that any action the agent is able to do to achieve its achievement goal is not allowed because this would lead to a violation of the agent's maintenance goal, and, moreover, there is no possibility to actively ensure that the maintenance goal is not violated. In general, there are several possibilities of dealing with such a situation.

The first option is not to do anything about it. The intuition here is that the agent should never violate its maintenance goals, i.e., maintenance goals are *hard constraints*, and the agent wants "all or nothing" when it comes to

[3] Other papers [3, 4, 6] have used a similar maintenance goal in some of their examples.

its achievement goals. In the example scenario, it may be the case that it is of utmost importance that the truck is not overloaded, e.g., because the truck has a device with which the weight of the freight is measured, and if the weight exceeds the threshold the truck cannot start. Moreover, it may be the case that bringing the parcels only makes sense if all parcels are brought, e.g., because the parcels contain parts of a closet and there is no use for bringing only part of the closet. Put differently, the utility of delivering only part of the parcels is zero.

A second option is to allow the agent to *violate its maintenance goals*, if this is absolutely necessary in order to achieve an achievement goal. An intuitive implementation of such a mechanism would have to make sure that the agent really only violates maintenance goals if there is no way around it, and if this is necessary, it should try to "minimize" the violation, e.g., by trying to make sure that the maintenance goal is satisfied again as soon as possible after the achievement goal that was the reason to violate the maintenance goal has been satisified. In the example scenario, it may be the case that overloading the truck does not do too much harm, as long as this does not happen too often. It is then important that the truck is unloaded as soon as the destination is reached.

The third option is to *modify the achievement goal*, such that the modified achievement goal does not conflict anymore with the agent's maintenance goals. The idea here is that there might be achievement goals that can be achieved "to a certain degree", i.e., it might be possible to "weaken" the achievement goal, in case it would conflict with a maintenance goal. In the example scenario, the conflict between the achievement goal of getting all parcels at location A, and the maintenance goal of not overloading the truck, could be resolved by modifying the achievement goal such that the agent settles on bringing only *part* of the parcels to location B. The decision of which parcels to leave behind can be based on the weight of the parcels, i.e., the weight of the parcels to be taken along should not exceed the threshold, and on the utility of getting certain parcels at the destination, i.e., some parcels may be more important than others.

Of course, combinations of these possibilities of dealing with conflicts are also possible. Such combinations might define certain maintenance goals as hard constraints and certain achievement goals as "all or nothing" goals, while other maintenance goals and achievement goals may be violated or modified, respectively. In this paper, however, we focus on the third option, i.e., we view maintenance goals as hard constraints, and opt for the modification or weakening of achievement goals in case a conflict with a maintenance goal arises. In domains in which maintenance goals relate, e.g., to the limited availability of resources and time which cannot easily be lifted the third strategy will typically be valid.

## 3 The GOAL Language

In this section, the GOAL programming language [5, 10] is briefly introduced and a GOAL agent that implements a simplified version of the carrier agent of Section 2.1 is presented. A GOAL agent selects actions on the basis of its beliefs and achievement goals, i.e., maintenance goals were not investigated in

the original GOAL language. Whenever goals are mentioned in this section, this should thus be interpreted as meaning achievement goals. The definitions we provide in this section are used to make the notion of an agent computation precise, which we use in Section 4 to define the semantics for maintenance goals.

A GOAL program for the carrier agent is specified in Table 1. The program consists of four sections: (1) a set of initial beliefs, collectively called the (initial) *belief base* of the agent, (2) a set of initial achievement goals, called the (initial) *goal base*, (3) a *program section* which consists of a set of conditional actions, and (4) an *action specification* that consists of a specification of the pre- and post-conditions of *basic actions* of the agent. In the example, variables are used as a means for abbreviation; variables should be thought of as being instantiated with the relevant arguments to yield propositions. The constants used in the example denote locations (`a`, `ab1`, `ab2`, `b`, assumed to be spatially positioned in this order), parcels (`p1`,`p2`) and a truck `truck`. The order of the locations means that if the agent wants to get from `a` to `b`, it first has to pass `ab1`, and then `ab2`. We use the comma to denote conjunction.

```
:beliefs{ loc(p1,a). loc(p2,a). loc(truck,a). loc(gasstation,ab1).
          fuel(2). next(a,ab1). next(ab1,ab2). next(ab2,b). }
:a-goals{ loc(p1,b), loc(p2,b). }
:program{
    if B(loc(truck,X), loc(P,X), X≠Y), G(loc(P,Y)) then load(P).
    if B(loc(truck,a)), ∼(B(loc(P,a)), G(loc(P,b))), G(loc(R,b))
       then adopt(loc(truck,b)).
    if G(loc(truck,b)) then move.
    if B(loc(gasstation,X)) then tank.
    if B(loc(truck,X), in(P,truck)), G(loc(P,X)) then unload(P). }
:action-spec{
    move     { :pre{loc(truck,X), next(X,Y), fuel(Z), Z > 0}
               :post{loc(truck,Y), not loc(truck,X), fuel(Z-1), not fuel(Z)} }
    load(P)  { :pre{loc(P,X), loc(truck,X)}     :post{in(P,truck), not loc(P,X)} }
    unload(P){ :pre{in(P,truck), loc(truck,X)}  :post{loc(P,X), not in(P,truck)} }
    tank     { :pre{loc(truck,X), loc(gasstation,X), fuel(Y), Y<3}
               :post{fuel(3), not fuel(Y)}} }
```

**Table 1.** GOAL Carrier Agent

The belief base, typically denoted by $\Sigma$, and the goal base, typically denoted by $A$, together define the *mental state* of a GOAL agent. Mental states should satisfy a number of rationality constraints, which are introduced next.

**Definition 1** *(Mental States)*
Assume a language of propositional logic $\mathcal{L}_0$ with the standard entailment relation $\models$ and typical element $\phi$. A mental state of a GOAL agent, typically denoted by $s$, is a pair $\langle \Sigma, A \rangle$ with $\Sigma, A \subseteq \mathcal{L}_0$ where $\Sigma$ is the belief base, and $A$ with typical element $\alpha$ is the goal base. Additionally, mental states need to satisfy the following *rationality constraints*:

| | | |
|---|---|---|
| (i) | The belief base is consistent: | $\Sigma \not\models \bot$, |
| (ii) | Individual goals are consistent[4]: | for all $\alpha \in A$: $\not\models \neg\alpha$, |
| (iii) | Goals are not believed to be achieved: | for all $\alpha \in A$: $\Sigma \not\models \alpha$. |

In the example carrier agent, the two parcels and the truck are initially believed to be at location `a`, represented by `loc(p1,a)`, `loc(p2,a)`, and `loc(truck,a)`. The agent also believes it has two units of fuel, and that the gas station is at location `ab1`. The initial achievement goal of the agent is to have both parcels at location `b`, represented by `loc(p1,b), loc(p2,b).` Note that the carrier agent satisfies the rationality constraints on mental states.

A GOAL agent derives its choice of action from its beliefs and goals. In order to do so, a GOAL agent inspects its mental state by evaluating so-called *mental state conditions*. The syntax and semantics of these conditions is defined next.

**Definition 2** *(Mental State Conditions)*
The language $\mathcal{L}_M$ of mental state conditions, typically denoted by $\psi$, is inductively defined by the two clauses:

- if $\phi \in \mathcal{L}_0$, then $\mathbf{B}\phi, \mathbf{G}\phi \in \mathcal{L}_M$,
- if $\psi_1, \psi_2 \in \mathcal{L}_M$, then $\neg\psi_1, \psi_1 \wedge \psi_2 \in \mathcal{L}_M$.

The truth conditions of mental state conditions $\psi$, relative to a mental state $s = \langle \Sigma, A \rangle$, are defined by the following four clauses:

| | | |
|---|---|---|
| $s \models_m \mathbf{B}\phi$ | iff | $\Sigma \models \phi$, |
| $s \models_m \mathbf{G}\phi$ | iff | there is $\alpha \in A$ such that $\alpha \models \phi$ and $\Sigma \not\models \phi$, |
| $s \models_m \neg\psi$ | iff | $s \not\models_m \psi$, |
| $s \models_m \psi_1 \wedge \psi_2$ | iff | $s \models_m \psi_1$ and $s \models_m \psi_2$. |

The semantics of $\mathbf{B}\phi$ defines that this holds iff $\phi$ follows from the belief base under a standard proposition logic entailment relation. The definition of the semantics of $\mathbf{G}\phi$ is somewhat more involved. It specifies that $\mathbf{G}\phi$ holds, iff $\phi$ is not already believed by the agent, and there is a formula in the goal base from which $\phi$ follows. Also multiple goals are not required to be consistent which reflects the fact that each goal may be realized at a different moment in time.

In GOAL, two types of actions are distinguished: basic actions and goal update actions. The execution of basic actions updates and modifies the agent's beliefs, apart from changing the agent's environment. Indirectly, a basic action may also affect the goal base of an agent. That is, in case a goal is believed to be achieved after action execution the goal is dropped by the agent and may be removed from the agent's goal base.

In the example program, the way in which the execution of basic actions changes the beliefs of the agent is specified using pre- and post-conditions. The example agent has four basic actions at its disposal, i.e., the actions `move`, `load(P)`, `unload(P)`, and `tank`. Through the action `move`, it can move one position towards location `b`. Using `unload(P)` and `load(P)`, it can unload and load the parcel `P`, respectively, if the agent is at the same location as the parcel. The action `tank` can be executed if the agent is at location `ab1`, resulting in the amount of fuel becoming `3`.

In the formal definition of GOAL, we use a *transition function* $\mathcal{T}$ to model the effects of basic actions. This function maps a basic action `a` and a belief

base $\Sigma$ to an updated belief base $\mathcal{T}(\mathtt{a}, \Sigma) = \Sigma'$. The transition function is undefined if an action is not enabled in a mental state. In a GOAL agent, the action specification section of that agent specifies this transition function. In the example agent in Table 1 a STRIPS-like notation is used, where positive literals define the add list and negative literals define the delete list (cf. [12]). (Other, extended action formalisms could be used but for the purpose of this paper a more extended formalism is not needed.) GOAL has two built-in goal update actions: the **adopt**($\phi$) action to adopt a goal, and the **drop**($\phi$) to drop goals from the agent's goal base. An **adopt**($\phi$) action has to satisfy the rationality constraints on mental states, i.e. $\phi$ must be consistent and not believed by the agent. The **drop**($\phi$) action removes all goals from the goal base that imply $\phi$.

**Definition 3** *(Mental State Transformer $\mathcal{M}$)*
Let $\mathtt{a}$ be a basic action, $\phi \in \mathcal{L}_0$ and $\mathcal{T}$ be a transition function for basic actions. Then the *mental state transformer function* $\mathcal{M}$ is defined as a mapping from actions and mental states to updated mental states as follows:

$$\begin{aligned}
\mathcal{M}(\mathtt{a}, \langle \Sigma, A \rangle) &= \begin{cases} \langle \Sigma', A \setminus \{\psi \mid \Sigma' \models \psi\} \rangle & \text{if } \mathcal{T}(\mathtt{a}, \Sigma) = \Sigma' \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathcal{M}(\mathbf{adopt}(\phi), \langle \Sigma, A \rangle) &= \begin{cases} \langle \Sigma, A \cup \{\phi\} \rangle & \text{if } \not\models \neg\phi \text{ and } \Sigma \not\models \phi \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathcal{M}(\mathbf{drop}(\phi), \langle \Sigma, A \rangle) &= \langle \Sigma, A \setminus \{\psi \in A \mid \psi \models \phi\} \rangle
\end{aligned}$$

In order to select the appropriate actions to achieve the goal of having the two parcels at location $\mathtt{b}$, our example carrier agent has five conditional actions as listed in the program section of Table 1. A conditional action $c$ has the form **if** $\psi$ **then** $\mathtt{a}$, with $\mathtt{a}$ either a basic action or a goal update action. This conditional action specifies that $\mathtt{a}$ may be performed if the mental state condition $\psi$ and the preconditions of $\mathtt{a}$ hold. In that case we say that conditional action $c$ is *enabled*.

During execution, a GOAL agent selects non-deterministically any of its enabled conditional actions. This is expressed in the following transition rule, describing how an agent gets from one mental state to another.

**Definition 4** *(Conditional Action Semantics)*
Let $s$ be a mental state, and $c =$ **if** $\psi$ **then** $\mathtt{a}$ be a conditional action. The transition relation $\xrightarrow{c}$ is the smallest relation induced by the following transition rule.

$$\frac{s \models \psi \quad \mathcal{M}(\mathtt{a}, s) \text{ is defined}}{s \xrightarrow{c} \mathcal{M}(\mathtt{a}, s)}$$

The execution of a GOAL agent results in a *computation*. We define a computation as a sequence of mental states, such that each mental state can be obtained from the previous by applying the transition rule of Definition 4. As GOAL agents are non-deterministic, the semantics of a GOAL agent is defined as the *set* of possible computations of the GOAL agent, where all computations start in the initial mental state of the agent.

**Definition 5** *(Agent Computation)*
A computation, typically denoted by $t$, is an infinite sequence of mental states $s_0, s_1, s_2, \ldots$ such that for each $i$ there is an action $c_i$ and $s_i \xrightarrow{c_i} s_{i+1}$ can be derived using the transition rule of Definition 4, or $s_i \not\xrightarrow{c_i}$ and for all $j > i$, $s_j = s_i$. The meaning $S_{\mathcal{A}}$ of a GOAL agent named $\mathcal{A}$ with initial mental state $\langle \Sigma_0, A_0 \rangle$ is the set of all computations starting in that state.

Observe that a computation is infinite by definition, even if the agent is not able to perform any action anymore from some point in time on. Also note that the concept of an agent computation is a general notion in program semantics that is not particular to GOAL. The notion of a computation can be defined for any agent programming language that is provided with a well-defined operational semantics. For such languages, it is possible to transfer the analysis of maintenance goals in this paper that is based on the notion of a computation and to incorporate the proposed maintenance goal semantics.

Our example carrier agent may execute the following computations. In the initial mental state, the conditional action for loading a parcel is executed, and the agent non-deterministically picks up one of the parcels, followed by another execution of this conditional action to load the other parcel. Consecutively, the only enabled conditional action is the one for adopting the goal `loc(truck,b)`, by which the example agent adopts the goal to be at location `b`. As the agent now has the goal to be at location `b` it will execute the enabled action `move`. After executing the `move` action, the agent is at location `ab1`, and has one unit of fuel left.

In this situation, there are two possibilities. The agent can execute another `move` action, after which the agent will be at location `ab2` without any fuel. The other option is that the agent executes the `tank` action, after which the agent will have three units of fuel while still being at location `ab1`. If the agent chooses the first option, it will get stuck at `ab2`, as it has no fuel and there is no possibility to tank. If the agent chooses the second option, it can execute two `move` actions after tanking and get to location `b`. Then the only option is to execute the conditional action for unloading parcels two times, after which the achievement goal of having the parcels at location `b` is reached.

## 4 Semantics of Maintenance Goals

In this section, we define the semantics of a GOAL agent if this agent is given a set of maintenance goals to satisfy. In defining the operational semantics for maintenance goals, the idea is that agents reason about the result of the execution of their actions, in order to make sure that only those actions are chosen that do not violate the agent's maintenance goals. That is, agents *look ahead* in order to foresee the consequences of their actions. Adding maintenance goals that may have a constraining role makes sense only if the original agent is underspecified, that is, if alternative courses of action are available, as in the case of GOAL agents. Only then can the agent actually *choose* to take actions that do

not violate maintenance goals. Intuitively, the idea is thus that the incorporation of maintenance goals leads to the exclusion of (parts of) computations that were allowed in the agent semantics of Definition 5 without maintenance goals.

In the example program of Section 3, we have seen that the carrier agent gets stuck at location `ab2` if it does not tank at location `ab1`. The idea is that such behavior can be prevented by introducing a maintenance goal that expresses that the agent should not be in a situation where it has no fuel left (Table 2).

```
:m-goals{ fuel(X), X > 0. }
```

**Table 2.** Extension With Maintenance Goals

Syntactically, the introduction of maintenance goals thus poses no problems. Incorporating maintenance goals in the semantics, however, is more involved and is the subject of the remainder of this section. In Section 5 we look at the case that maintenance and achievement goals cannot be satisfied simultaneously.

### 4.1 Operational Semantics of Maintenance Goals

Ideally, an agent should look ahead infinitely far into the future, in order to be absolutely sure that it does not choose a path that will lead to the violation of a maintenance goal. In practice, however, infinite lookahead cannot be implemented, and presumably it will neither be necessary. We propose a general definition of lookahead, that takes the *number of steps* that an agent may look ahead as a parameter. This parameter is called the *lookahead range*.

In the following, $\Omega \subseteq \mathcal{L}_0$ will denote a set of maintenance goals. A set of maintenance goals will be assumed to be consistent, i.e., $\Omega \not\models \bot$. If maintenance goals are hard constraints, it is not rational to have two maintenance goals that are inconsistent, as it will never be possible to satisfy both maintenance goals. Moreover, we assume that maintenance goals are satisfied initially, i.e., it should be the case that for the initial belief base $\Sigma_0$ we have $\Sigma_0 \models \Omega$ (where $\Sigma_0 \models \Omega$ abbreviates $\forall \omega \in \Omega : \Sigma \models \omega$). Also, we take the set of maintenance goals as being static. That is, an agent cannot drop or adopt new maintenance goals. Although there might be situations where one would want to consider dropping or adopting maintenance goals, we think that maintenance goals are intuitively more stable than achievement goals as the former express a kind of background requirements that an agent should always fulfill.

In order to provide a formal definition of the effect of $n$-step lookahead on the computations of an agent, we first introduce some additional terminology and notation. A *prefix* of a computation $t$ is an initial finite sequence of $t$ or $t$ itself. A prefix of length $n$ of a computation $t$ is denoted by $t^{\langle n \rangle}$ with $n \in \mathbb{N} \cup \{\infty\}$, where $t^{\langle \infty \rangle}$ is defined as $t$. $\mathbb{N}$ is the set of natural numbers including 0, and $\infty$ is the first infinite ordinal. We write $p \preccurlyeq p'$ to denote that $p$ is a prefix of $p'$.

The order $\preccurlyeq$ is lifted to sets as follows: $S \preccurlyeq S'$ iff each $p \in S$ is a prefix of some $p' \in S'$. A set $S$ of sequences is called a *chain* if for all $p, p' \in S$ we have either $p \preccurlyeq p'$ or $p' \preccurlyeq p$. The *least upper bound* of a chain $S$ is denoted by $\sqcup S$. In case of a set $S$ of prefixes of a computation $t$, $\sqcup S$ is either a maximal element in $S$ (i.e. a prefix that has the greatest finite length), or the computation $t$ itself (which need not be in $S$); moreover, $\sqcup\emptyset = \epsilon$ with $\epsilon$ the empty sequence. Finally, $s \in p$ for $s$ a mental state and $p$ a prefix of a computation abbreviates that $s$ is a state on the prefix $p$; sometimes $s_i$ is used to denote the $i^{th}$ state in the sequence.

Now we are in a position to formally define how maintenance goals, given an $n$-step lookahead operator $\restriction_n$, restrict the possible computations of an agent $\mathcal{A}$. First, we define the notion of a *safe prefix* of a computation $t$, given a set of maintenance goals $\Omega$ and the capability to do a lookahead of $n$ steps. The predicate $safe_n(p, \Omega)$, with $n \in \mathbb{N} \cup \{\infty\}$, is true if all states of the prefix $p$ of computation $t$ satisfy the maintenance goals $\Omega$ and, in the next $n$ steps of computation $t$ no violation of such a goal will occur, except possibly for the last state. (Note that we leave the computation $t$ implicit in $safe_n(p, \Omega)$.) This corresponds with the behavior of a very cautious agent that will avoid to go in a direction that may lead towards a violation of a maintenance goal. Formally, we define $safe_n(\epsilon, \Omega)$ to be false for technical reasons, and we define $safe_n(t^{\langle k \rangle}, \Omega)$ for prefixes of non-zero length $k > 0$ as follows:

$$safe_n(t^{\langle k \rangle}, \Omega) \text{ iff } \forall s \in t^{\langle k+n-1 \rangle}(\Sigma_s \models \Omega).$$

When the set of maintenance goals $\Omega$ is clear from the context, we also simply write $safe_n(t^{\langle k \rangle})$. All states on a safe prefix of a computation $t$ based on $n$-step lookahead have the property that lookahead does not predict any violations of a maintenance goal in $\Omega$ in less then $n$ steps. Note that there is at least one non-empty safe prefix including the initial state using 0-step lookahead since a goal agent initially must believe that its maintenance goals are satisfied. The set of all safe prefixes of computation $t$ is denoted by $Safe_n(t, \Omega)$. Note that the set $Safe_n(t, \Omega)$ is a chain and has a least upper bound, which is the computation $t$ itself when all prefixes of $t$ are safe.

The $n$-step lookahead operator $\restriction_n$ applied to a computation $t$ and a set of maintenance goals $\Omega$ can now be defined in terms of safe prefixes. Using this operator it is easy to define the effect of maintenance goals as hard constraints on the behavior of an agent with an $n$-step lookahead capability: The semantics $S_{\mathcal{A}}$ of an agent without such goals, i.e. its associated set of computations, is restricted by applying the lookahead operator to each computation in $S_{\mathcal{A}}$ to ensure that an agent with such lookahead capabilities will act cautiously and will never head towards a predicted violation of one of its maintenance goals.

**Definition 6** (Lookahead Operator and Semantics of Maintenance Goals)
The $n$-step lookahead operator $\restriction_n$, applied to a computation $t$ and a set of maintenance goals $\Omega$, is defined as the least upper bound of the set of safe prefixes of $t$ with respect to $\Omega$, and is also lifted to sets of computations.

- The $n$-step lookahead operator $\restriction_n$ is defined as: $t\restriction_n\Omega = \sqcup Safe_n(t, \Omega)$.

– The lift of $\restriction_n$ to a set $S$ is defined by:

$$S\restriction_n \Omega = \bigcup_{t \in S} \{t\restriction_n \Omega \mid \forall t' \in S : t\restriction_n \Omega \preccurlyeq t'\restriction_n \Omega \Rightarrow t\restriction_n \Omega = t'\restriction_n \Omega\}$$

– Let $\mathcal{A}$ be an agent with an $n$-step lookahead capability. Then the semantics of $\mathcal{A}$ with a set of maintenance goals $\Omega$ is defined as: $S_{\mathcal{A}}\restriction_n \Omega$.

The lift of $\restriction_n$ to a set $S$ is the set of all maximal elements of the set $\bigcup_{t \in S} t\restriction_n \Omega$. Only the maximal elements are taken in order to exclude prefixes $p$ that are a strict prefix of another prefix $p'$ in this set, i.e., $p \prec p'$. The semantics $S_{\mathcal{A}}\restriction_n \Omega$ for an agent $\mathcal{A}$ with maintenance goals $\Omega$ thus specifies that the agent continues until all further action would lead to a violation within $n$ steps. Note that the set $S_{\mathcal{A}}\restriction_n \Omega$ may be empty when the set of maintenance goals $\Omega$ is so restrictive that each computation would violate a maintenance goal within $n$ steps.

### 4.2 Properties

The following proposition says that a lookahead capability with a bigger lookahead range than another one is more restrictive than the latter. Since the semantics implements a cautious strategy towards possible violations of maintenance goals, an agent that detects such potential violations sooner, will act cautiously and will not follow a course of action that may lead to this violation.

**Proposition 1.** *If $n > m$, then $S_{\mathcal{A}}\restriction_n \Omega \preccurlyeq S_{\mathcal{A}}\restriction_m \Omega$.*

The proposition suggests that agents with a more powerful lookahead capability, i.e. with a greater lookahead range, possibly are able to satisfy fewer achievement goals than they would be able to satisfy with a less powerful lookahead capability. That is, an agent that does everything to avoid maintenance goal violation will not allow itself to achieve a highly valued goal on a path that will lead to such a violation. Such computation paths may be excluded by the the more powerful lookahead capability while still being allowed by the weaker one.

For the idealized situation where an agent has infinite lookahead, we have the following proposition.

**Proposition 2.** (Infinite Lookahead Maintenance Goal Semantics)

$$S_{\mathcal{A}}\restriction_\infty \Omega = \{t \in S_{\mathcal{A}} \mid \forall s \in t : \Sigma_s \models \Omega\}$$

This proposition states that an agent with infinite lookahead will only execute a computation that is completely free of maintenance goal violations. For the example carrier agent, if we assume infinite lookahead, any computation where the agent does not tank at location `ab1` are excluded from the semantics. The reason is that in these computations the agent will violate its maintenance goal as it will be at location `ab2` without any fuel.

Although the infinite lookahead semantics is elegant and captures intuitions in a simple manner, such lookahead cannot be implemented. In the next proposition we look at *bounded lookahead* where lookahead ranges are less than $\infty$.

**Proposition 3.** (Bounded Lookahead Maintenance Goal Semantics)
*Let $n \in \mathbb{N}$. The n-step lookahead semantics $S_{\mathcal{A}}\restriction_n \Omega$ is equal to:*

$$\bigcup_{t \in S_{\mathcal{A}}} \{p \prec t \mid safe_n(p) \ \&\ (\forall p', t' : p \preccurlyeq p' \prec t' \ \&\ safe_n(p') \Rightarrow p = p')\} \cup S_{\mathcal{A}}\restriction_\infty \Omega$$

**Corollary 1.** (One-Step Lookahead Maintenance Goal Semantics)
*The one-step lookahead semantics $S_{\mathcal{A}}\restriction_1 \Omega$ of an agent $\mathcal{A}$ is equal to:*

$$\bigcup_{t \in S_{\mathcal{A}}} \{p \prec t \mid (\forall s \in p : \Sigma_s \models \Omega) \ \&\ (\forall t' : p \prec t' \ \&\ s_{k+1} \in t' \Rightarrow \Sigma_{s_{k+1}} \not\models \Omega)\} \cup\ S_{\mathcal{A}}\restriction_\infty \Omega$$

Bounded lookahead implies that the agent may choose a path which inevitably will violate a maintenance goal because potential violations of the maintenance goal lie outside of the agent's lookahead range. As discussed above, it might be the case that on such a path an achievement goal is achieved that would never have been achieved if the agent would have had a greater lookahead range that would have predicted these violations. Note, however, that the fact that an agent takes a path on which it would violate a maintenance goal if it would continue still does *not* lead to violation of a maintenance goal. The reason is that the agent will be required to stop acting as soon as there are only actions enabled that would lead to a violation of a maintenance goal. This is in line with our assumption that maintenance goals are hard constraints.

In our example carrier agent it is sufficient to have a lookahead of one. As stated in Corollary 1, an agent with a lookahead range of one continues acting until it recognizes that by doing so at all possible next states it violates a maintenance goal. The carrier agent with a lookahead of one will be able to detect that if it executes a `move` action at location `ab1` before tanking, it will immediately violate its maintenance goal and will select the alternative action of tanking as a result. This illustrates that the lookahead mechanism, which primarily *constrains* the actions of the agent, may also induce the agent to *actively* prevent the violation of maintenance goals (in the example realized through tanking). To be more accurate, our mechanism does not distinguish between preventive actions that should prevent the violation of an achievement goal, and actions that are executed to fulfill achievement goals. As we can see in this example, in practice a very limited lookahead range may already be sufficient to prevent the agent from taking a path that would lead to violation of maintenance goals. To be more specific, the semantics of the example agent with lookahead range of one is equal to the semantics with lookahead range $\infty$. In general, the minimally needed lookahead range should be derived from available domain knowledge.

In this simple example, it is not difficult to modify the GOAL program in such a way that the desired behavior is obtained without explicitly incorporating maintenance goals. One could, e.g., add a condition to the conditional action for moving, specifying that if the agent is at location `ab1`, it may not move unless its tank is full. We argue, however, that the explicit incorporation of maintenance goals in the GOAL program provides a separation of concerns, and thereby potentially yields more transparent and easier to verify agent programs.

It is interesting to investigate under what circumstances bounded lookahead is guaranteed to be sufficient to avoid violation of maintenance goals. One particular such case is the case that an agent can *undo* actions, that is, if it has a *rollback mechanism* to go back to a previous state. In the presence of such a rollback mechanism, a bounded lookahead of 1 is sufficient to satisfy all maintenance goals. Obviously, the ability to rollback combined with 1 step lookahead will not be sufficient in all cases to realize the agent's achievement goals. The combination does allow the agent, however, to continue any computation given that at least one action is enabled. For our purposes, we model such a rollback mechanism simply by adding for each transition $s \rightarrow s'$ the inverse transition $s' \rightarrow s$ to the agent semantics.

**Theorem 1.** (Lookahead of One Sufficient with Rollback Mechanism)
*For agents that can do at least one action initially without violating a maintenance goal, and that have a rollback mechanism to undo arbitrary actions, that is, are able to reverse a computation step $s \rightarrow s'$ by performing the step $s' \rightarrow s$, we have the following:*

$$S_{\mathcal{A}}\restriction_1 \Omega = S_{\mathcal{A}}\restriction_\infty \Omega$$

*Proof. The main observation needed in the proof is that any finite, safe prefix can be continued without violating a maintenance goal by doing either a "regular" action or otherwise by doing an "undo" action. By assumption, the agent can at least do one action initially, and so any finite safe prefix can be extended to a complete computation that does not violate a maintenance goal.*

Although Theorem 1 shows that an agent will always be able to continue pursuing its goals, it does not state that it will also achieve these goals if possible. In the presence of a rollback mechanism, computations that make no progress but instead repeatedly have to recover from performing an action that leads to a violation of a maintenance goal are included in the set $S_{\mathcal{A}}\restriction_\infty \Omega$. What is missing is a notion of fairness that would prevent such repeated execution of a part of the computation (cf. [7]). Fairness is included in the original GOAL semantics but is not discussed further in this paper (cf. [5]). Intuitively, moreover, by using lookahead of more than one step computations that require rollback can be detected sooner which will reduce the need for such rollbacks.

## 5 Detecting and Revising Goal Conflicts

In this section an algorithm is presented that implements the maintenance goal semantics and, additionally, it includes an extension that provides the agent with the option to revise its achievement goals in case no achievement goal is reachable without choosing a path that would lead to violation of a maintenance goal. As discussed in Section 2.2, revising achievement goals is a way of dealing with conflicts between maintenance goals and achievement goals, if maintenance goals are taken as hard constraints. Revision of achievement goals is not the main subject of this paper (see e.g. [9]), but we will illustrate the main ideas using the carrier agent example.

```
Function SelectAction(E, s, n)
Input: A set of enabled conditional actions E, a state s, a lookahead range n
Output: A selected conditional action c, or skip
1.  actionOkSet ← ∅
2.  for each c ∈ E
3.      do conflict[c] ← ConflictSets(c, s, n)
4.          if ∅ ∈ conflict[c] then actionOkSet ← actionOkSet ∪ {c}
5.  if actionOkSet ≠ ∅
6.      then return ChooseAction(actionOkSet)
7.      else  c' ← SelectActionWithMinimalConficts(E, conflict)
8.            ReviseConflictingAchievementGoals(conflict[c'])
9.            (* do nothing and recompute enabled actions using revised achievement goal(s) *)
10.           return skip

Function ConflictSets(c, s, n)
Input: A conditional action c, a state s, and a lookahead range n
Output: The conflict sets of c
1.  if n ≤ 0
2.      then return {∅} (* Indicates that at least one path is ok. *)
3.      else  S ← SuccessorStates(c, s)
4.            for each s' ∈ S
5.                do cset ← ∅ (* Conflict set *)
6.                   if Σ_s' ⊭ Ω
7.                      then cset ← cset ∪ {ReasonConflict(c)}
8.                      else  E ← ComputeEnabledActions(s')
9.                            for each c' ∈ E
10.                               do cset ← cset ∪ ConflictSets(c', s', n − 1)
11.           return cset
```

**Table 3.** Action Selection Algorithm Including Maintenance Goals

The first step to implement the semantics for maintenance goals based on lookahead is to define an algorithm which is able to *detect* potential future maintenance goal violations. The algorithm depicted in Table 3 implements the detection of such violations as well as the cautious strategy of an agent that avoids taking a path that would lead to violation of a maintenance goal. The function SelectAction computes for each enabled conditional action whether it might result in any conflicts with or violations of maintenance goals for a given lookahead range $n$. In case executing an action does not inevitably lead to such a conflict, it is added to the set of actions that are *ok* to select for execution. Only if there are no actions that are "safe" in this sense, the action selection algorithm will select an achievement goal in order to revise it. The detection of these conflicts is done through the function ConflictSets. This function recursively computes the so-called conflict sets, which will be explained in more detail below. An empty conflict set indicates that no future violation of a maintenance goal within lookahead range is detected.

As discussed in Section 2.2, detected conflicts between achievement goals and maintenance goals may cause the agent not to do anything at a certain point, as it might be the case that any action would lead to a future violation of a maintenance goal. In the example scenario, adding a weight constraint that expresses that the truck cannot carry a load that weighs more than a certain threshold, has this effect if the sum of the weight of the two parcels is higher than the threshold (see Table 4, where `weightTotal(N)` computes the total weight of the parcels in the truck).

```
:beliefs{ ... weight(p1,3). weight(p2,2). threshold(4). weightTotal(N) :- ... }
:a-goals{ loc(p1,b), loc(p2,b). }
:m-goals{ fuel(X), X > 0. weightTotal(T), threshold(W), T<W. }
```

**Table 4.** GOAL Carrier Agent

If at least a lookahead of two is used, the agent will not be able to execute any action in the initial mental state. After loading either one of the parcels, loading the other one would lead to a violation of the weight maintenance goal. With the cautious strategy, taking a path on which the violation of a maintenance goal is foreseen within two steps, is not an option (note that the agent can only unload parcels at location `b`).

In this case where the agent cannot execute any action as this would lead to violation of maintenance goals, the algorithm of Table 3 allows the revision of achievement goals by means of lowering ones ambitions. The idea here is that actions are induced by achievement goals and these actions thus may be prevented from being taken by revising those goals (we disregard the possibility of incorrect beliefs, which might instead require an agent to revise its beliefs). In order to revise its achievement goals the agent needs more information to base the revision on and to this end the notion of a *conflict set* is introduced. A *conflict set* is an achievement goal $\alpha$ which has been identified as a potential reason for the violation of a maintenance goal. In general, identifying such a reason may involve complicated diagnostic reasoning, but in GOAL a more pragmatic solution is available. In GOAL, goal conditions are typically associated with the selection of actions and we can simply take these conditions as the reason why a maintenance goal is violated. In our example agent, the function ReasonConflict($c$) extracts *an instance* of the goal condition `loc(P,b)` as a reason for the violation of the maximum weight loaded. The function ReviseConflictingAchievementGoals then may revise the achievement goal in the goal base and drop one of the conjuncts to avoid the violation. Consecutively, the agent verifies again if the maintenance goal violation has been eliminated. If no reason can be identified in this way, # is returned to indicate a violation of a maintenance goal.

## 6 Conclusion and Related Work

In this paper, we have looked at a mechanism for agents to handle maintenance goals. In particular, we have proposed a formal semantics of maintenance goals based on the notion of lookahead, and we have analyzed the semantics by proving some properties, in order to gain a better understanding of the role of maintenance goals in action selection. We presented an algorithm for detecting maintenance goal violation, parametrized by a variable lookahead range in order to be able to control computational costs. Additionally, we have discussed the issue of achievement goal revision, in case the maintenance goals are so restrictive

that all courses of action for satisfying achievement goals will lead to a violation of maintenance goals.

There are several interesting directions for future research. Regarding the revision of achievement goals, several issues have remained unexplored. For example, we have suggested one possible way of determining that an achievement goal conflicts with a maintenance goal. In future research, we plan to investigate this approach and possible alternatives in more detail. One research direction in this respect is the investigation of existing techniques for determining whether achievement goals conflict with each other [16, 15, 13]. It will need to be investigated whether the issue of conflicts between maintenance goals and achievement goals is the same as or similar to the issue of conflicts between achievement goals.

Existing approaches for defining preferences over goals, such as in utility theory [1], may be useful to refine the strategy for revising achievement goals. Intuitively, an agent should revise its achievement goals in such a way that they are reachable without violating maintenance goals, and the revision should maximize the agents expected utility. Moreover, in this paper we have taken maintenance goals as hard constraints, and have suggested to revise achievement goals in case they conflict with the agent's maintenance goals. Alternatively, it could be allowed to violate maintenance goals under certain circumstances. Again utility theory could be useful here, in order to weigh the violation of a maintenance goal against the realization of an achievement goal. For example, negative utility could be associated with the violation of a maintenance goal assigning a maintenance goal that defines a hard constraint e.g. as having infinitely negative utility. The work in [8] on qualitative preferences in agent programming could also be relevant here. There are also some similarities with the planning literature on oversubscription (e.g. [2]), but as with planning approaches in general the main difference is that GOAL agents check violations of maintenance goals while executing actions.

Regarding related work on maintenance goals, we discuss the approach followed in the Jadex framework [3], the language presented by Dastani et al. [4], and the work of Duff et al. [6]. These approaches can be categorized into approaches that use maintenance goals as a *trigger* for the execution of actions, and approaches that use some mechanism for *reasoning* about the result of action execution in order to prevent maintenance goals from being violated. Jadex uses maintenance goals to trigger the execution of actions in case the maintenance goal is violated. In the framework of Dastani et al., a trigger condition is used to determine when action is needed to prevent the violation of maintenance goals. In our approach and in the framework of Duff et al., a reasoning mechanism is used in order to prevent maintenance goals from being violated.

One of the main differences between the work of Duff et al. and our work is that in Duff et al. it is determined *before* an achievement goal is pursued whether the plans for achieving this achievement goal may conflict with one of the agent's maintenance goals. In our work, by contrast, we propose to use a lookahead mechanism for keeping maintenance goals from being violated *during* pursuit of achievement goals. We also suggested the possibility to revise achievement goals

when they cannot be realized without violating maintenance goals, while Duff et al. propose to not adopt such achievement goals to avoid the risk of violating maintenance goals. The approaches also differ in that in this paper a mechanism to ensure satisfaction of maintenance goals is based on a semantic analysis and Duff et al. validate their work using an experimental approach.

Finally, an advantage of doing lookahead during achievement goal pursuit, we believe, is that it may provide for more flexible agent behavior. An approach based on executing a preventive plan that is associated with the maintenance goal in case an achievement goal might conflict with a maintenance goal, as proposed in Duff et al., does not seem to leave the agent with as many options as are possible. Moreover, such an approach still does not guarantee that the consecutive pursuit of the achievement goal will not violate the maintenance goal. The approach of Duff et al. can be compared with planning approaches, in the sense that reasoning takes place before execution. If something is about to go wrong during execution, this is not detected. In our approach, the agent pursues achievement goals, but takes any measures that it has at its disposal if this is necessary to prevent a maintenance goal from being violated.

## References


1. Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of AI Research*, 11:1–94, 1999.
2. Ronen I. Brafman and Yuri Chernyavsky. Planning with goal preferences and constraints. In *Proceedings of ICAPS'05*, 2006.
3. Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal representation for BDI agent systems. In *Programming multiagent systems, second international workshop (ProMAS'04)*, volume 3346 of *LNAI*, pages 44–65. Springer, Berlin, 2005.
4. Mehdi Dastani, M. Birna van Riemsdijk, and John-Jules Ch Meyer. Goal types in agent programming. In *Proceedings of the 17th European Conference on Artifical Intelligence 2006 (ECAI'06)*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 220–224. IOS Press, 2006.
5. F.S. de Boer, K.V. Hindriks, W. van der Hoek, and J.-J.Ch. Meyer. A Verification Framework for Agent Programming with Declarative Goals. *Journal of Applied Logic*, 2006. To appear.
6. Simon Duff, James Harland, and John Thangarajah. On Proactivity and Maintenance Goals. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS'06)*, pages 1033–1040, Hakodate, 2006.
7. N. Francez. *Fairness*. Springer, 1986.
8. Christian Fritz and Sheila A. McIlraith. Decision-theoretic golog with qualitative preferences. In *KR*, pages 153–163, 2006.
9. P. Gardenfors. *Belief Revision*. Cambridge Computer Tracts. Cambridge University Press, 1992.
10. Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Agent Programming with Declarative Goals. In *Proceedings of ATAL00*, volume 1986 of *LNCS*, pages 228–243, 2000.

11. Jomi Fred Hübner, Rafael H. Bordini, and Michael Wooldridge. Declarative goal patterns for AgentSpeak. In *Proceedings of the fourth International Workshop on Declarative Agent Languages and Technologies (DALT'06)*, 2006.
12. V. Lifschitz. On the semantics of strips. In M.P. Georgeff and A.L. Lansky, editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufman, 1986.
13. Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. A goal deliberation strategy for BDI agent systems. In *MATES 2005*, volume 3550 of *LNAI*, pages 82–93. Springer-Verlag, 2005.
14. Sebastian Sardina and Steven Shapiro. Rational action in agent programs with prioritized goals. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems (AAMAS'03)*, pages 417–424, Melbourne, 2003.
15. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 2003.
16. J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding resource conflicts in intelligent agents. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artifical Intelligence 2002 (ECAI 2002)*, Lyon, France, 2002.
17. M. Birna van Riemsdijk, Mehdi Dastani, John-Jules Ch Meyer, and Frank S. de Boer. Goal-oriented modularity in agent programming. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS'06)*, pages 1271–1278, Hakodate, 2006.
18. Micheal Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the eighth international conference on principles of knowledge respresentation and reasoning (KR2002)*, Toulouse, 2002.

# Conflict Resolution in Norm-Regulated Environments via Unification and Constraints

M. J. Kollingbaum[1,*], W. W. Vasconcelos[1,†], A. García-Camino[2,⋄], and T. J. Norman[1,‡]

[1]Dept. of Computing Science, Univ. of Aberdeen, Aberdeen AB24 3UE, UK
{*mkolling,† wvasconc,‡ tnorman}@csd.abdn.ac.uk
[2]IIIA-CSIC, Campus UAB 08193 Bellaterra, Spain
⋄andres@iiia.csic.es

**Abstract.** We present a novel mechanism for the detection and resolution of conflicts within norm-regulated virtual environments, populated by agents whose behaviours are regulated by explicit obligations, permissions and prohibitions. A conflict between norms arises when an action is simultaneously prohibited and obliged or prohibited and permitted. In this paper, we use first-order unification and constraint satisfaction to detect and resolve such conflicts, introducing a concept of norm curtailment. A flexible and robust algorithm for norm adoption is presented and aspects of indirect conflicts and conflicts across delegation of actions between agents is discussed.

## 1 Introduction

Norm-governed virtual organisations use obligations, permissions and prohibitions for the regulation of the behaviour of self-interested, heterogeneous software agents. Norms are important in the design and management of virtual organisations, as they allow a detailed specification of these social structures in terms of roles and the rights and duties of agents adopting these roles. Norm-regulated VOs, however, may experience problems when norms assigned to agents are in *conflict* – actions that are forbidden, may, at the same time, also be obliged and/or permitted. For example, a norm "Agent $X$ is permitted to $send\_bid(ag_1, 20)$" and "Agent $ag_2$ is prohibited from doing $send\_bid(Y, Z)$" (where $X$, $Y$ and $Z$ are variables and $ag_1$, $ag_2$ and 20 are constants) show two norms that are in conflict regarding an action *send_bid*.

In order to detect and resolve norm conflicts and to check norm-compliance of actions, we propose a mechanism based on first-order term unification [1] and constraint satisfaction. With that, we develop further the work presented in [2] where we used first-order term unification for conflict detection and norm annotations to avoid conflicts indicating what the variables of a prohibition *cannot* be when actions are deployed. In this paper, we also use unification, but add constraint satisfaction for conflict detection and resolution.

In the following section, we introduce a "lightweight" definition of virtual organisations and their enactments. In Section 3 we define norms, constraints and global normative states. Section 4 describes in detail a machinery for conflict detection and resolution. In section 5, we describe how agents check the norm-compliance of their actions with the use of unification and constraint satisfaction. Section 6 describes *indirect* conflicts occurring via domain-specific relationships between actions and via the delegation between roles. Section 7 describes the

application of the conflict resolution machinery in a detailed example. Section 8 provides an overview about related work and section 9 concludes this paper.

## 2 Virtual Organisations

Following [2], we base our discussion of norm conflicts on a simple representation of a virtual organisation [3] as a finite-state machine where actions of individual agents lead to state transitions. Figure 1 depicts a graphical representation of

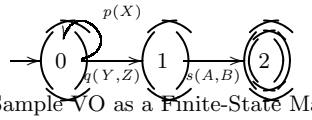


Fig. 1: Sample VO as a Finite-State Machine

this finite-state machine, where the edges between discrete states are labelled with first-order formulae representing actions performed by individual agents [1]. Although there are more sophisticated and expressive ways to represent agent activity and interaction (*e.g.*, AUML [5] and electronic institutions [6], to name a few), but for the sake of generalising our approach, we shall assume any higher-level formalism can be mapped onto a finite-state machine (possibly with some loss of expressiveness). A virtual organisation is defined as follows:

**Definition 1.** *A virtual organisation $\mathcal{I}$ is the tuple $\langle S, s_0, E, T\rangle$, where $S = \{s_1, \ldots, s_n\}$ is a finite and non-empty set of states, $s_0 \in S$ is the initial state, $E$ is a finite set of edges $(s, s', \varphi)$ with $s, s' \in S$ connecting $s$ to $s'$ and labelled with a first-order atomic formula $\varphi$, and $T \subseteq S$ is the set of terminal states.*

Notice that edges are directed, so $(s, t, \varphi) \neq (t, s, \varphi)$. The sample VO of Figure 1 is formally represented as $\mathcal{I} = \langle\{0, 1, 2\}, 0, \{(0, 0, p(X)), (0, 1, q(Y, Z)), (1, 2, s(A, B)\}, \{2\}\rangle$. We assume an implicit existential quantification on any variables in $\varphi$, so that, for instance, $s(A, B)$ stands for $\exists\, A, B\ s(A, B)$.

Roles, as exploited in, for instance, [7] and [6], define a pattern of behaviour to which any agent that adopts a role ought to conform. Moreover, all agents with the same role are guaranteed the same rights, duties and opportunities. We shall make use of two finite, non-empty sets, $Agents = \{ag_1, \ldots, ag_n\}$ and $Roles = \{r_1, \ldots, r_m\}$, representing, respectively, the sets of agent identifiers and role labels.

The specification of a VO as a finite-state machine gives rise to a possibly infinite set of histories of computational behaviours, in which the actions labelling the paths from the initial state to a final state are recorded. Although the actions comprising a VO are carried out distributedly, we propose an explicit global account of all events. In practice, this can be achieved if we require individual agents to declare/inform whatever actions they have carried out; this assumes trustworthy agents, naturally[2].

In order to record the authorship of the action, we annotate the formulae with the agents' unique identification. Our explicit global account of all events is

[1] We adopt Prolog's convention [4] and use strings starting with a capital letter to represent variables and strings starting with a small letter to represent constants.

[2] Non-trustworthy agents can be accommodated in this proposal, if we associate to each of them a *governor agent* which supervises the actions of the external agent and reports on them. This approach was introduced in [8] and is explained in section 5.

a set of ground atomic formulae $\bar{\varphi}$, that is, we only allow constants to appear as terms of formulae. Each formula is a truthful record of an action specified in the VO. Notice, however, that in the VO specification, we do not restrict the syntax of the formulae: variables may appear in them, and when an agent performs an actual action then any variables of the specified action must be assigned values. We thus define:

**Definition 2.** *A global execution state of a VO, denoted as $\Xi$, is a finite, possibly empty, set of tuples $\langle a : r, \bar{\varphi}, t\rangle$ where $a \in Agents$ is an agent identifier, $r \in Roles$ is a role label, $\bar{\varphi}$ is a ground first-order atomic formula, and $t \in \mathbb{N}$ is a time stamp.*

For instance, $\langle ag_1{:}buyer, p(a, 34), 20\rangle$ states that agent $ag_1$ adopting role *buyer* performed action $p(a, 34)$ at instant 20. Given a VO $\mathcal{I} = \langle S, s_0, E, T\rangle$, an execution state $\Xi$ and a state $s \in S$, we can define a function which obtains a possible next execution state, *viz.*, $h(\mathcal{I}, \Xi, s) = \Xi \cup \{\langle a{:}r, \bar{\varphi}, t\rangle\}$, for one $(s, s', \varphi) \in E$. Such a function $h$ must address the two kinds of non-determinism above, as well as the choice on the potential agents that can carry out the action and their adopted roles. We also define a function to compute the set of all possible execution states, $h^*(\mathcal{I}, \Xi, s) = \{\Xi \cup \{\langle a{:}r, \bar{\varphi}, t\rangle\} \mid (s, s', \varphi) \in E\}$.

The VO specification introduced previously must be augmented to accommodate the agent identification as well as its associated role. We thus have edges specified as $(s, s', \langle a, r, \varphi, t\rangle)$. More expressiveness can be achieved if we allow constraints (as introduced below) to be added to edges, as in, for instance, $(s, s', \langle a, r, (p(X, Y) \wedge X > Y), t\rangle)$, depicting that the formula $p(X, Y)$ causes the progress of the VO, provided $X > Y$. Such VOs are as expressive as the logic-based electronic institutions proposed in [9].

## 3 Norms

Norms are the central element in our discussion. We regard agents adopting specific roles and, with that, a set of norms that regulate their actions within a virtual organisation. We extend our previous work [2], and introduce a more expressive norm definition, accommodating constraints. We, again, adopt the notation of [10] for specifying norms and complement it with *constraints* [11]. By using constraints, we can *restrict* the influence of norms on specific parameters of actions. Our building blocks are first-order terms $\tau$, that is, constants, variables and functions (applied to terms). We shall make use of numbers and arithmetic functions to build those terms. Arithmetic functions may appear infix, following their usual conventions. Constraints are defined as follows:

**Definition 3.** *Constraints, generically represented as $\gamma$, are any construct of the form $\tau \lhd \tau'$, where $\lhd \in \{=, \neq, >, \geq, <, \leq\}$.*

We then introduce the syntax of norms:

**Definition 4.** *A norm $\omega$ is a tuple $\langle \nu, t_d, t_a, t_e\rangle$, where $\nu$ is any construct of the form $\mathsf{O}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (an obligation), $\mathsf{P}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (a permission) or $\mathsf{F}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ (a prohibition), where $\tau_1, \tau_2$ are terms, $\varphi$ is a first-order atomic formula and $\gamma_i$, $0 \leq i \leq n$, are constraints. The elements $t_d, t_a, t_e \in \mathbb{N}$ are, respectively, the time when $\nu$ was declared (introduced), when $\nu$ becomes active and when $\nu$ expires, $t_d \leq t_a \leq t_e$.*

Term $\tau_1$ identifies the agent(s) to whom the norm is applicable and $\tau_2$ is the role of such agent(s). $\mathsf{O}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n}\gamma_i$ thus represents an obligation on agent $\tau_1$ taking up role $\tau_2$ to bring about $\varphi$, subject to constraints $\gamma_i$, $0 \leq i \leq n$. The $\gamma_i$'s express constraints on those variables occurring in $\varphi$.

In the definition above, we only cater for conjunctions of constraints. If disjunctions are required then a norm must be established for each disjunct. For instance, if we required the norm $\mathsf{P}_{A:R}move(X) \wedge (X < 10 \vee X = 15)$ then we must break it into two norms $\mathsf{P}_{A:R}move(X) \wedge X < 10$ and $\mathsf{P}_{A:R}move(X) \wedge X = 15$. This holds because we assume an implicit universal quantification over variables in $\nu$. For instance, $\mathsf{P}_{A:R}p(X, b, c)$ stands for $\forall\, A \in Agents. \forall\, R \in Roles. \forall\, X.\mathsf{P}_{A:R}p(X, b, c)$. We comment on the existential quantification in the final section of this paper.

We propose to formally represent the normative positions of all agents, taking part in a virtual society, from a global perspective. By "normative position" we mean the "social burden" associated with individuals [8], that is, their obligations, permissions and prohibitions:

**Definition 5.** *A global normative state $\Omega$ is a finite and possibly empty set of tuples $\omega = \langle \nu, t_d, t_a, t_e \rangle$.*

As a simplification, we assume a single global normative state $\Omega$ for a virtual organisation. However, this can be further developed into a fully distributed form, with each agent maintaining its own $\Omega$, thus allowing the scaling up of our machinery.

Global normative states complement the execution states of VOs with information on the normative positions of individual agents. We can relate them via a function to obtain a norm-regulated next execution state of a VOs, that is, $g(\mathcal{I}, \Xi, s, \Omega, t) = \Xi'$, $t$ standing for the time of the update. For instance, we might want all prohibited actions to be excluded from the next execution state, that is, $g(\mathcal{I}, \Xi, s, \Omega, t) = \Xi \cup \{\langle a{:}r, \bar{\varphi}, t\rangle\}, (s, s', \varphi) \in E$ and $\langle \mathsf{F}_{a:r}\varphi, t_d, t_a, t_e\rangle \notin \Omega, t_a \leq t \leq t_e$. We might equally be interested that only permitted actions be chosen for the next execution state. We do not legislate, or indeed recommend, any particular way to regulate VOs. We do, however, offer simple underpinnings to allow arbitrary policies to be put in place. In the same way that a normative state is useful to obtain the next execution state of a VO, we can use an execution state to update a normative state. For instance, we might want to remove any obligation specific to an agent and role, which has been carried out by that specific agent and role, that is, $f(\Xi, \Omega) = \Omega - Obls$, $Obls = \{\langle \mathsf{O}_{a:r}\varphi, t_d, t_a, t_e\rangle \in \Omega \mid \langle a : r, \bar{\varphi}, t\rangle \in \Xi\}$. The management (*i.e.*, creation and updating) of global normative states is an interesting area of research. A simple and useful approach is reported in [12]: production rules generically depict how norms should be updated to reflect what agents have done and which norms currently hold. In this paper our focus is not proposing how $\Omega$'s should be managed, and assume some mechanism which does it.

## 4 Norm Conflicts

A conflict between two norms occurs if a formula representing an action is simultaneously *under the influence* of a permission and prohibition or an obligation and prohibition for the same agent (or set of agents) – the agent experiences a

normatively ambiguous situation for a specific set of actions. A norm *influences* who (what agent/set of agents in a specific role) is either permitted, prohibited or obliged to perform a specific action (or set of actions). We regard norms having a *scope of influence* as they may have an influence on a set of actions.

Figure 2 shows the scope of influence of a prohibition and a permission on instantiations of the action $shift(X, Y, Z), X \in \{a, b\}, Y \in \{r, s\}, Z \in \{u, v\}$,

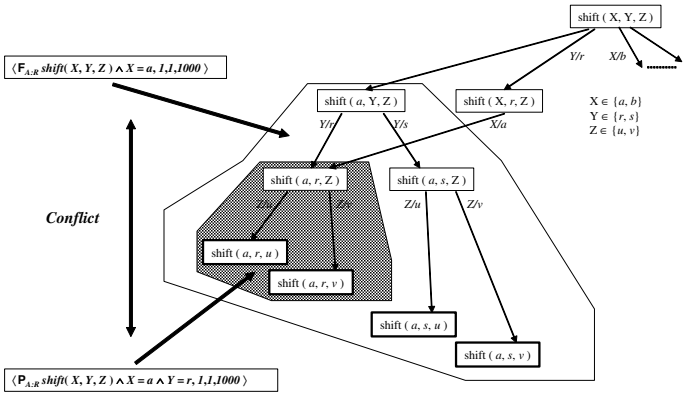



Fig. 2: Conflict between a Permission and a Prohibition

in a blocks world scenario, representing that block $X$ is shifted from the top of block $Y$ to the top of block $Z$. The prohibition prevents any agent in any role to shift a specific block $a$ from any block to any block. The scope of this prohibition is the portion of the action's space of possibilities enclosed within the larger irregular polygon. The diagram also shows the scope of a permission conflicting with this prohibition – it permits any agent in any role to shift a specific block $a$ from a specific block $r$ to any other block. The scope of influence of the permission is the portion of $shift$'s space of possibilities enclosed within the smaller grey irregular polygon, contained within the scope of the prohibition. This is a typical situation of conflict – the scopes of influence of both norms overlap.

We use *unification* of first-order terms [4, 1] as an instrument to detect and resolve conflicts between norms. Unification allows us *i)* to detect whether norms are in conflict and *ii)* to detect the set of actions that are under the influence of a norm. Unification is a fundamental problem in automated theorem proving and many algorithms have been proposed [1], recent work proposing means to obtain unifiers efficiently. Unification is based on the concept of substitution:

**Definition 6.** *A substitution $\sigma$ is a finite and possibly empty set of pairs $x/\tau$, where $x$ is a variable and $\tau$ is a term.*

We define the application of a substitution in accordance with [1] – a substitution $\sigma$ is a *unifier* of two terms $\tau_1 : \tau_2$, if $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$. In addition, we describe, how substitutions are applied to obligations, permissions and prohibitions. Below, $\mathsf{X}$ stands for either $\mathsf{O}, \mathsf{P}$ or $\mathsf{F}$:

1. $c \cdot \sigma = c$ for a constant $c$.
2. $x \cdot \sigma = \tau \cdot \sigma$ if $x/\tau \in \sigma$; otherwise $x \cdot \sigma = x$.

3. $p^n(\tau_0, \ldots, \tau_n) \cdot \sigma = p^n(\tau_0 \cdot \sigma, \ldots, \tau_n \cdot \sigma)$.
4. $(\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i) \cdot \sigma = (\mathsf{X}_{(\tau_1\cdot\sigma):(\tau_2\cdot\sigma)}\varphi \cdot \sigma) \wedge \bigwedge_{i=0}^{n} \gamma_i \cdot \sigma)$.
5. $\langle \nu, t_d, t_a, t_e \rangle \cdot \sigma = \langle (\nu \cdot \sigma), t_d, t_a, t_e \rangle$

We shall use unification in the following way:

**Definition 7.** *$unify(\tau_1, \tau_2, \sigma)$ holds for two terms $\tau_1, \tau_2$, iff $\tau_1 \cdot \sigma = \tau_2 \cdot \sigma$ holds, for some $\sigma$; $unify(p^n(\tau_0, \ldots, \tau_n), p^n(\tau'_0, \ldots, \tau'_n), \sigma)$ holds, for two atomic formulae $p^n(\tau_0, \ldots, \tau_n), p^n(\tau'_0, \ldots, \tau'_n)$, iff $unify(\tau_i, \tau'_i, \sigma), 0 \leq i \leq n$, for some $\sigma$.*

We assume that *unify* is based on a suitable implementation of a unification algorithm that *i)* always terminates (possibly failing, if a unifier cannot be found), *ii)* is correct and *iii)* is of linear computational complexity. The *unify* relationship checks, on the one hand, that substitution $\sigma$ is a unifier, but can also be used to find $\sigma$. By extending the definition of *unify* for handling norms, we can use unification for detecting a conflict between two norms ($\mathsf{X}, \mathsf{X}'$, again, stand for either $\mathsf{O}, \mathsf{P}$ or $\mathsf{F}$):

**Definition 8.** *$unify(\omega, \omega')$ holds for two norms $\omega = \langle (\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i), T_a, T_d, T_e \rangle$ and $\omega' = \langle (\mathsf{X}'_{\tau'_1:\tau'_2}\varphi' \wedge \bigwedge_{j=0}^{m} \gamma'_j), T'_a, T'_d, T'_e \rangle$, iff*

1. *$unify(\langle \tau_1, \tau_2, \varphi, T_a, T_d, T_e \rangle, \langle \tau'_1, \tau'_2, \varphi', T'_a, T'_d, T'_e \rangle, \sigma)$ and*
2. *$satisfy((\bigwedge_{i=0}^{n}(\gamma_i \cdot \sigma)) \wedge (\bigwedge_{j=0}^{m}(\gamma'_j \cdot \sigma)))$*

Two conditions are tested: the first one checks that the various components of a norm, organised as a tuple, unify; the second one checks that the constraints associated with the norms are satisfiable[3].

### 4.1 Conflict Detection

With unification, we can detect whether norms are in conflict. We define formally a conflict between norms as follows:

**Definition 9.** *A conflict arises between $\omega, \omega' \in \Omega$ under a substitution $\sigma$, denoted as* **conflict**$(\omega, \omega', \sigma)$*, iff the following conditions hold:*

1. *$\omega = \langle (\mathsf{F}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i), t_d, t_a, t_e \rangle$, $\omega' = \langle (\mathsf{O}_{\tau'_1:\tau'_2}\varphi' \wedge \bigwedge_{i=0}^{n} \gamma'_i), t'_d, t'_a, t'_e \rangle$,*
2. *$unify(\langle \tau_1, \tau_2, \varphi \rangle, \langle \tau'_1, \tau'_2, \varphi' \rangle, \sigma)$, $satisfy(\bigwedge_{i=0}^{n} \gamma_i \wedge (\bigwedge_{i=0}^{m} \gamma'_i \cdot \sigma))$*
3. *$overlap(t_a, t_e, t'_a, t'_e)$.*

That is, a conflict occurs if *i)* a substitution $\sigma$ can be found that unifies the variables of two norms[4], and *ii)* the conjunction $\bigwedge_{i=0}^{n} \gamma_i \wedge (\bigwedge_{i=0}^{m} \gamma'_i) \cdot \sigma)$ of constraints from both norms can be satisfied (taking $\sigma$ under consideration), and *iii)* the activation period of the norms overlap. The *overlap* relationship holds if *i)* $t_a \leq t'_a \leq t_e$; or *ii)* $t'_a \leq t_a \leq t'_e$. For instance, for the two norms $\mathsf{P}_{A:R}p(c, X) \wedge X > 50$ and $\mathsf{F}_{a:b}p(Y, Z) \wedge Z < 100$, a substitution $\sigma = \{A/a, R/b, Y/c, X/Z\}$ can be found that allows the unification of both norms – being able to construct such a

[3] We assume an implementation of the *satisfy* relationship based on "off the shelf" constraint satisfaction libraries such as those provided by SICStus Prolog [13–15] and it holds if the conjunction of constraints is satisfiable.

[4] A similar definition is required to address the case of conflict between a prohibition and a permission – the first condition should be changed to $\omega' = \langle (\mathsf{P}_{\tau'_1:\tau'_2}\varphi' \wedge \bigwedge_{i=0}^{n} \gamma'_i), t'_d, t'_a, t'_e \rangle$. The rest of the definition remains the same.

unifier is a first indication that there may be a conflict, expressed as an overlap of their influence on actions. The unifier expresses that the two norms conflict if the variables $A, R, Y$ and $X$ receive as bindings the values contained in the unifier. On the other hand, there will be no conflict if different bindings are chosen. The constraints on the norms may restrict this overlap and, therefore, leave actions under certain variable bindings free of conflict. The constraints of both norms have to be investigated to see if an overlap of the values indeed occurs. In our example, the permission has a constraint $X > 50$ and the prohibition has $Z < 100$. By using the substitution $X/Z$, we see that $50 < X < 100$ and $50 < Z < 100$ represent ranges of values for variables $X$ and $Z$ where a conflict will occur.

For convenience (and without any loss of generality) we assume that our norms are in a special format: any non-variable term $\tau$ occurring in $\nu$ is replaced by a fresh variable $X$ (not occurring anywhere in $\nu$) and a constraint $X = \tau$ is added to $\nu$. This transformation can be easily automated by scanning $\nu$ from left to right, collecting all non-variable terms $\{\tau_1, \ldots, \tau_n\}$; then we add $\wedge_{i=1}^{n} X_i = \tau_i$ to $\nu$. For example, norm $\mathsf{P}_{A:R}p(c, X) \wedge X > 50$ is transformed into $\mathsf{P}_{A:R}p(C, X) \wedge X > 50 \wedge C = c$.

### 4.2 Conflict Resolution

In order to resolve a conflict with respect to a specific action that is located in the overlap of the scopes of influence of both norms, a social entity has to decide which of the two conflicting norms it should adhere and which it should ignore. For a software agent, a machinery has to be put in place that computes a possible disambiguation of its normative situation – the set of norms $\Omega$ has to be transformed into a set $\Omega'$ that does not contain any conflicting norms so that the agent can proceed with its execution. In [2], we achieved this by using a concept of *curtailment* – one of the norms is changed in a way so that its scope of influence is retracted from specific actions (which norm to choose for curtailment is a different matter and not discussed in this paper). By curtailing the scope of influence of a norm, the overlap between the two norms is eliminated.

Extending [2], we achieve curtailment by manipulating the constraints of the norms. In figure 3, we show how a curtailment of the prohibition changes its scope

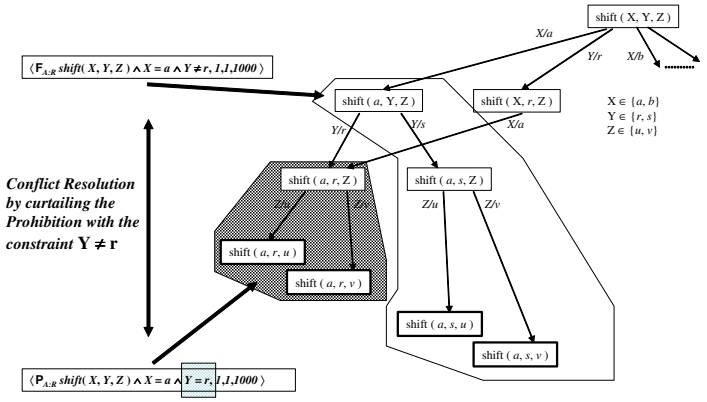



Fig. 3: Conflict Resolution with Curtailment

of influence and thus eliminates the overlap between the two norms. Specific constraints are added to the prohibition in order to perform this curtailment – as shown in figure 3, these additional constraints are derived from the permission. The scope of the permission is determined by the constraints $X = a$ and $Y = r$, restricting the set of bindings for variables $X$ and $Y$ to values $a$ and $r$. Adding a constraint $Y \neq r$ to the prohibition curtails its scope of influence and eliminates the overlap with the scope of influence of the permission.

We now formally define how the curtailment of norms takes place. It is important to notice that the curtailment of a norm creates a new (possibly empty) set of curtailed norms:

**Definition 10.** *Relationship* **curtail**$(\omega, \omega', \Omega)$*, where* $\omega = \langle \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i, t_d, t_a, t_e \rangle$ *and* $\omega' = \langle \mathsf{X}'_{\tau'_1:\tau'_2}\varphi' \wedge \bigwedge_{j=0}^{m} \gamma'_j, t'_d, t'_a, t'_e \rangle$ *(*$\mathsf{X}$ *and* $\mathsf{X}'$ *being either* $\mathsf{O}, \mathsf{F}$ *or* $\mathsf{P}$*) holds iff* $\Omega$ *is a possibly empty and finite set of norms obtained by curtailing* $\omega$ *with respect to* $\omega'$*. The following cases arise:*

1. *If* **conflict**$(\omega, \omega', \sigma)$ *does not hold then* $\Omega = \{\omega\}$*, that is, the set of curtailments of a non-conflicting norm* $\omega$ *is* $\omega$ *itself.*
2. *If* **conflict**$(\omega, \omega', \sigma)$ *holds, then* $\Omega = \{\omega^c_0, \ldots, \omega^c_m\}$*, where* $\omega^c_j = \langle \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge (\neg\gamma'_j \cdot \sigma), t_d, t_a, t_e \rangle$*,* $0 \leq j \leq m$*.*

The rationale for the definition above is as follows. In order to curtail $\omega$ thus avoiding any overlapping of values its variables may have with those variables of $\omega'$, we must "merge" the negated constraints of $\omega'$ with those of $\omega$. Additionally, in order to ensure the appropriate correspondence of variables between $\omega$ and $\omega'$ is captured, we must apply the substitution $\sigma$ obtained via **conflict**$(\omega, \omega', \sigma)$ on the merged negated constraints. By combining the constraints of $\nu = \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i$ and $\nu' = \mathsf{X}'_{\tau'_1:\tau'_2}\varphi' \wedge \bigwedge_{j=0}^{m} \gamma'_j$, we obtain the curtailed norm $\nu^c = \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\bigwedge_{j=0}^{m} \gamma'_j \cdot \sigma)$. The following equivalences hold:

$$\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\bigwedge_{j=0}^{m} \gamma'_j \cdot \sigma) \equiv \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge (\bigvee_{j=0}^{m} \neg\gamma'_j \cdot \sigma)$$

That is, $\bigvee_{j=0}^{m}(\mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg(\gamma'_j \cdot \sigma))$. This shows that each constraint of $\nu'$ leads to a possible solution for the resolution of a conflict and a possible curtailment of $\nu$. The curtailment thus produces a set of curtailed norms $\nu^c_j = \mathsf{X}_{\tau_1:\tau_2}p(t_1, \ldots, t_n) \wedge \bigwedge_{i=0}^{n} \gamma_i \wedge \neg\gamma'_j \cdot \sigma, 0 \leq j \leq m$. Although each of the $\nu^c_j$, $0 \leq j \leq m$, represents a solution to the norm conflict, we advocate that *all* of them have to be added to $\Omega$ in order to replace the curtailed norm. This would allow a preservation of as much of the original scope of the curtailed norm as possible. During the formation of a conflict-free $\Omega'$, the agent has to choose which norm to curtail in case of a conflict. In order to express such a choice, we introduce the concept of special curtailment *policies* that determine, given a pair of norms, which norm to curtail. We define curtailment policies as:

**Definition 11.** *A policy* $\pi$ *is a tuple* $\langle \omega, \omega', (\bigwedge_{i=0}^{n} \gamma_i) \rangle$ *establishing that* $\omega$ *should be curtailed (and* $\omega'$ *should be preserved), if* $(\bigwedge_{i=0}^{n} \gamma_i)$ *hold.*

For example, a policy $\langle\langle \mathsf{F}_{A:R}p(X,Y), T_d, T_a, T_e\rangle, \langle \mathsf{P}_{A:R}p(X,Y), T'_d, T'_a, T'_e\rangle, (T_d < T'_d)\rangle$ expresses that any prohibition held by any agent that corresponds to the pattern $\mathsf{F}_{A:R}p(X,Y)$ has to be curtailed, if the additional constraint, which expresses that the prohibition's time of declaration $T_d$ precedes that of the permission's $T'_d$, holds. Adding constraints to policies allows us a fine-grained control of conflict resolution, capturing classic forms of resolving deontic conflicts – the constraint in the example establishes a precedence relationship between the two norms that is known as *legis posterior* (see section 8 for more details). We shall represent a set of such policies as $\Pi$.

The algorithm shown in figure 4 depicts how to obtain a conflict-free set of norms. It describes how an originally conflict-free (possibly empty) set $\Omega$ can be extended in a fashion that resolves any emerging conflicts during norm adoption. With that, a conflict-free $\Omega$ is always transformed into a conflict-free

```
algorithm adoptNorm(ω, Ω, Π, Ω')
input ω, Ω, Π
output Ω'
begin
  Ω' := ∅
  if Ω = ∅ then Ω' := Ω ∪ {ω}
  else
    for each ω' ∈ Ω do
      // test for conflict
      if unify(ω, ω', σ) then
        // test policy
        if ⟨ω_π, ω'_π, (⋀_{i=0}^{n} γ_i)⟩ ∈ Π and unify(ω, ω_π, σ) and unify(ω', ω'_π, σ) and
          satisfy(⋀_{i=0}^{n}(γ_i · σ))
        then
          curtail(ω, ω', Ω'')
            Ω' := Ω ∪ Ω''
        else
          // test policy
          if ⟨ω'_π, ω_π, (⋀_{i=0}^{n} γ_i)⟩ ∈ Π and unify(ω, ω_π, σ) and unify(ω', ω'_π, σ) and
            satisfy(⋀_{i=0}^{n}(γ_i · σ))
          then
            curtail(ω', ω, Ω'')
            Ω' := (Ω − {ω'}) ∪ ({ω} ∪ Ω'')
          endif
        endif
      endif
    endfor
  endif
end
```

Fig. 4: Norm Adoption Algorithm

$\Omega'$ that may contain curtailments. The algorithm makes use of a set $\Pi$ of policies determining how the curtailment of conflicting norms should be done. Policies determine whether the new norm $\omega$ is curtailed in case of a conflict or whether a curtailment of one of the existing $\omega' \in \Omega$ should take place. When a norm is curtailed, a set of new norms replace the original norm. This set of norms is collected into $\Omega''$ by $\mathbf{curtail}(\omega, \omega', \Omega'')$. A curtailment takes place if there is a conflict between $\omega$ and $\omega'$. This test creates a unifier $\sigma$ that is re-used in the policy test. When checking for a policy that is applicable, the algorithm uses unification to check *(a)* whether $\omega$ matches/unifies with $\omega_\pi$ and $\omega'$ with $\omega'_\pi$; and *(b)* whether the policy constraints hold under the given $\sigma$. If a previously agreed policy in $\Pi$ determines that the newly adopted norm $\omega$ is to be curtailed in case of a conflict with an existing $\omega' \in \Omega$, then the new set $\Omega'$ is created by adding $\Omega''$ (the curtailed norms) to $\Omega$. If the policy determines a curtailment of an existing $\omega' \in \Omega$ when a conflict arises with the new norm $\omega$, then a new set $\Omega'$ is formed by *a)* removing $\omega'$ from $\Omega$ and *b)* adding $\omega$ and the set $\Omega''$.

## 5 Norm-Aware Agent Societies

With a set $\Omega$ that reflects a conflict-free normative situation, the agent can test whether its actions are norm-compliant. In order to check actions for norm-compliance, we, again, use unification. If an action unifies with a norm, then it is within its scope of influence:

**Definition 12.** *$\langle a : r, \bar{\varphi}, t\rangle$, is within the scope of influence of $\langle \mathsf{X}_{\tau_1:\tau_2}\varphi \wedge \bigwedge_{i=0}^{n}\gamma_i, t_d, t_a, t_e\rangle$ (where $\mathsf{X}$ is either $\mathsf{O}$, $\mathsf{P}$ or $\mathsf{F}$) iff the following conditions hold:*

1. *$unify(a, \tau_1, \sigma)$, $unify(r, \tau_2, \sigma)$, $unify(\bar{\varphi}, \varphi, \sigma)$ and $satisfy(\bigwedge_{i=0}^{n}\gamma_i \cdot \sigma)$*
2. *$t_a \leq t \leq t_e$*

This definition can be used to establish a predicate `check`/2, which holds if its first argument, a candidate action (in the format of the elements of $\Xi$ of Def. 2), is within the influence of an prohibition $\omega$, its second parameter. Figure 5 shows

$$
\begin{array}{l}
\texttt{check}(Action, \omega) \leftarrow \\
\quad Action = \langle a : r, \bar{\varphi}, t\rangle \wedge \\
\quad \omega = \langle (\mathsf{F}_{\tau_1:\tau_2}\varphi' \wedge \bigwedge_{i=0}^{n}\gamma_i), t_d, t_a, t_e\rangle \wedge \\
\quad unify(\langle a, r, \bar{\varphi}\rangle, \langle \tau_1, \tau_2, \varphi'\rangle, \sigma) \wedge satisfy(\bigwedge_{i=0}^{n}\gamma_i \cdot \sigma) \wedge \\
\quad t_a \leq t \leq t_e
\end{array}
$$

Fig. 5: Check if Action is within Influence of a Prohibition

the definition of this relationship as a logic program. Similarly to the check of conflicts between norms, it tests *i)* if the agent performing the action and its role unify with the appropriate terms $\tau_1, \tau_2$ of $\omega$; *ii)* if the actions $\bar{\varphi}, \varphi$ themselves unify; and *iii)* the conjunction of the constraints of both norms can be satisfied, all under the same unifier $\sigma$. Lastly, it checks if the time of the action is within the norm temporal influence.

## 6 Indirect Conflicts

In our previous discussion, norm conflicts were detected via a direct comparison of atomic formulae representing actions. However, conflicts and inconsistencies may also arise *indirectly* via relationships among actions. For instance, if we consider that an agent holds the two norms $\mathsf{P}_{A:R}p(X)$ and $\mathsf{F}_{A:R}q(X, X)$ and that the action $p(X)$ amounts to the action $q(X, X)$, then we can rewrite the permission as $\mathsf{P}_{A:R}q(X, X)$ and identify an *indirect* conflict between these two norms. We use a set of *domain axioms* in order to declare such domain-specific relationships between actions:

**Definition 13.** *The set of domain axioms, denoted as $\Delta$, are a finite and possibly empty set of formulae $\varphi \rightarrow (\varphi'_1 \wedge \cdots \wedge \varphi'_n)$ where $\varphi, \varphi'_i, 1 \leq i \leq n$, are atomic first-order formulae.*

In order to accommodate indirect conflicts between norms based on domain-specific relationships of actions, we have to adapt our curtailment mechanism. A curtailment occurs, if there is a conflict, that is, if for two norms $\omega$ and $\omega'$, their variables unify, the conjunction of their constraints can be satisfied and their activation periods overlap. With the introduction of domain axioms, this test has to be performed for each of the conjuncts in the relationship. For example, if we have a set of domain axioms $\Delta = \{(p(X) \rightarrow q(X, X) \wedge r(X, Y))\}$

and a permission $\langle \mathsf{P}_{A:R} p(X), t_d, t_a, t_e \rangle$ then $q(X, X)$ and $r(X, Y)$ are also permitted. There is, thus, an indirect conflict between $\langle \mathsf{P}_{A:R} p(X), t_d, t_a, t_e \rangle$ and $\langle \mathsf{F}_{A:R} q(X, X), t_d, t_a, t_e \rangle$ and $\langle \mathsf{F}_{A:R} r(X, Y), t_d, t_a, t_e \rangle$.

Domain axioms may also accommodate the delegation of actions between agents. Such a delegation transfers norms across the agent community and, with that, also conflicts. We introduce a special logical operator $\varphi \xrightarrow{\tau_1:\tau_2\ \tau_1':\tau_2'} (\varphi_1' \wedge \cdots \wedge \varphi_n')$ to represent that agent $\tau_1$ adopting role $\tau_2$ can transfer any norms on action $\varphi$ to agent $\tau_1'$ adopting role $\tau_2'$, which should carry out actions $\varphi_1' \wedge \cdots \wedge \varphi_n'$ instead.

## 7 Example: Agents for the Grid

We address a scenario taken from the e-Science/Grid domain in which a service provider may request payment that introduces a financial obligation for users, but, at the same time commits to the provision of the service that represents a right for the user to access the service.

In this scenario, a Principal Investigator (PI) of a research project has to perform a specific research task that involves the analysis of data. We assume that a contract exists between the PI and the funding body that introduces certain rights, restrictions and obligations for the contracting partners. We regard both the PI and the funding body as being represented as agents operating on the Grid and that this contract is available in electronic form and taken into account by the agents in their actions.

$$\left\{ \begin{array}{l} \langle \mathsf{F}_{rsa:pi} claim(X), 1, 1, 1000 \rangle \\ \langle \mathsf{P}_{rsa:pi} claim(staff\_costs), 1, 1, 1000 \rangle \\ \langle \mathsf{P}_{rsa:pi} claim(travel), 1, 1, 1000 \rangle \\ \langle \mathsf{O}_{rsa:pi} report\_experiment(rsa, D), 1, 1, 1000 \rangle \\ \langle \mathsf{F}_{X:Y} publish(D), 1, 1, 1000 \rangle \end{array} \right\}$$

**Fig. 6:** Contract $C$

A possible initial contract $C$ is shown in Fig. 6. The first three norms represent financial requirements of the agent taking on the principal investigator role. All claims are prohibited (norm 1) with the exception of a number of specific types of item: staff costs (norm 2) and travel costs (norm 3) are itemised here. In addition, an obligation is stated that requires the PI to report about the experiment as well as a prohibition for anybody to publish data. The last norm is a basic prohibition, forbidding any agent in any role to publish data. Contract $C$ in its alternative (equivalent) format in which constants are replaced by variables and constraints is shown in Fig. 7.

$$\left\{ \begin{array}{l} \langle \mathsf{F}_{A:R}\ claim(X) \wedge \left( \begin{smallmatrix} A = rsa \wedge \\ R = pi \end{smallmatrix} \right), 1, 1, 1000 \rangle \\ \langle \mathsf{P}_{A:R}\ claim(X) \wedge \left( \begin{smallmatrix} A = rsa \wedge R = pi \wedge \\ X = staff\_costs \end{smallmatrix} \right), 1, 1, 1000 \rangle \\ \langle \mathsf{P}_{A:R}\ claim(X) \wedge \left( \begin{smallmatrix} A = rsa \wedge \\ R = pi \wedge \\ X = travel \end{smallmatrix} \right), 1, 1, 1000 \rangle \\ \langle \mathsf{O}_{A:R}\ report\_experiment(A, D) \wedge \left( \begin{smallmatrix} A = rsa \wedge \\ R = pi \wedge \end{smallmatrix} \right), 1, 1, 1000 \rangle \\ \langle \mathsf{F}_{X:Y}\ publish(D), 1, 1, 1000 \rangle \end{array} \right\}$$

**Fig. 7:** Alternative Format of Contract $C$

### 7.1 Conflict Resolution

Contract $C$ has conflicting norms. We use our machinery to obtain a conflict-free version $C'$ of it, in which only the first prohibition is curtailed. $C'$ is shown in Fig. 8. In our example, two Grid services are made available by two potential subcontractors for the execution of the data analysis task. These are: *i)* a public non-profit organisation provides a free service, but

$$\left\{ \begin{array}{l} \langle \mathsf{F}_{A:R}\ claim(X) \wedge \left( \begin{smallmatrix} A = rsa \wedge R = pi \wedge \\ X \neq staff\_costs \wedge \\ X \neq travel \end{smallmatrix} \right), 1, 1, 1000 \rangle \\ \langle \mathsf{P}_{A:R}\ claim(X) \wedge \ldots, 1, 1, 1000 \rangle \\ \vdots \\ \langle \mathsf{F}_{X:Y}\ publish(D), 1, 1, 1000 \rangle \end{array} \right\}$$

**Fig. 8:** Contract $C'$ with Curtailed Norm

requires the disclosure of data in a public repository; and *ii)* a private commercial organisation provides the service without the need for disclosure, but requests a payment. These conditions of use can be expressed as norms in our formalism. The terms of the service, provided by the public non-profit organisation, are $N_1 = \{\langle \mathsf{O}_{A:R}\ publish(D'), 1, 1, 1000\rangle\}$, that is, according to the terms of conditions of the public service, the input data have to be published. The terms of the service of the private commercial organisation, on the other hand, are $\langle \mathsf{O}_{A:R}\ pay(fee), 1, 1, 1000\rangle$ or, alternatively, $N_2 = \{\langle \mathsf{O}_{A:R}\ pay(X) \wedge X = fee, 1, 1, 1000\rangle\}$ That is, whoever uses the service is obliged to pay a fee. The Research Assistant Agent (*rsa*) has to choose which service to use. Each of them introduces a new obligation with associated inconsistencies, explained below.

If the public Grid service is chosen, then the set $N_1$, containing a new obligation, is introduced. The set $C' \cup N_1$ contains a conflict: the obligation to publish overlaps with the influence of the prohibition to publish. Our machinery handles this, completely curtailing the prohibition and giving rise to a new set $C''$, shown in Fig. 9. The constraint $D \neq D'$ expresses that variable $D$ cannot be bound to anything (since $D'$ is a free variable) – the prohibition, therefore, becomes completely curtailed and has no effect any more and, hence, it is removed.

$$\left\{\begin{array}{l}\langle \mathsf{F}_{A:R}\ claim(X) \wedge \ldots, 1, 1, 1000\rangle\\ \langle \mathsf{P}_{A:R}\ claim(X) \wedge \ldots, 1, 1, 1000\rangle\\ \langle \mathsf{P}_{A:R}\ claim(X) \wedge \ldots, 1, 1, 1000\rangle\\ \langle \mathsf{O}_{A:R}\ report\_experiment(A, D) \ldots, 1, 1, 1000\rangle\\ \langle \mathsf{F}_{X:Y}\ publish(D) \wedge D \neq D', 1, 1, 1000\rangle\\ \langle \mathsf{O}_{A:R}\ publish(D'), 1, 1, 1000\rangle\end{array}\right\}$$

**Fig. 9:** Contract $C'' = C' \cup N_1$

A conflict within the set $C' \cup N_2$ is not immediately obvious. Intuitively, in terms of paying expenses for research (the domain of discussion here), the action *pay* is related to the action *claim*. In order for our mechanism to cope with such a situation, a concept of *indirect* conflicts based on domain axioms for relating actions has to be introduced. We have explored such indirect conflicts in [2] and we plan to extend that work to handle arbitrary constraints.

### 7.2 Indirect Conflict Resolution

In choosing the private service, the obligation $N_2 = \{\langle \mathsf{O}_{A:R}\ pay(X) \wedge X = fee, 1, 1, 1000\rangle\}$ is introduced and a contract $C'' = C' \cup N_2$ created. Intuitively, we know that this introduces an *indirect* conflict, as the original contract does not allow such a claim. With a domain axiom, we can express that to pay for something eventually amounts to claiming it: $\Delta = \{pay(X) \xrightarrow{A:R\ \ A:R} claim(X)\}$. In contract $C''$, we have to permissions that allow claiming staff costs and travel, but not claiming fees. According to the given domain axiom, obligation $N_2$ can be transformed into $N_2^{\Delta} = \mathsf{O}_{A:R}\ claim(X) \wedge X = fee, 1, 1, 1000\rangle\}$. By forming a new contract $C'' = C' \cup N_2^{\Delta}$, a direct conflict between the first prohibition regarding claims and obligation $N_2^{\Delta}$ arises (Fig. 10). The conflict resolution can now take place as shown in the case of *direct* conflicts (see contract $C'$ in Fig. 8).

$$\left\{\begin{array}{l}\langle \mathsf{F}_{A:R}\ claim(X) \wedge \ldots, 1, 1, 1000\rangle\\ \langle \mathsf{P}_{A:R}\ claim(X) \wedge \ldots, 1, 1, 1000\rangle\\ \langle \mathsf{P}_{A:R}\ claim(X) \wedge \ldots, 1, 1, 1000\rangle\\ \langle \mathsf{O}_{A:R}\ report\_experiment(A, D) \ldots, 1, 1, 1000\rangle\\ \langle \mathsf{F}_{X:Y}\ publish(D) \wedge D \neq D', 1, 1, 1000\rangle\\ \langle \mathsf{O}_{A:R}\ claim(X) \wedge X = fee, 1, 1, 1000\rangle\end{array}\right\}$$

**Fig. 10:** Contract $C'' = C' \cup N_2^{\Delta}$

### 7.3 Solving Conflicts arising from Delegation

Conflicts can also arise from delegation among agents/roles. Let there be the set of domain axioms $\Delta$ of Fig. 11: it contains axioms describing how the Research Assistant Agent can fulfil its obligation to report the result of an

experiment. As the domain axioms show, there is a relationship between the action *report_experiment* and *do_exp*. An additional axiom tells us that the action *do_exp* leads to the sending of experimental data to one of the chosen Grid services of subcontractors. The domain axiom $send(A, R', E, D) \xrightarrow{A:R \;\; A':R'} receive(A', R', A, E, D)$ shows the delegation of activities from the agent responsible for the data analysis to a subcontractor for actually performing the experiment. The rest of the domain axioms describe how a subcontractor performs an experiment and sends back results upon receiving such a request. For example, the obligation to report experimental results gives rise to an obligation to perform the action *do_exp* and, continuing in this transitive fashion, obligations for all the related actions as described before. Due to the delegation step, obligations also arise for the partner agents. These obligations, in their turn, may interfere with prohibitions held by the collaborating agents and may have to be dealt with in the same way.

$$\left\{ \begin{array}{l} pay(X) \xrightarrow{A:R \;\; A:R} claim(X) \\ report\_experiment(A, E, D) \xrightarrow{A:R \;\; A:R} do\_exp(A, E, D) \\ do\_exp(A, e_1, D) \xrightarrow{A:pi \;\; A:pi} send(A, exp, e_1, D) \\ send(A, R', E, D) \xrightarrow{A:R \;\; A':R'} receive(A', R', A, E, D) \\ receive(A', R', A, E, D) \xrightarrow{A':R' \;\; A':R'} \left( \begin{array}{c} analyse(A', E, D, S) \wedge \\ send(A, A', S) \end{array} \right) \end{array} \right\}$$

**Fig. 11:** Set of Domain Axioms $\Delta$

## 8 Related Work

The work presented in this paper is an extension and adaptation of the work presented in [2, 16] and [17]. It can also be seen as a logic-theoretic investigation into deontic logics to represent normative modalities along with their paradoxes [18, 19]. In [2], we introduced conflict detection and resolution based on unification. In this paper, we re-visited this research and introduced constraints into the given conflict detection/resolution mechanism. The result is a new machinery for conflict detection/resolution and reported in this paper.

Efforts to keep law systems conflict-free can be traced back to the jurisprudential practice in human society. Inconsistency in law is an important issue and legal theorists use a diverse set of terms such as, for example, normative inconsistencies/conflicts, antinomies, discordance, etc., in order to describe this phenomenon. There are three classic strategies for resolving deontic conflicts by establishing a precedence relationship between norms: *legis posterior* – the most recent norm takes precedence, *legis superior* – the norm imposed by the strongest power takes precedence, and *legis specialis* – the most specific norm takes precedence [20]. The work presented in [16] discusses a set of conflict scenarios and conflict resolution strategies, among them the classic strategies mentioned above. For example, one of these conflict resolution strategies achieves a resolution of a conflict via negotiation with a norm issuer. In [21], an analysis of different normative conflicts is provided. The authors suggest that a deontic inconsistency arises when an action is simultaneously permitted and prohibited. In [22], three forms of conflict/inconsistency are described as *total-total*, *total-partial* and *intersection*. These are special cases of the intersection of norms as described in figure 2 and in [16] – a permission entailing the prohibition, a prohibition entailing the permission or an overlap of both norms.

The *S*CIFF framework [23] is related to our work in that it also uses constraint resolution to reduce the scope of expectations to avoid conflict – expectation is a concept closely related to norms [24]. For instance, in that work, $\mathbf{E}(p, X), 0 \leq X \leq 10$ means that $p$ is expected to hold true between 0 and

10, and $\mathbf{EN}(p, Y), Y > 5$ means that p is expected not to hold true when $Y$ is greater than 5; positive expectations are related to obligations (and are implicitly existentially quantified) and negative expectations are related to prohibitions (and are implicitly universally quantified). The *S*CIFF proof procedure uses constraint resolution to reduce the domain of the expectations (and non-expectations). However, *S*CIFF always gives higher priority to negative expectations against positive ones.

## 9 Conclusions and Future Work

We have presented a novel mechanism to detect and resolve conflicts in norm-regulated environment. Such conflicts arise when an action is simultaneously obliged and prohibited or, alternatively, when an action is permitted and prohibited. We introduce norms as first-order atomic formulae to whose variables we can associate arbitrary constraints – this allows for more expressive norms, with a finer granularity and greater precision. The proposed mechanism is based on first-order unification and constraint satisfaction algorithms, extending our previous work [2], addressing a more expressive class of norms. Our conflict resolution mechanism amounts to manipulating the constraints of norms to avoid overlapping values of variables – this is called the "curtailment" of variables/norms. We have also introduced a robust and flexible algorithm to manage the adoption of possibly conflicting norms, whereby explicit policies depict how the curtailment between specific norms should take place. Our proposed formalism naturally allows the detection of indirect normative conflicts, arising when an action is broken down into composite actions appearing in conflicting norms.

In this paper we only considered universally quantified norms, leaving out important cases of existential quantifications. If existential quantification is allowed, then disjunction of constraints must be preserved. In this case, replacing a norm that has a disjunction of constraints with a conjunction of separate norms does not work anymore. If we allow existential quantification then we must preserve disjunctions of constraints and the set of norms $\Omega$ should be managed differently, in particular, disjunctions of norms should be allowed. We are currently working to address these issues.

The policies establishing which of two conflicting norms should be curtailed, confers generality on our approach, being neatly accommodated in our algorithms. We observe, however, that it would also be possible to make policies part of the virtual organisation (VO) specification, giving higher priority to those norms that allow the progress of the organisation. For instance, if $p(X)$ is forbidden and $p(Y)$ is permitted (both for the same group of agents/roles), that is, there is a complete overlap on the norms' scope of influence, then a policy on the VO could specify which of the two should be "removed" (by adding the constraint $X \neq Y$ onto it), based on which of them would allow the VO to progress. For example, if the VO progresses when an agent performs $p(a)$, then the prohibition could be lifted.

We want to extend our work to also address the removal of norms: when a norm is removed, all those curtailments it caused must be undone. We envisage a roll-back/roll-forward mechanism, whereby a history of normative states allows us to retrieve the state prior to the introduction of the norm to be removed (roll-back) and apply to this state all the updates which took place after the norm was

introduced, skipping the actual norm to be removed (roll-forward). Additionally, we want to integrate our mechanisms with norm-updating approaches such as [12] – we want to investigate if it is possible (and in which circumstances) to detect conflicts at the design stage of norm updates (as opposed to run-time).

**Acknowledgements**: This research is continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence (`http://www.usukita.org`).

## References

1. Fitting, M.: First-Order Logic and Automated Theorem Proving. Springer-Verlag, New York, U.S.A. (1990)
2. Vasconcelos, W., Kollingbaum, M., Norman, T., García-Camino, A.: Resolving Conflict and Inconsistency in Norm-Regulated Virtual Organizations. In: Proceedings of AAMAS 2007. (2007)
3. O'Leary, D.E., Kuokka, D., Plant, R.: Artificial Intelligence and Virtual Organizations. Commun. ACM **40**(1) (1997)
4. Apt, K.R.: From Logic Programming to Prolog. Prentice-Hall, U.K. (1997)
5. Parunak, H.V.D., Odell, J.: Representing Social Structures in UML. In: Procs 5th Int'l Conf. on Autonomous Agents, Montreal, Canada, ACM Press (2001) 100–101
6. Rodríguez-Aguilar, J.A.: On the Design and Construction of Agent-mediated Electronic Institutions. PhD thesis, IIIA-CSIC, Spain (2001)
7. Pacheco, O., Carmo, J.: A Role Based Model for the Normative Specification of Organized Collective Agency and Agents Interaction. Autonomous Agents and Multi-Agent Systems **6**(2) (2003) 145–184
8. Garcia-Camino, A., Rodriguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.W.: A Distributed Architecture for Norm-Aware Agent Societies. Volume 3904 of LNAI. Springer-Verlag (2005)
9. Vasconcelos, W.W.: Expressive Global Protocols via Logic-Based Electronic Institutions. In: Proc. 2nd Int'l Joint Conf. on Autonomous Agents & Multi-Agent Systems (AAMAS 2003), Melbourne, Australia, ACM, U.S.A (2003)
10. Pacheco, O., Carmo, J.: A Role Based Model for the Normative Specification of Organized Collective Agency and Agents Interaction. Autonomous Agents and Multi-Agent Systems **6**(2) (2003) 145–184
11. Jaffar, J., Maher, M.J.: Constraint Logic Programming: A Survey. Journal of Logic Progr. **19/20** (1994) 503–581
12. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: A Rule-based Approach to Norm-Oriented Programming of Electronic Institutions. ACM SIGecom Exchanges **5**(5) (2006) 33–40
13. Swedish Institute of Computer Science: SICStus Prolog. (2005) `http://www.sics.se/isl/sicstuswww/site/index.html`, viewed on 10 Feb 2005 at 18.16 GMT.
14. Jaffar, J., Maher, M.J., Marriott, K., Stuckey, P.J.: The Semantics of Constraint Logic Programs. Journal of Logic Programming **37**(1-3) (1998) 1–46
15. Holzbaur, C.: ÖFAI clp(q,r) Manual, Edition 1.3.3. TR-95-09, Austrian Research Institute for A. I., Vienna, Austria (1995)
16. Kollingbaum, M., Norman, T., Preece, A., Sleeman, D.: Norm Refinement: Informing the Re-negotiation of Contracts. In Boella, G., Boissier, O., Matson, E., Vazquez-Salceda, J., eds.: ECAI 2006 Workshop on Coordination, Organization, Institutions and Norms in Agent Systems, COIN@ECAI 2006. (2006) 46–51
17. García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A.: An Algorithm for Conflict Resolution in Regulated Compound Activities. In: Seventh Annual International Workshop Engineering Societies in the Agents World (ESAW'06). (2006)

18. Dignum, F.: Autonomous Agents with Norms. Artificial Intelligence and Law **7** (1999) 69–79
19. Sergot, M.: A Computational Theory of Normative Positions. ACM Transactions on Computational Logic **2**(4) (2001) 581–622
20. Leite, J.A., Alferes, J.J., Pereira, L.M.: Multi-Dimensional Dynamic Knowledge Representation. Volume 2173 of LNAI. Springer-Verlag (2001)
21. Elhag, A., Breuker, J., Brouwer, P.: On the Formal Analysis of Normative Conflicts. Information & Comms. Techn. Law **9**(3) (2000) 207–217
22. Ross, A.: On Law and Justice. Stevens & Sons (1958)
23. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: The SCIFF Abductive Proof Procedure. Volume 3673 of LNAI. Springer-Verlag (2005)
24. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Sartor, G., Torroni, P.: Mapping Deontic Operators to Abductive Expectations. Computational & Mathematical Organization **12**(2-3) (2006) 205–225

# Structured Argumentation in a Mediator for Online Dispute Resolution

Ioan Alfred Letia[1] and Adrian Groza[1]

Technical University of Cluj-Napoca
Department of Computer Science
Baritiu 28, RO-400391 Cluj-Napoca, Romania
{letia,adrian}@cs-gw.utcluj.ro

**Abstract.** Online dispute resolution is becoming the main method when dealing with a conflict in e-commerce. A family of defeasible reasoning patterns is used to provide a useful link between dispute resolution agents and legal doctrines. The proposed argumentation framework combines defeasible logic with temporal reasoning and argumentation with level of certainty. The evaluation of arguments depends on the stage of the dispute: commencement, discovery, pre-trial, arbitration, according to current practice in law. By applying the open world assumption to the rules, the argumentative semantics of defeasible logic is enriched with three types of negated rules which offer symmetrical means of argumentation for both disputants. A corollary of this extension consists in defining a specialized type of undercutting defeater. The theory is illustrated with the help of a concrete business-to-client case in a partially implemented system.

## 1 Introduction

Online Dispute Resolution (ODR) promises to become the predominant approach to settle e-commerce disputes. To reach this statute it needed ten years of fast and sustained development [1]: starting in 1996 as a hobby, an experimental stage sustained by academics and non-profit organizations during 1997-1998, an entrepreneurial stage from 1999 (75% rate of success as business), and beginning with 2003 there have been much governmental effort and many projects to institutionalize the online dispute resolution process.

Regarding the automation of the ODR process, one goal of this paper is to provide a flexible argumentation framework, according to the current practice in law, which can be effectively employed in online dispute resolution agents. In recent years several researchers acknowledged the value of argumentation theory for ODR [2]. Flexibility in configuring ODR systems is both an opportunity and a challenge. The opportunity is that any business can, quite quickly, have its own "court" specialized in disputes that might occur in its specific business domain. The challenge is that the technical instrumentation must simultaneously satisfy the business viewpoint asking for trust [3] and the legal viewpoint, which requires accordance with the current practice in law.

## 2 Argumentation Framework

We enrich the defeasible logic of Governatori [4] with interval-based temporal reasoning and its argumentation semantics with level of certainty and negated rules. Although defeasible logic has already been proved suitable for legal reasoning [5], by introducing interval-based reasoning we attempt to provide a more appropriate framework for practical scenarios, having the possibility to model contract deadlines. The levels of certainty for weighting arguments are meant to better handle incomplete information, vagueness, or fuzziness of the terms implied in the dispute.

**Definition.** *A theory in temporal defeasible logic (TDL) is a structure* $\langle F, R\rangle$ *formed by a finite set of facts* $f(\beta)[a,b] \in F$ *valid at time* $t$, $a \leq t \leq b$, *and a finite set of rules* $r(\gamma) \in R$, *with certainty factors* $\beta$, $\gamma \in (0..1]$. *A fact* $f(\beta) \in F$ *is strict if* $\beta = 1$ *and defeasible if* $\beta < 1$.

The rules are split in two disjoint sets: the set of support rules $R_{sup}$ which can be used to infer conclusions and the set of defeaters $R_{def}$ that can be used only to block the derivation of some conclusions.

**Definition.** *A rule* $r(\gamma) \in R_{sup}$ *is strict* ($\rightarrow$) *iff* $\gamma = 1$, *with the set of strict rules* $R_s = \{r(\gamma) \in R_{sup} | \gamma = 1\}$. *A rule* $r(\gamma) \in R_{sup}$ *is defeasible* ($\Rightarrow$) *iff* $\gamma < 1$, *with the set of defeasible rules* $R_d = \{r(\gamma) \in R_{sup} | \gamma < 1\}$.

Strict rules are rules in the classical sense, that is whenever the premises are indisputable, then so is the conclusion, while defeasible rules are rules that can be defeated by contrary evidence. Following Pollock's terminology [6], a defeasible conclusion $q$ can be defeated either by inferring the opposite one $\sim q$ with a superior certainty factor (rebuttal defeater), or by attacking ($\rightsquigarrow q$) the link between the premises and the conclusion $q$ (undercutting defeater[1]).

Facts within TDL are enhanced with validity intervals. For premise $a[x,y]$ and a conclusion $b[u,v]$ the following weak semantics is used: if $a$ is valid in at least one moment within $[x,y]$, then $b$ is valid in all moments from $[u,v]$. In this interpretation (imprecise premise, precise conclusion), the validity interval [a,b] of a rule depends on the activation intervals of its own premises: $r_i(\gamma)[a,b] : q_1(\beta_1)[a_1,b_1], ...q_k(\beta_k)[a_k,b_k] \Rightarrow q_0(\beta_0)[a_0,b_0]$, with $a = min(a_i)$ and $b = max(b_i), i \in [1..k]$. For the particular case when a defeasible rule has only one premise, its activation interval is synonym to the validity of that premise: $q_1[a_1,b_1] \Rightarrow q_0[a_0,b_0] \Leftrightarrow (q_1 \Rightarrow q_0[a_0,b_0])[a_1,b_1]$. This feature is used in nested rules[2].

[1] Intuitively, an undercutting defeater argues that the conclusion is not sufficiently supported by its premises.

[2] In our approach, rules are allowed to appear as premises or conclusions within other rules. The general case of such nested rule is represented by: $r_i(\gamma_i) : r_j[a_2,b_2] \Rightarrow r_k[a_3,b_3]$, where the existence of the rule $r_j$ fires the conclusion $r_k$, which can be seen as a dynamic rule. Another technical approach [7] consists in using an objectivation operator to translate a meta-level expression to an object-level expression.

Similar to facts, the rules acting as premises or conclusion within the body of a nested rule can appear negated. We use the following notations: $\nrightarrow$ for $\neg\,(a \rightarrow b)$, meaning that "$a$ does not strictly determine $b$", $\not\Rightarrow$ for $\neg\,(a \Rightarrow b)$, meaning that "$a$ does not defeasibly determine $b$", and $\not\rightsquigarrow$ for $\neg\,(a \rightsquigarrow b)$ meaning that "$a$ does not defeat $b$". We note by $R_{ns}$ the set of negated strict rules, by $R_{nd}$ the set of negated defeasible rules, and by $R_{ndef}$ the set of negated defeaters. The problem consists in giving a proper interpretation to a negated rule. Firstly, the negated rule represents a counterargument to the opposite rule, negated rules having the same role as an undercutting defeater, attacking the links between the premises and the conclusion. The difference consists in the fact that a defeater of the consequent $q$ attacks all rules which sustain $q$, whilst the negated rule attacks a single rule sustaining the respective conclusion[3]. The version of Toulmin's standard example

*Claim*: Harry is a British subject now.
*Datum*: Harry was born in Bermuda in 1937.
Harry is become an American citizen[1966,1966].
Very probably Harry speaks English.
*Warrant*: A man born in Bermuda will generally be a British subject.
English speakers are usually British subject.
*Backing*: Civil Code 123 provides that persons born in Bermuda
are generally British subjects.
*Exception*: An American Citizen cannot be a British subject.
*Counter-example*: Speaking English does not mean one is a British subject.

$Harry\_Born\_Bermuda(1.0)[1937, 1937]$
$Harry\_American\_Citizen(1.0)[1960, 1960]$
$Harry\_Speaks\_English(0.95)[1937, now]$
$r_1 : (0.9) : Born\_Bermuda[t, t] \Rightarrow British\_Subject[t, now]$
$r_2 : (0.5) : Speak\_English[1, 1] \Rightarrow [t, t]British\_Subject[t, t]$
$r_3 : (0.9)Harry\_American\_Citizen[1, 1] \rightsquigarrow British\_Subject[2, 2])$.
$r_4 : (0.9)Speak\_English[1, 1] \not\Rightarrow British\_Subject[t, t])$.
$r_5 : valid\_code\_123[0, t] \rightarrow (Born\_Bermuda[t, t] \Rightarrow British\_Subject[t, now])$.

**Fig. 1.** A special type of undercutting defeater: negated rules.

about British citizenship in figure 1 illustrates this difference. Here, the rule $r_4$ attacks only the rule $r_2$, which is defeated. Opposite, the undercutting defeater $r_3$ attacks both $r_1$ and $r_2$ with a stronger certainty factor, blocking the claim $+\partial British\_Subject : now$. We use Pollock's undercutting defeaters to model exceptions and negated rules in representing counter-examples. Undercutting defeaters or negated rules cannot be used to draw a conclusion, their only use is to prevent some conclusions. Practically, introducing negated rules, we extend the open world assumption to the rules. A relation between two terms $a$ and

[3] If defeaters represent rules used to block the derivation of some conclusion $q$, the negated rules are used to block the activation of a specific support argument for $q$.

| $\varphi$ | $\sim \varphi$ |
|---|---|
| $q$ | $\neg q,\ X \rightarrow \neg q,\ X \Rightarrow \neg q$ |
| $A \rightarrow q$ | $\neg q,\ X \rightarrow \neg q,\ A \nrightarrow q$ |
| $A \Rightarrow q$ | $\neg q,\ X \rightarrow \neg q,\ X \Rightarrow \neg q,\ X \rightsquigarrow \neg q,\ A \not\Rightarrow q$ |
| $A \rightsquigarrow q$ | $A \not\rightsquigarrow q$ |
| $A \nrightarrow q$ | $A \rightarrow q$ |
| $A \not\Rightarrow q$ | $A \Rightarrow q$ |
| $A \not\rightsquigarrow q$ | $A \rightsquigarrow q$ |

**Fig. 2.** Attacking a sentence $\varphi$ depends on its type.

$b$ can be positive ($a \rightarrow b$), negative ($a \nrightarrow b$), or unspecified. Pairs of relations provide symmetrical means of argumentation for both disputants. The type of counterargument depends on the type of the current sentence $\varphi$: fact, support rule, defeater (figure 2). Here, one can see that the support rules ($\rightarrow$, $\Rightarrow$) can be attacked in different ways. The negated rule $A \not\rightsquigarrow q$ represents an argument in favor of $q$, because it attacks the undercutting defeater $A \rightsquigarrow q$. The second utility of the negated rules is the dynamic elimination of some arguments from the knowledge base. The existence of a negated rule allows the deactivation of a rule, when the certainty factor is strong enough.

## 3 Types of Agents for ODR

A family of defeasible reasoning patterns is discussed next, employed in dispute resolution agents for more flexibility of the decision. The strategy of an agent consists of three orthogonal components which modularly capture different concerns of the problem: basic component, tactical component, and attitude component.

### 3.1 Basic Component

**Fuzzy Inference.** Using the weakest link principle for deductive arguments [6], the conclusion $q_0$ is as good as the weakest premise, given by $min(\beta_1, ..., \beta_k)$. Additionally, the certainty factor is also influenced by the strength $\gamma$ of the inferencing rule (figure 3). The figure presents the generalized modus ponens where given the premises $q_i(\beta_i)[t_i]$ valid at time $t_i$ required by the rule $r_i$, the conclusion $q_0$ is inferred with a strength equal to the minimum between the strength of the premises $\beta_i$ and the strength of the rule $r_i$.

**Probabilistic inference.** A probabilistic approach of computing the certainty factor of a conclusion would multiply the certainty factors of all premises. Practically, the certainty factor depends on the number of premises. In this probabilistic context, the temporal persistence issue can also be considered. Suppose the fact $a$ having the certainty factor $\beta_a$ is valid at time $t$. The following interpretation could arise: if $a$ at $t$ then there is a defeasible reason to infer $a$ at $t + \Delta t$, the

$$rule\_r_i : \_q_0[a_0, b_0] \xleftarrow{\gamma} q_1[a_1, b_1] \wedge ..... \wedge q_k[a_k, b_k]$$
$$facts : \_q_1(\beta_1)[t_1], a_1 \leq t_1 \leq b_1 ....., q_k(\beta_k)[t_k], a_k \leq t_k \leq b_k$$
$$\overline{q_0(min(\beta_1, ....., \beta_k, \gamma)[a_i], \forall a_i, a_0 \leq a_i \leq b_0}$$

**Fig. 3.** Inferring the conclusion $q_0$ when no valid defeaters exist.

certainty factor for $a$ being a monotonic decreasing function of argument $\Delta t$. A typical scenario might be: the probability that the new business partner will breach the contract is 0.2. This probability decreases as time passes and the contract meets its time of maturity. Similarly, an agent believes that his business partner is trust-able with a factor of 0.6. If nothing defeats this believe in time, the agent increases the trust in the partnership as the business runs. By default we consider that the certainty factor is constant in time and we provide mechanisms to adjust it for each scenario.

### 3.2 Tactical Component

The same conclusion $q$ can be sustained by several arguments with different degrees of reliance. The tactical component defines how an agent handles the accrual of such valid arguments. Let $n$ be the number of valid derivations of the consequent $q$ and $cf[q_i]$ the certainty factor of the inference number $i$ of $q$, $i \in [1..n]$. Similarly, $m$ is the number of valid undercutting defeaters (both defeaters and negated rules) of the sentence $q$ and we note by $cf[\sim q_j]$ the certainty factor of the $j$ defeater of $q$, $j \in [1..m]$. If $p$ is the number of valid rebuttal defeaters, we note with $cf[\neg q_k]$ the certainty factor of the $k$ rebuttal defeater for $q$, $k \in [1..p]$.

**Persuasion Agent.** In some situations, independent reasons supporting the same action provide stronger arguments in favor of that conclusion. For instance, the testimony of two witnesses is required in judicial cases. This approach is appropriate for *practical reasoning*, when the decision is about what actions to perform [6] or evidential reasoning [8]. One issue related to this agent regards the difficulty to identify independent reasons. Thus, an argument presented in different forms contributes with all its avatars to the certainty factor. Similarly, an argument subsumed by another general argument also contributes to the certainty factor. Correlated to the same judicial example, if the two witnesses are kin or they conferred with each other, only one testimony is accepted in the trial. The accrual of dependent arguments is not necessarily useless. Changing the perspective, this case can be valuable in persuasion dialogs, where an agent, by repeatedly posting the same argument in different representations, will end in convincing his partner to accept that sentence.

A persuasion agent computes the certainty factor of the thesis $q$ under dispute as follows. Firstly, it considers all the accepted arguments supporting the claim

$q$ at time $t$. This amount is decreasing by all his objections about deriving $q$, in our case all the undercutting defeaters. If the remaining certainty factor is still greater than all the existing support for the opposite conclusion $\neg q$, the thesis is successfully established. Formally, the model of persuasion based on the defeasible pattern of inference becomes:

$$cf[q] = \begin{cases} min(1, \sum_{i=1}^{n} cf[q_i] - \sum_{j=1}^{m} cf[\sim q_j]), & \sum_{i=1}^{n} cf[q_i] - \sum_{j=1}^{m} cf[\sim q_j] > \sum_{k=1}^{p} cf[\neg q_k] \\ 0, & \text{otherwise} \end{cases}$$

**Epistemic Agent.** In reasoning about what to believe or *epistemic reasoning* the accrual of arguments does not hold [6]. The sentence $q$ is inferred if it has a greater support than any of the undercutter or rebuttal defeaters, but the certainty factor is not diminished:

$$cf[q] = \begin{cases} max(cf[q_i]), & max(cf[q_i]) > max(cf[\neg q_k], cf[\sim q_j]) \\ 0, & \text{otherwise} \end{cases}$$

The choice between a persuasion or an epistemic agent depends on the context. A hybrid agent would include modalities such as *action* or *knowledge* for capturing practical and, respectively, epistemic reasoning, with the certainty factor of the conclusion computed accordingly.

**Rigorous Agent.** A rigorous agent will treat differently each type of defeater. Thus, only the strongest undercutting defeater contributes to the decreasing of the certainty factor. If the remaining strength of the conclusion overwhelms the most powerful rebuttal defeater, the respective conclusion is derived.

$$cf[q] = \begin{cases} max(cf[q_i]) - max(cf[\sim q_j], & max(cf[q_i]) - max(cf[\sim q_j]) \\ & > max(cf[\neg q_k]) \\ 0, & \text{otherwise} \end{cases}$$

Next we present the derivation formula of a consequent according to the reasoning strategy of the rigorous agent. A conclusion in TDL is a tagged literal which can have the following forms: i) $+\Delta q : t \Leftrightarrow q$ is definitely provable at time $t$ in $TDL$, using only strict facts and rules (figure 4); ii) $-\Delta q : t \Leftrightarrow q$ is not definitely provable at time $t$ in $TDL$; iii) $+\partial q : t \Leftrightarrow q$ is defeasibly provable at time $t$ in $TDL$ (figure5); iv) $-\partial q : t \Leftrightarrow q$ is not defeasibly provable at time $t$ in $TDL$.

A conclusion $q$ is strictly provable at time $t$ (figure 4) if (1) $q$ is a strict fact valid at time $t$ or (2) there exists a strict rule with conclusion $q[u, v]$ and the instant of time $t$ within $[u, v]$, which rule, (2.1) for all its antecedents $a[x_1, y_1]$, there is a time $t'$ when they are strictly valid and (2.2) there is no strict negated rule $ns$, attacking rule $r$.

Defeasible derivations have an argumentation like structure [4]: firstly, we choose a supported rule having the conclusions $q$ we want to prove, secondly we consider all the possible counterarguments against $q$, and finally we rebut all

$+\Delta$:
If $P(i+1) = +\Delta q : t$ then
(1) $\exists q(\beta)[u, v] \in F$ and $\beta = 1$ and $u \leq t \leq v$ or
(2) $\exists r \in R_s[q[u, v]]$ with $u \leq t \leq v$ such as
(2.1)$\forall a[x_1, y_1] \in A(r) \exists t' : +\Delta a : t' \in P(1..i)$ and $x_1 \leq t' \leq y_1$
(2.2) $\nexists ns \in R_{ns}[r]$

**Fig. 4.** Definite proof for the consequent $q$ at time $t$ for the rigorous agent.

the above counterarguments showing that, either some of their premises do not hold, or the rule used for its derivation is weaker than the rule supporting the initial conclusion $q$. The sentence $q$ is defeasibly provable at time $t$ (figure 5) if

$+\partial$:
If $P(i+1) = +\partial q : t$ then
(1) $+\Delta q : t \in P(1..i)$ or
(2) q is supported
(2.1) $\exists q(\beta)[u, v] \in F$ and $\beta < 1$ and $t \in [u, v]$ or
(2.2) $\exists r(\gamma_r) \in R_{sup}[q[u, v]]$, $\forall a[x_1, y_1] \in A(r) \exists t'$ such as $+\partial a : t' \in P(1..i)$ and $t' \in [x_1, y_1]$)
and not defeated
(2.3) $\forall nd(\gamma_{nd}) \in R_{nd}[r] \cup R_{ns}[r]], \gamma_r > \gamma_{nd}$ and
(2.4) $\forall def(\gamma_{def}) \in R_{def}[q[u_1, v_1]]$or
(2.4.1) $t \notin [u_1, v_1]$ or
(2.4.2) $\exists a[x_1, y_1] \in A(def)$ $\forall t' \in [x_1, y_1] - \partial a : t'$ or
(2.4.3) $\exists ndef(\gamma_{ndef}) in R_{ndef}[def]$, $\gamma_{ndef} > \gamma_{def}$ and
(2.5) $\forall d(\gamma_d) \in R_{sup}[\sim q[u_2, v_2]]$ with
$\forall a[x_2, y_2] \in A(d)$, $\exists t' \in [x_2, y_2] + \partial a : t'$, $t \in [u_2, v_2]$ either
(2.5.1) $\exists nnd(\gamma_{nnd}) \in R_{nd}[d] \cup R_{ns}[d], \gamma_{nnd} > \gamma d$, or
(2.5.2) $\gamma_r - \gamma_{def} > \gamma_d$

**Fig. 5.** Defeasible derivation of consequence $q$ at time $t$ for the rigorous agent.

(1) it is strictly provable at $t$, or (2) there is a valid support for $q$ either (2.1) it is a defeasible fact valid at $t$, or (2.2) there exists a rule with all premises valid sustaining that conclusion $q$ and it is not defeated by (2.3) a negated rule with a stronger certainty factor, or (2.4) by an undercutting defeater $def$ where (2.4.1) time $t$ is not within the validity interval of the defeater, or (2.4.2) the defeater has an antecedent $a$ which cannot be derived, or (2.4.3) there exists a negated defeater stronger than $def$, and (2.5) for all valid rebuttal defeaters $d$ either (2.5.1) there is a negated rule which defeats $d$ or (2.5.2) the support for conclusion $q$ after it is attacked by the undercutter defeaters remains stronger than all the valid rebuttal defeaters. The strict order relation in (2.3), (2.4.3), and (2.5.2) provides a skeptical reasoning mechanism, meaning that none of $q : t$ and $\sim q : t$

is derived when they have equal support. Allowing the ambiguity propagation increases the number of inferred conclusions, useful in the argumentation process of ODR systems oriented towards solution rather than finding the degree of guilt.

### 3.3 Attitude Component

The attitude component defines the argumentative attitude of an agent towards other participants, making a distinction between the agent's private collection of arguments and its public uttered sentences. We adapt the claim-attitude and concede-attitude [9], defining the level of proof sufficient to convince the opponent that a given sentence is true, to our defeasible formalism.

The following standards of proofs from current legal practice are modeled: *scintilla of evidence*, *reasonable suspicion*[4], *preponderance of evidence*[5], *clear and convincing evidence*, and *beyond reasonable doubt*[6].

**Definition.** *Claim-attitude at time t*

- *A confident agent can claim any sentence $q : t$ for which there is a valid support rule $r \in R_{sup}$ (scintilla of evidence).*
- *A careful agent can claim any proposition $q : t$ if there is no valid rebuttal defeater sustaining the opposite sentence $\neg q : t$ (reasonable suspicion).*
- *A precaution agent can claim any proposition $q : t$ if there is no valid rebuttal or undercutting defeater for the opposite sentence $\neg q : t$ (preponderance of evidence).*
- *A thoughtful agent can claim any proposition $q : t$ for which it can construct a defeasible proof $+\partial q : t$ (clear and convincing evidence).*
- *A strict agent can claim any proposition $q : t$ for which it can construct a definite proof $+\Delta q : t$ according to its theory (beyond reasonable doubt).*

**Definition.** *Concede-attitude at time t*

- *A credulous agent can concede to any sentence $q : t$ for which it has a valid support rule $r \in R_{sup}$ (scintilla of evidence).*
- *A cautious agent can concede to any proposition $q : t$ if it is not able to provide a stronger rebuttal defeater for the opposite sentence $\neg q : t$ (reasonable suspicion).*
- *A vigilant agent can concede to any proposition $q : t$ if it is not able to provide a stronger rebuttal or undercutting valid defeater (preponderance of evidence).*

[4] Reasonable suspicion is a low standard of proof used to determine whether a brief investigative stop or a brief search by a police officer is warranted.

[5] Also known as the "balance of probabilities", this standard is met if the proposition is more likely to be true than not true

[6] This means that the proposition must be proved to the extent that there is no "reasonable doubt" in the mind of a reasonable person, such as 90% certain in the US.

- *A skeptical agent can concede only to those propositions $q : t$ for which it can construct a defeasible proof $+\partial q : t$ (clear and convincing evidence).*
- *A wary agent can concede to any proposition $q : t$ for which it can construct a definite proof $+\Delta q : t$ according to its theory (beyond reasonable doubt).*

During the argumentation process, a confident agent might claim any proposition for which it is able to construct an argument (propositions which are not credible can also be uttered). When, for example, the knowledge base of the agent consists of the rules $r_1 : (0.5) : a[1,1] \Rightarrow q[2,2]$, and $r_2 : b[1,1] \rightarrow \neg q[2,2]$ where $a$ and $b$ are strict valid facts, then it is still presumable for the agent to claim $q$, even if it is aware of the existence of the stronger counterargument $r_2$ sustaining the opposite consequent. A careful agent does not communicate a sentence if it is conscious about the validity of a rebuttal defeater, no matter what certainty factor that argument has. Similarly, a precaution agent additionally considers the validity of an undercutter defeater in order to minimize the risk of a potential counterattack from the other disputants. A more critical attitude is the thoughtful one, where an agent will claim propositions for which it is able to construct an acceptable argument, an argument which is defeasibly provable from its knowledge base. A strict agent does not take any risk to be combated in its claims, therefore it conveys only sentences supported by strict inference according to its defeasible theory. The concede-attitudes are used similarly to the claim-attitudes.

## 4 Choosing the Proper Strategy

Various situations might be encountered. (i) The market may have substantial authority, and the same mediation strategy is imposed to all disputants. (ii) Consistent with party autonomy, the agents may settle on different mediation strategies at contracting time or just prior to the arbitration. This approach increases the flexibility and efficiency, because the agents are the ones who know what type of mediation strategy better protects their interests[7]. (iii) All the above mediator's strategies might be used during the resolution process[8].

In markets where the consumer protection is the main concern, the mediator may provide different interfaces to the disputants. For instance, the persuasion strategy might guarantee high level of protection to the client being irritated by several issues. The strategies may also be correlated to the current dispute: persuasion strategy is connected to cases involving fairness or good faith. Similarly, the persuasion strategy is adequate in the first stage of the dispute, the so called evidential phase, when the factual information is collected.

[7] Mediators and arbitrators are humans who might have biases and prejudices. Frequently, the disputants have the opportunity to select the arbitrator who is likely to be sensitive to their predicament.

[8] Most of the human mediators use a form of the co-mediation model. Having two mediators can be an effective way to deal with many different ODR challenges, fitting well to legal systems based on jury.

On the one hand, a probabilistic approach is a good candidate when the dispute process is in its early stages, when there is little information available, and the mediator tries to figure out if the initial claim is warranted[9]. It also may be considered when the information sources are not trust-able. On the other hand, when the process reaches its maturity stage, the irrelevant facts become clear. Therefore, within a fuzzy inference, the unimportant facts do not influence the decision. Legal rules are often open to several interpretations because some terms within legal texts are vague. It is the mediator who gives the appropriate interpretation to terms such as *reasonable* or *sufficient*. The agent strategy depends on the active legal doctrines within the market. If the required standard of proof is *preponderance of evidence*, the probabilistic approach fits better, but when *beyond a reasonable doubt* doctrine is active, the fuzzy reasoning is appropriate.

The attitude component is relevant in the context of revealing information. Sometimes, the arguments uttered, either fact or rule, represent private information. The agents must assign a utility cost to revealing information, as well as a utility to winning an argument. The strategy depends on the rules of dialog game where the agent participates. When the dialog protocol stipulates that a claim which has been defeated by a party cannot be uttered again, then a strict or thoughtful attitude must be considered. Opposite, a confident attitude is adequate when a party wants to find information, because his opponent defeats the claim by revealing his private arguments.

The relevant question concerns the validity of the semantic model. This question requires empirical evaluations with realistic test cases[10] in order to choose the best suited defeasible pattern within a particular market. The common disputes are translated into defeasible theories[11], and the agent's decision is compared with the one given by the human mediator. The highest scored strategy is provided to the disputant who might better anticipate the verdict and the execution timing. The advantage here consists in the fact that judicial cases that are not conforming to a pattern useful in deriving rules, are not treated as noise and removed. Simply, they are considered exceptions and encapsulated as defeaters or strong defeasible rules.

## 5 Dispute Resolution Phases

The client orders a hardware object through a shop-on-line web site (scenario adapted from [11]). The seller has published general contractual conditions on the web site. One of the clauses stipulates that if the product sent is defective, the client has the right to get it repaired or replaced, depending on the seller's choice. After an order is made at $t_0$, the seller sends the item. When the client

[9] The *probable cause* doctrine may be invoked which requires a fair probability that a breach took place. Courts vary when determining what constitutes a "fair probability," some say 30%, others 40%, others 51%.

[10] See http://www.as.uky.edu/polisci/ulmerproject/index.html for a collection of such a legal dataset.

[11] ILP techniques are available for deriving defeasible theories from legal datasets [10].

receives it at $t_7$, he notices both that it does not work and its design was not quite similar to the picture on the web site. The seller accepts that the hardware might be defective, but invokes the mentioned clause. His default choice is to repair the item, but he also proposes to replace the product if the client accepts to pay the transport fee. The client replies that he will only pay half the fee. The client asks an ODR system for arbitration, submitting his argumentation. The seller asks the product to be replaced. The ODR system accepts to lead the arbitration and notifies the seller. The seller accepts and submits his own argumentation.

### 5.1 Commencement of Dispute

A dispute action is commenced by filling a complaint. If minimum of evidence is provided[12], the mediator takes into consideration the plaintiff's claim. Consequently, a judicial summon is addressed to the defendant. The probabilistic rigorous mediator with a credulous concede attitude is appropriate for this stage. The plaintiff believes with a certainty factor of 0.9 that the picture illustrating the item was irrelevant ($f_2$ in figure 6). Considering rule $r_5$ such a mediator will prove the $+\partial replace : 7$ conclusion with a certainty factor of $0.9 * 0.95 = 0.855$. Because this value is greater than the threshold of 0.2, the complaint is accepted and a dispute starts.

### 5.2 Discovery

The discovery is the pre-trial phase in a lawsuit in which each disputant can request evidence from the other party. Under the *duty of disclose* doctrine, the disputants have the obligation to share their own supporting evidence without being requested to by the other party. Failure to do so can preclude that evidence from being used in trial[13]. Modern dispute resolution strategies try to set the dispute in its early stages. Thus, the discovery phase is meant to clarify what the lawsuit is about, and perhaps to make a party realize it should settle or drop the claim, all before wasting court resources[14]. Because this early phase is mainly about evidence, a probabilistic epistemic mediator is recommended. Also, confident or careful claim attitudes prevail in obtaining information. During this dialog, the following facts become known: the item might be defective (defeasible fact $f_1$ has a certainty factor of 0.9), and the seller option is to repair the item ($f_3$). He advocates this through the contractual clauses $r_1$, $r_2$, $r_3$, and $r_4$, accepted by the buyer when the contract has been signed. The seller proposes to repair the product if the client accepts to pay the transport fee ($r_6$). The client might agree to pay half the fee ($r_7$) in order to derive the *seller_choice_replace* consequent, which is defeated by the seller response $r_8$.

[12] The claim is supported with 20% certainty factor.

[13] This applies only to evidence that supports their own case, not anything that could harm their case.

[14] A procedural rule stipulates that parties have the right to query 25 questions to each other in order to reveal information.

### 5.3 Pre-trial

The pre-trial represents the last gate-keeping function before trial, answering the question of whether the claim could even go to the arbitration phase. In this stage, the movant can affirmatively negate the claim, whilst the plaintiff may provide different arguments to support the claim. Therefore, a probabilistic persuasion mediator is appropriate in this stage. Because the negation of claims is modeled by rebuttal defeaters, the vigilant concede attitude functioning under the *reasonable suspicion* doctrine is recommended. The rebuttal defeater $r_9$ is conveyed by the defendant who argues that usually he does not replace items to non-premium customers. The probabilistic persuasion mediator will derive the *replace* conclusion with a certainty factor of 0.985.

### 5.4 Arbitration

$f_1 : defective\_item(0.9)[t_7, t_7]$.
$f_2 : irrelevant\_picture(0.9)[t_7, t_7]$.
$f_3 : seller\_choice\_repair(0.8)[t_0, t_7]$.
$f_4 : \neg premium\_customer(1.0)[t_0, t_7]$.
$r_1 : (0.5) defective\_item[t_0, t_7] \Rightarrow repair[t_0, t_7]$
$r_2 : (0.5) defective\_item[t_0, t_7] \Rightarrow replace[t_0, t_7]$
$r_3 : (0.6) seller\_choice\_replace[t_0, t_7] \rightsquigarrow repair[t_0, t_7]$
$r_4 : (0.6) seller\_choice\_repair[t_0, t_7] \rightsquigarrow replace[t_0, t_7]$
$r_5 : (0.95) irrelevant\_picture[t_0, t_7] \Rightarrow replace[t_0, t_7]$
$r_6 : transport\_fee[t_0, t_7] \rightarrow seller\_choice\_replace[t_0, t_7]$
$r_7 : (0.9) transport\_fee[t_0, t_7] \Rightarrow seller\_choice\_replace[t_0, t_7]$
$r_8 : transport\_fee[t_0, t_7] \nrightarrow seller\_choice\_replace[t_0, t_7]$
$r_9 : (0.7) \neg premium\_customer[t_0, t_7] \Rightarrow \neg replace[t_0, t_7]$
$r_{10} : (0.9) offer[t_0, t_0], acceptance[t_0, t_0], consideration[t_0, t_0] \Rightarrow contract\_valid[t_0, t_7]$
$r_{11} : (0.8) contract\_valid[t_0, t_0] \Rightarrow (irrelevant\_picture[t_0, t_7] \Rightarrow replace[t_0, t_7])$

**Fig. 6.** Sample of arguments collected during the run of a dispute

This phase is the presentation of the evidence gathered during earlier stages (figure 6). In the next step, the mediator decides to which jurisdiction the case belongs and loads the corresponding legal doctrines encapsulated as defeasible theories. He uses both the hard law (enactments, i.e. $r_{10}$) and the soft law (usages, customs within the e-market, i.e $r_{11}$) to activate the rules or to adjust the certainty factor of the disputants' arguments. As nested rules are allowed in our framework the activation can be done dynamically (rule $r_{11}$). Consider that *offer*, *acceptance*, and *consideration* accepted as strict facts, the contract is validated with a certainty factor of 0.9[15]. Thus, the dynamic rule

[15] In order to accommodate some exceptions like "the signer is under 18".

is activated with a certainty factor of $min(0.9, 0.8) = 0.8$, resulting the rule $r'_5(0.8) : irrelevant\_picture \Rightarrow replace$, which takes the place of the rule $r_5$ in figure 6. This mechanism provides the mediator the ability to dynamically adjust priorities among rules[16].

We recommend a fuzzy rigorous mediator with a skeptical concede attitude in order to compute the expected outcome. In the probabilistic approach the claim *replace* is sustained by the rule $r_2$ with 0.5 and by the rule $r'_5$ with $0.72 * 0.9 = 0.64$. In the fuzzy approach $r'_5$ supports the consequent with $min(0.8, 0.9)$. According to the fuzzy rigorous mediator $max(0.5, 0.8) - max(0, 6) < max(0.7)$, therefore the conclusion *replace* is not derived. Users can also explore hypothetical situations when mediators have different strategies: fuzzy, probabilistic, persuasion or epistemic. How the dispute outcome depends on each defeasible mediator type is shown in figure 7. Here, in case the ambiguity propagation is enabled, the fuzzy persuasion agent proves the consequent. Users are also able to have dialogs with the system to explore what would happen if some of their claims were rejected or just partially accepted[17].

| $+\partial replace : t_7$ | Persuasion | Epistemic | Thorough |
|---|---|---|---|
| Probabilistic | $No_{0.5+0.64-0.6<0.7}$ | $No_{max(0.5,0.64)<max(0.6,0.7)}$ | $No_{max(0.5,0.64)-0.6<0.7}$ |
| Fuzzy | $No/Yes_{0.5+0.8-0.6=0.7}$ | $Yes_{max(0,5,0.8)>max(0.6,0,7)}$ | $No_{max(0.5,0.8)-0.6<0.7}$ |

**Fig. 7.** Answer for query $+\partial replace : t_7$ depends on mediator type.

### 5.5 Post-trial

After the arbitration is done two paths might follow: the enforcement of judgment and appealing the results of the arbitration process. Appealing after the trial may be quite difficult. To facilitate trust in e-commerce, many governments have enacted a norm similar to: "Any item achieved in online transaction can be returned within 15 days, without reason." Under these circumstances, the seller concedes to replace the defective item within 3 days if the client requests this: $r_{20} : request[t_8, t_8] \rightarrow must\_replace\_item[t_8, t_{10}]$. If the client is satisfied this obligation is no longer active: $r_{21} : satisfied[t_8, t_10] \rightarrow \neg must\_change\_item[8, 10]$. The last role of the system is to monitor contract enactment. This is done simply by trying to prove $+\partial must\_change\_item : 10$. If the client has asked for replacement and within 3 days he did not get satisfaction, the obligation still stands.

[16] Under most laws, the arbitrator can assign as much probatory force as he believes they deserve, as long as this assessment is not arbitrary [11]. In the long run of ODR it is necessary to create specialized jurisdiction for e-commerce cases, where the certainty factor would be fine tuned according to precedents and mediator experience.

[17] For the scenario in figure 6 if the certainty factor of *irrelevant_picture* fact is greater than 0.8 a persuasion mediator will infer the *replace* conclusion.

## 6 Related Work

The need for computerized mechanisms for decision support comes from well known limits of human knowledge processing. One aim is to provide disputants information about the expected outcome of the resolution process[18]. The other goal is to enrich the mediator's ability to process knowledge and weight arguments. By enhancing the expertise level of the mediator we argue that such decision support system can be looked at as a fourth party as defined in [13].

In the DiaLaw system [14], if the respondent of an argument accepts all the premises of a valid argument, he must also accept the conclusion, in case the respective inductive rule was previously accepted. In our framework, in the light of new information, an undercutting defeater might be used to attack the link between the premises and the consequent. In our view, the existence of a finite set of pre-agreed rules is not feasible for practical applications[19]. Thus, both facts and rules may be dynamically asserted and retracted within the defeasible framework.

In the Carneades argumentation framework [15] three kinds or premises are used: ordinary premises, presumptions, and exceptions, where presumptions are used to model uncertain knowledge. In our approach, the nondeterminacy inherent in the early stages is handled by probabilistic reasoning, whilst defeaters deal with exceptions and information obtained during the resolution process. The framework also deals with information about dialectical status of a sentence: undisputed, at issue, accepted, rejected. We treated this issue elsewhere [16], by defining defeasible commitment machines as a flexible mechanism to adapt the life-cycle of the conveyed facts.

An ODR system was modeled within a multi-agent context by identifying and representing the types of communication between the different actors: parties, mediator, arbitrator, experts, witnesses, ODR-administrator, system-administrator, visitors [11]. Our approach does not regard the architectural requirements of an ODR system, but rather the reasoning capabilities of the arbitrator.

Rule-based systems are suitable for modeling the logical structure of legislation and are practically successful when the gap between factual and legal language is small and the regulations are uncontroversial, but they fail to model legal argument. Defeasible logic, through its argumentative semantics, overcomes this drawback. It is also adequate in practical applications due to its low complexity [17]. As a simulation tool, the ODR system designer may obtain results

[18] In the negotiation literature this is called BATNA: Know your best alternative to a negotiated agreement [12]

[19] By accepting a jurisdiction parties practically agree on a set of legal rules. Through a signed contract, parties agree not only on some contractual clauses, but also regarding several specific doctrines under which that contract is enacted (such as expectation damages, opportunity costs, reliance damages). Due to the open character of both legal rules and contractual clauses, there are situations when supplementary rules have to be considered.

regarding what types of strategies better suit the e-market or how information sharing can be used to settle the dispute.

The formalization of virtual organizations and contracts based on commitments [18] opens another path for ODR by enabling to capture the social structure. Changes of organizations impose some treatment for the dynamics of enacted contracts.

## 7 Conclusions

There is a strong motivation for the need of ODR systems to reflect different types of argumentation patterns, mainly those models where persuasion can be functionally embedded into negotiation protocols [2]. From the knowledge representation viewpoint the implemented system accommodates temporal defeasible reasoning, nested rules, and a dynamic notion of priority over the rules[20]. From the argumentative semantics viewpoint the system introduces negated rules to model counter-exampls, whilst Pollock's style undercutting defeaters are used to represents exceptions.

We advocate two strong points of this approach: (i) the *flexibility* of the framework due to the different patterns of weighting arguments and to the property of defeasible logic to model exceptions; (ii) the *accordance to legal practice*, by establishing a connection between these patterns and disputes phases as they appear in current practice in law. This view on the ODR issue does not insist on the temporal aspects included in the logic. They can be subject to further investigation regarding the application of the framework to contract enactment [18]. Our future work regards also the enrichment of the logical framework with explanation capabilities of the outcome, as a need for the trustworthiness and practical usability in a dispute resolution system.

### Acknowledgments

We are grateful to the anonymous reviewers for useful comments. Part of this work was supported by the grant 27702-990 from the National Research Council of the Romanian Ministry for Education and Research.

## References

1. Tyler, M.C., Bretherton, D.: Seventy-six and counting: An analysis of ODR sites. In: Workshop on Online Dispute Resolution at the International Conference on Artificial Intelligence and Law, Edinburgh, UK (2003) 13–28
2. Walton, D., Godden, D.: Persuasion dialogues in online dispute resolution. Artificial Intelligence and Law **13** (2006) 273–295
3. Rule, C., Friedberg, L.: The appropriate role of dispute resolution in building trust online. Artificial Intelligence and Law **13** (2006) 193–205

[20] A prototype based on LISA (Lisp-based Intelligent Software Agents) is available at http://cs-gw.utcluj.ro/∼adrian/tdl.html.

4. Governatori, G.: Representing business contracts in RuleML. Journal of Cooperative Information Systems **14** (2005)
5. Hage, J.: Law and defeasibility. Artificial Intelligence and Law **11** (2003) 221–242
6. Pollock, J.L.: Defeasible reasoning with variable degrees of justification. Artificial Intelligence **133** (2001) 233–282
7. Pollock, J.L.: How to reason defeasibly. Artificial Intelligence **57** (1992) 1–42
8. Prakken, H.: A study of accrual of arguments, with applications to evidential reasoning. In: 10th International Conference on Artificial Intelligence and Law, New York, NY, USA (2005) 85–94
9. Parsons, S., Wooldridge, M., Amgoud, L.: Properties and complexity of some formal inter-agent dialogues. Journal of Logic and Computation **13** (2003) 347–376
10. Johnston, B., Governatori, G.: An algorithm for the induction of defeasible logic theories from databases. In: Australasian Database Conference. (2003) 75–83
11. Bonnet, V., Boudaoud, K., Gagnebin, M., Harms, J., Schultz, T.: Online dispute resolution systems as web services. ICFAI Journal of Alternative Dispute **3** (2004) 57–74
12. Bellucci, E., Lodder, A.R., Zeleznikow, J.: Integrating artificial intelligence, argumentation and game theory to develop an online dispute resolution environment. In: 16th International Conference on Tools with Artificial Intelligence, IEEE Computer Society (2004) 749–754
13. Katsh, E., Rifkin, J.: Online Dispute Resolution: Resolving Conflicts in Cyberspace. John Wiley (2001)
14. Lodder, A.: DiaLaw: On Legal Justification and Dialogical Models of Argumentation. Kluwer, Dordrecht (1999)
15. Gordon, T., Walton, D.: The Carneades argumentation framework: Using presumptions and exceptions to model critical questions. In: 1st International Conference on Computational Models of Argument, Amsterdam, IOS Press (2006) 208–219
16. Letia, I.A., Groza, A.: Running contracts with defeasible commitment. In Moonis, A., Dapoigny, R., eds.: Advances in Applied Artificial Intelligence. LNCS 4031. Springer (2006) 91–100
17. Maher, M.J.: Propositional defeasible logic has linear complexity. Theory and Practice of Logic Programming **1** (2001) 691–711
18. Udupi, Y.B., Singh, M.P.: Contract enactment in virtual organizations: A commitment-based approach. In: 21st National Conference on Artificial Intelligence, AAAI (2006) 722–727

# Reflections on Agent Beliefs

J.W. Lloyd[1] and K.S. Ng[2]

[1]Computer Sciences Laboratory
Research School of Information Sciences and Engineering
The Australian National University
jwl@mail.rsise.anu.edu.au
[2]Symbolic Machine Learning and Knowledge Acquisition
National ICT Australia*
kee.siong@nicta.com.au

**Abstract.** Some issues concerning beliefs of agents are discussed. These issues are the general syntactic form of beliefs, the logic underlying beliefs, acquiring beliefs, and reasoning with beliefs. The logical setting is more expressive and aspects of the reasoning and acquisition processes are more general than are usually considered.

## 1 Introduction

Beliefs are an important component of every agent system that assist in the selection of actions. Because of their importance, there is a huge literature on representing, reasoning with, and acquiring beliefs. This paper contributes to this literature with a setting for beliefs that employs an unusually expressive logic.

We argue that since the purpose of beliefs is to help select actions, the general syntactic form for beliefs matters and that this form should be function definitions. We also argue that it is desirable that the logic in which these definitions are written be as expressive as possible. For this reason, we admit higher-order functions so that functions may take other functions as arguments. This means that the programming idioms of functional programming are available, and that sets and multisets can be represented by abstractions. Also it is common for beliefs to have a modal nature, usually temporal or epistemic. For example, on the temporal side, it might be important that at the last time or at some time in the past, some situation held and, therefore, a certain action is now appropriate. Similarly, on the epistemic side, beliefs about the beliefs of other agents may be used to determine which action to perform. The usefulness of modal beliefs for agents is now well established, in [1] and [2], for example. Besides, introspection reveals that people use temporal and epistemic considerations when deciding what to do. These considerations lead to the choice of multi-modal, higher-order logic as the logic for the beliefs.

* NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

While many beliefs can be built into agents beforehand by their designers, it is also common for beliefs to be acquired by some kind of learning process during deployment. We discuss an approach to belief acquisition that includes as special cases simple updating, belief revision [3], and learning [4].

During action selection, it is necessary to reason about beliefs or, more accurately in our case, compute with beliefs. We discuss a computation system for the logic that greatly extends existing modal and temporal logic programming systems, and give examples to illustrate how computation works. For most applications, computation is efficient enough that it could be used to select actions in real time.

The paper provides a general discussion of these issues. The extensive technical details to support the arguments have already appeared or will soon appear elsewhere [5–9]. All the facilities described here have been implemented.

The next section contains a discussion of the necessary logical machinery. Section 3 considers the idea that beliefs should be function definitions. Section 4 shows how an agent can acquire beliefs. Section 5 discusses how reasoning with beliefs is handled. Section 6 gives some conclusions.

## 2 Logic

In this section, we outline the most relevant aspects of the logic, focussing to begin with on the monomorphic version. We define types and terms, and give an introduction to the modalities that will be most useful in this paper. Full details of the logic, including its reasoning capabilities, can be found in [8].

**Definition 1.** *An* alphabet *consists of three sets:*

1. *A set* $\mathfrak{T}$ *of type constructors.*
2. *A set* $\mathfrak{C}$ *of constants.*
3. *A set* $\mathfrak{V}$ *of variables.*

Each type constructor in $\mathfrak{T}$ has an arity. The set $\mathfrak{T}$ always includes the type constructor $\Omega$ of arity 0. $\Omega$ is the type of the booleans. Each constant in $\mathfrak{C}$ has a signature. The set $\mathfrak{V}$ is denumerable. Variables are typically denoted by $x, y, z, \ldots$. Types are built up from the set of type constructors, using the symbols $\rightarrow$ and $\times$.

**Definition 2.** *A* type *is defined inductively as follows.*

1. *If* $T$ *is a type constructor of arity* $k$ *and* $\alpha_1, \ldots, \alpha_k$ *are types, then* $T\ \alpha_1 \ldots \alpha_k$ *is a type. (Thus a type constructor of arity* 0 *is a type.)*
2. *If* $\alpha$ *and* $\beta$ *are types, then* $\alpha \rightarrow \beta$ *is a type.*
3. *If* $\alpha_1, \ldots, \alpha_n$ *are types, then* $\alpha_1 \times \cdots \times \alpha_n$ *is a type.*

The set $\mathfrak{C}$ always includes the following constants.

1. $\top$ and $\bot$, having signature $\Omega$.

2. $=_\alpha$, having signature $\alpha \to \alpha \to \Omega$, for each type $\alpha$.
3. $\neg$, having signature $\Omega \to \Omega$.
4. $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$, having signature $\Omega \to \Omega \to \Omega$.
5. $\Sigma_\alpha$ and $\Pi_\alpha$, having signature $(\alpha \to \Omega) \to \Omega$, for each type $\alpha$.

The intended meaning of $=_\alpha$ is identity (that is, $=_\alpha\ x\ y$ is $\top$ iff $x$ and $y$ are identical), the intended meaning of $\top$ is true, the intended meaning of $\bot$ is false, and the intended meanings of the connectives $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$ are as usual. The intended meanings of $\Sigma_\alpha$ and $\Pi_\alpha$ are that $\Sigma_\alpha$ maps a predicate to $\top$ iff the predicate maps at least one element to $\top$ and $\Pi_\alpha$ maps a predicate to $\top$ iff the predicate maps all elements to $\top$.

We assume there are necessity modality operators $\Box_i$, for $i = 1, \ldots, m$.

**Definition 3.** *A* term*, together with its type, is defined inductively as follows.*

1. *A variable in $\mathfrak{V}$ of type $\alpha$ is a term of type $\alpha$.*
2. *A constant in $\mathfrak{C}$ having signature $\alpha$ is a term of type $\alpha$.*
3. *If $t$ is a term of type $\beta$ and $x$ a variable of type $\alpha$, then $\lambda x.t$ is a term of type $\alpha \to \beta$.*
4. *If $s$ is a term of type $\alpha \to \beta$ and $t$ a term of type $\alpha$, then $(s\ t)$ is a term of type $\beta$.*
5. *If $t_1, \ldots, t_n$ are terms of type $\alpha_1, \ldots, \alpha_n$, respectively, then $(t_1, \ldots, t_n)$ is a term of type $\alpha_1 \times \cdots \times \alpha_n$.*
6. *If $t$ is a term of type $\alpha$ and $i \in \{1, \ldots, m\}$, then $\Box_i t$ is a term of type $\alpha$.*

Terms of the form $(\Sigma_\alpha\ \lambda x.t)$ are written as $\exists_\alpha x.t$ and terms of the form $(\Pi_\alpha\ \lambda x.t)$ are written as $\forall_\alpha x.t$ (in accord with the intended meaning of $\Sigma_\alpha$ and $\Pi_\alpha$). Thus, in higher-order logic, each quantifier is obtained as a combination of an abstraction acted on by a suitable function ($\Sigma_\alpha$ or $\Pi_\alpha$).

Constants can be declared to be *rigid*; they then have the same meaning in each world (in the semantics). A term is *rigid* if every constant in it is rigid.

If $\alpha$ is a type, then $\mathfrak{B}_\alpha$ is the set of basic terms of type $\alpha$ [5]. Basic terms represent individuals. For example, $\mathfrak{B}_\Omega$ is $\{\top, \bot\}$. Also $\mathfrak{B}_{Int}$ is $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.

The polymorphic version of the logic extends what is given above by also having available parameters which are type variables (denoted by $a, b, c, \ldots$). The definition of a type as above is then extended to polymorphic types that may contain parameters and the definition of a term as above is extended to terms that may have polymorphic types. We work in the polymorphic version of the logic in the remainder of the paper. In this case, we drop the $\alpha$ in $\exists_\alpha$, $\forall_\alpha$, and $=_\alpha$, since the types associated with $\exists$, $\forall$, and $=$ are now inferred from the context. The universal closure of a formula $\varphi$ is denoted by $\forall(\varphi)$.

An important feature of higher-order logic is that it admits functions that can take other functions as arguments. (First-order logic does not admit these so-called higher-order functions.) This fact can be exploited in applications, through the use of predicates to represent sets and predicate rewrite systems that are used for learning, for example.

Theories in the logic consist of two kinds of assumptions, global and local. The essential difference is that global assumptions are true in each world in the intended interpretation, while local assumptions only have to be true in the actual world in the intended interpretation. Each kind of assumption has a certain role to play when proving a theorem. A theory is denoted by a pair $(\mathcal{G}, \mathcal{L})$, where $\mathcal{G}$ is the set of global assumptions and $\mathcal{L}$ is the set of local assumptions.

As is well known, modalities can have a variety of meanings, depending on the application. Some of these are indicated here; much more detail can be found in [1], [2] and [8], for example.

In multi-agent applications, one meaning for $\Box_i\varphi$ is that 'agent $i$ knows $\varphi$'. In this case, the modality $\Box_i$ is written as $\boldsymbol{K}_i$.

A weaker notion is that of belief. In this case, $\Box_i\varphi$ means that 'agent $i$ believes $\varphi$' and the modality $\Box_i$ is written as $\boldsymbol{B}_i$.

The modalities also have a variety of temporal readings. We will make use of the (past) temporal modalities $\bullet$ ('last') and $\blacksquare$ ('always in the past'). We also use the modality $\blacklozenge$ ('sometime in the past'), which is dual to $\blacksquare$.

Modalities can be applied to terms that are not formulas. Thus terms such as $\boldsymbol{B}_i 42$ and $\bullet A$, where $A$ is a constant, are admitted. We will find to be particularly useful terms that have the form $\Box_{j_1} \cdots \Box_{j_r} f$, where $f$ is a function and $\Box_{j_1} \cdots \Box_{j_r}$ is a sequence of modalities.

Throughout, it is assumed that all belief bases contain the standard equality theory given in [8] which includes definitions for equality, the connectives, the quantifiers, the *if_then_else* function, an assumption that gives $\beta$-reduction, and some assumptions concerning modalities.

One of these modal assumptions is the following schema that can be used as a global assumption.

$$(\Box_i \mathbf{s}\ \mathbf{t}) = \Box_i(\mathbf{s}\ \mathbf{t}),$$

where $\mathbf{s}$ is a syntactical variable ranging over terms of type $\alpha \to \beta$ and $\mathbf{t}$ is a syntactical variable ranging over *rigid* terms of type $\alpha$. Specialised to some of the epistemic and temporal modalities discussed so far, this means, for example, that

$$(\boldsymbol{B}_i \mathbf{s}\ \mathbf{t}) = \boldsymbol{B}_i(\mathbf{s}\ \mathbf{t}) \quad \text{and} \quad (\bullet \mathbf{s}\ \mathbf{t}) = \bullet(\mathbf{s}\ \mathbf{t})$$

are global assumptions (under the rigidity assumption on $\mathbf{t}$).

Another useful global assumption in the standard equality theory is

$$\Box_i \boldsymbol{t} = \boldsymbol{t},$$

where $\boldsymbol{t}$ is a syntactical variable ranging over *rigid* terms and $i \in \{1, \ldots, m\}$. Instances of this schema that could be used as global assumptions include the following.

$$\boldsymbol{B}_i 42 = 42, \quad \boldsymbol{B}_i \top = \top \quad \text{and} \quad \bullet \bot = \bot.$$

## 3 Beliefs as Function Definitions

In this section, we discuss suitable syntactic forms for beliefs.

In [6], it was argued that beliefs should take the form of function definitions, in particular, definitions of features of states of an agent. Briefly, the motivation for this is that usually there is a very large number of states. Thus it is helpful to use the features to induce an equivalence relation on the state space so that it is only necessary to deal with a much smaller number of equivalence classes, each of which contains states that can be treated in the same way.

We take the idea that beliefs should be function definitions as a suitable starting point for the discussion here. Of course, there are still lots of different ways that one could write function definitions in the logic; we choose a particular form that is motivated by the need to acquire beliefs during deployment of the agent. We consider beliefs of the following form.

$$
\begin{array}{l}
\Box\forall x.((f\ x) = \\
\quad \mathit{if}\ (p_1\ x)\ \mathit{then}\ v_1 \\
\quad \mathit{else\ if}\ (p_2\ x)\ \mathit{then}\ v_2 \\
\qquad \vdots \\
\quad \mathit{else\ if}\ (p_n\ x)\ \mathit{then}\ v_n \\
\quad \mathit{else}\ v_0),
\end{array}
$$

where $\Box$ is a (possibly empty) sequence of modalities, $p_1, \ldots, p_n$ are predicates that can be modal and/or higher order, and $v_0, v_1, \ldots, v_n$ are suitable values (usually basic terms). Such a belief is a definition for the function $f$ in the context of the modal sequence $\Box$.

Typically, for agent $j$, beliefs have the form $\boldsymbol{B}_j\varphi$, with the intuitive meaning 'agent $j$ believes $\varphi$', where $\varphi$ is $\forall x.((f\ x) = \mathit{if}\ (p_1\ x)\ \mathit{then}\ v_1\ \ldots\ \mathit{else}\ v_0)$. Other typical beliefs have the form $\boldsymbol{B}_j\boldsymbol{B}_i\varphi$, meaning 'agent $j$ believes that agent $i$ believes $\varphi$'. If there is a temporal component to beliefs, this is often manifested by temporal modalities at the front of beliefs. Then, for example, there could be a belief of the form $\bullet^2\boldsymbol{B}_j\boldsymbol{B}_i\varphi$, whose intuitive meaning is 'at the second last time, agent $j$ believed that agent $i$ believed $\varphi$'. (Here, $\bullet^2$ is a shorthand for $\bullet\bullet$.)

While the above form for beliefs may appear to be rather specialised, it turns out to be convenient and general, and easily encompasses beliefs in more conventional form. Here is an example to illustrate how one can represent a (relational) database.

*Example 1.* Consider an agent that recommends TV programs. Amongst other things the agent will need to have access to a TV guide as part of its belief base. Represented as a relational database, the TV guide would consist of a set of tuples, where each tuple gave details of the program that is on at a certain date, time, and channel. Similarly, as a Prolog program, the TV guide would be the corresponding set of facts. Actually, neither of these representations is a good one because each ignores a functional dependency in the data: each date, time

and channel triple uniquely determines a program. Here we represent the TV guide as a function definition that correctly models this functional dependency.

For this, we require the following type synonyms.

$$Occurrence = Date \times Time \times Channel$$
$$Date = Day \times Month \times Year$$
$$Time = Hour \times Minute$$
$$Program = Title \times Duration \times (List\ Genre) \times Classification \times Synopsis.$$

Now we can give (a typical instance of) the definition of the function

$$tv\_guide : Occurrence \rightarrow Program$$

that models the TV guide.

$$\begin{array}{l}
\boldsymbol{B}_t\ \forall x.((tv_guide\ x) = \\
\quad if\ ((= ((21,7,2004),(19,30),WIN))\ x) \\
\qquad\qquad\qquad then\ (\text{“}Seinfeld\text{”}, 30, [Sitcom], PG, \text{“}Kramer \ldots\text{”}) \\
\quad else\ if\ ((= ((20,7,2004),(20,30),ABC))\ x) \\
\qquad\qquad\qquad then\ (\text{“}The\ Bill\text{”}, 50, [Drama], M, \text{“}Sun\ Hill \ldots\text{”}) \\
\qquad \vdots \\
\quad else\ (\text{“ ”}, 0, [], NA, \text{“ ”})),
\end{array}$$

where $\boldsymbol{B}_t$ is the belief modality for the TV recommender and $(\text{“ ”}, 0, [], NA, \text{“ ”})$ is the default program (where 'default' has a technical meaning [5]). It is worth noting that all the queries that one might want to pose to the relational database (or Prolog) version of the TV guide can be just as easily posed to, and answered by, the function definition form (using computation, as discussed in Section 5).

It is also straightforward to rewrite Horn clause theories, a common way of representing beliefs, as function definitions in the form above.

*Example 2.* Consider an agent with belief modality $\boldsymbol{B}$ that has beliefs of the form

$$\begin{array}{l}
\boldsymbol{B}((p\ t_1) \longleftarrow W_1) \\
\qquad \vdots \\
\boldsymbol{B}((p\ t_n) \longleftarrow W_n).
\end{array}$$

This form of belief base includes Horn clause theories and logic programs. By adding equations to the bodies and existentially quantifying free local variables in the bodies, the beliefs can be written in the form

$$\begin{array}{l}
\boldsymbol{B}((p\ x) \longleftarrow V_1) \\
\qquad \vdots \\
\boldsymbol{B}((p\ x) \longleftarrow V_n).
\end{array}$$

This set of beliefs can then be written in the function definition form

$$
\begin{array}{l}
\boldsymbol{B}\,\forall x.((p\ x) = \\
\quad \mathit{if}\ (\lambda x.V_1\ x)\ \mathit{then}\ \top \\
\qquad \vdots \\
\quad \mathit{else\ if}\ (\lambda x.V_n\ x)\ \mathit{then}\ \top \\
\quad \mathit{else}\ \bot),
\end{array}
$$

which is equivalent to the original set of beliefs under the closed world assumption. (The latter formula is essentially the completion of the original set of beliefs, probably the semantics intended anyway.)

## 4 Acquiring Beliefs

Now we turn to belief acquisition. Belief bases are generally dynamic, that is, they change from time to time during deployment of the agent. It follows that agents need to have some method by which they can acquire new beliefs. We use the phrase 'belief acquisition' to name this process. The term 'acquire' is intended to be understood in a general sense that includes 'update', 'revise' and 'learn' as special cases. 'Update' refers to the simplest form of belief acquisition in which facts are added to or deleted from a simple database, 'revise' refers to the form of acquisition that is studied in the literature on belief revision [3], and 'learning' refers to machine learning [4]. Belief acquisition thus covers the spectrum from simple updating at one end to the generalisation that is characteristic of learning at the other end.

The approach we take to belief acquisition starts from the machine learning perspective in that it extends decision-list learning in [10]. In machine learning, one wants to learn a function definition. The input to the learning process is a collection of training examples that give the value of the function for some points in its domain. A space of hypotheses is searched to find a definition for the function that agrees 'as well as possible' according to some measure with the training examples. The hypothesis learned is intended to generalise, in the sense that it should give the correct value on unseen examples.

We extend the learning process in several ways so that it also includes update and belief revision. The first extension is that training examples can give the value of the function not just on a single point of the domain but on a subset of it given by some predicate. This allows us to capture some aspects of what happens in theory revision. In addition, the predicate can include modalities. Then, in order to control where on the spectrum from updating to learning we want to be, we make a careful choice of hypothesis language. If we want simple updating, then the hypothesis language is chosen to be very specific; if we want learning, then the hypothesis language is chosen to be general; for intermediate points on the spectrum, the hypothesis language is chosen accordingly.

A major ingredient for belief acquisition is a method of generating predicates. For this, we use predicate rewrite systems which we describe informally as follows. A predicate rewrite is an expression of the form $p \rightarrowtail q$, where $p$ and $q$ are predicates (in a particular syntactic form). The predicate $p$ is called the *head* and $q$ is the *body* of the rewrite. A predicate rewrite system is a finite set of predicate rewrites. One should think of a predicate rewrite system as a kind of grammar for generating a particular class of predicates. Roughly speaking, this works as follows. Starting from the weakest predicate *top* (defined below), all predicate rewrites that have *top* (of the appropriate type) in the head are selected to make up child predicates that consist of the bodies of these predicate rewrites. Then, for each child predicate and each redex in that predicate, all child predicates are generated by replacing each redex by the body of the predicate rewrite whose head is identical to the redex. This generation of predicates continues to produce the entire space of predicates given by the predicate rewrite system. The details of the non-modal version of this can be found in [5] and the modal version in [9].

A particular predicate language, called the basic language, often arises in applications.

**Definition 4.** *Let $\alpha$ be a type. A* basic predicate *for the type $\alpha$ is one of the form $(= t)$, for some $t \in \mathfrak{B}_\alpha$.*

*The set $\mathbf{B}_\alpha = \{(= t) \mid t \in \mathfrak{B}_\alpha\}$ of basic predicates for the type $\alpha$ is called the* basic language *for the type $\alpha$.*

We distinguish two predicate languages that are used in belief acquisition. One is the *training predicate language* that is used in training examples. The general form of a training example for a function $f$ is

$$\Box \forall x.((p\ x) \rightarrow (f\ x) = v),$$

where $p$ is a predicate from the training predicate language and $v$ is a value. It is common for training predicate languages to include the corresponding basic language (of the appropriate type).

The other language is the *hypothesis predicate language* that is used in hypotheses. The predicates appearing in a belief come from the hypothesis predicate language. In the case of learning, it would be very unlikely that the hypothesis predicate language would include any basic predicates at all (because in learning one wants to generalise beyond the training examples).

Here are two examples that illustrate some of the issues for belief acquisition.

*Example 3.* This example illustrates database updating which is the simplest form of belief acquisition. We show how to acquire the database of Example 1.

First, we set up the training examples. The training predicate language is the basic language $\mathbf{B}_{Occurrence}$. A typical predicate in this language is

$$(= ((21, 7, 2004), (19, 30), \mathit{WIN})).$$

The set of values is the set of basic terms $\mathfrak{B}_{Program}$. A typical value is

$$(\text{“}\mathit{Seinfeld}\text{”}, 30, [\mathit{Sitcom}], \mathit{PG}, \text{“}\mathit{Kramer}\ldots\text{”}).$$

Training examples have the form

$$\begin{aligned}
&\boldsymbol{B}_t\ \forall x.(((= ((21,7,2004),(19,30),\mathit{WIN}))\ x) \longrightarrow \\
&\qquad (\mathit{tv_guide}\ x) = (\text{“}\mathit{Seinfeld}\text{”}, 30, [\mathit{Sitcom}], \mathit{PG}, \text{“}\mathit{Kramer}\ldots\text{”})) \\
&\boldsymbol{B}_t\ \forall x.(((= ((20,7,2004),(20,30),\mathit{ABC}))\ x) \longrightarrow \\
&\qquad (\mathit{tv_guide}\ x) = (\text{“}\mathit{The\ Bill}\text{”}, 50, [\mathit{Drama}], M, \text{“}\mathit{Sun\ Hill}\ldots\text{”}))
\end{aligned}$$

and so on.

Now we choose the hypothesis predicate language. For database updating, one wants predicates in the hypothesis predicate language to pick out individuals. Thus $\mathbf{B}_{Occurrence}$ is also chosen as the hypothesis predicate language. With this choice, the belief acquisition algorithm returns the definition for the function *tv_guide* given in Example 1.

*Example 4.* Consider a majordomo agent that manages a household. There are many tasks for such an agent to carry out including keeping track of occupants, turning appliances on and off, ordering food for the refrigerator, and so on.

Here we concentrate on one small aspect of the majordomo's tasks which is to recommend television programs for viewing by the occupants of the house. Suppose the current occupants are Alice, Bob, and Cathy, and that the agent knows the television preferences of each of them. Methods for acquiring these preferences were studied in [11]. Suppose that each occupant has a personal agent that has acquired (amongst many other functions) the function *likes* : $\mathit{Program} \rightarrow \Omega$, where *likes* is true for a program iff the person likes the program. We also suppose that the majordomo has access to the definitions of this function for each occupant, for the present time and for some suitable period into the past. Let $\boldsymbol{B}_m$ be the belief modality for the majordomo agent, $\boldsymbol{B}_a$ the belief modality for Alice, $\boldsymbol{B}_b$ the belief modality for Bob, and $\boldsymbol{B}_c$ the belief modality for Cathy. Thus part of the majordomo's belief base has the following form:

$$\begin{aligned}
&\boldsymbol{B}_m\boldsymbol{B}_a\ \forall x.((\mathit{likes}\ x) = \varphi_0) \\
&\bullet\boldsymbol{B}_m\boldsymbol{B}_a\ \forall x.((\mathit{likes}\ x) = \varphi_1) \\
&\qquad\vdots \\
&\bullet^{n-1}\boldsymbol{B}_m\boldsymbol{B}_a\ \forall x.((\mathit{likes}\ x) = \varphi_{n-1}) \\
&\bullet^{n}\boldsymbol{B}_m \forall x.(\blacklozenge\boldsymbol{B}_a(\mathit{likes}\ x) = \bot) \\
&\boldsymbol{B}_m\boldsymbol{B}_b\ \forall x.((\mathit{likes}\ x) = \psi_0) \\
&\bullet\boldsymbol{B}_m\boldsymbol{B}_b\ \forall x.((\mathit{likes}\ x) = \psi_1) \\
&\qquad\vdots \\
&\bullet^{k-1}\boldsymbol{B}_m\boldsymbol{B}_b\ \forall x.((\mathit{likes}\ x) = \psi_{k-1}) \\
&\bullet^{k}\boldsymbol{B}_m \forall x.(\blacklozenge\boldsymbol{B}_b(\mathit{likes}\ x) = \bot) \\
&\boldsymbol{B}_m\boldsymbol{B}_c\ \forall x.((\mathit{likes}\ x) = \xi_0)
\end{aligned}$$

$$\bullet \boldsymbol{B}_m \boldsymbol{B}_c \; \forall x.((\mathit{likes}\ x) = \xi_1)$$
$$\vdots$$
$$\bullet^{l-1} \boldsymbol{B}_m \boldsymbol{B}_c \; \forall x.((\mathit{likes}\ x) = \xi_{l-1})$$
$$\bullet^{l} \boldsymbol{B}_m \forall x.(\blacklozenge \boldsymbol{B}_c(\mathit{likes}\ x) = \bot),$$

for suitable $\varphi_i$, $\psi_i$, and $\xi_i$. The form these can take is explained in [11].

In the beginning, the belief base contains the formula

$$\boldsymbol{B}_m \forall x.(\blacklozenge \boldsymbol{B}_a(\mathit{likes}\ x) = \bot),$$

whose purpose is to prevent runaway computations into the infinite past for certain formulas of the form $\blacklozenge\varphi$. The meaning of this formula is "the agent believes that for all programs it is not true that at some time in the past Alice likes the program". After $n$ time steps, this formula has been transformed into

$$\bullet^{n} \boldsymbol{B}_m \forall x.(\blacklozenge \boldsymbol{B}_a(\mathit{likes}\ x) = \bot).$$

In general, at each time step, the beliefs about *likes* at the previous time steps each have another $\bullet$ placed at their front to push them one step further back into the past, and a new current belief about *likes* is acquired. (For this application, a time step could occupy hours, days, or even longer, depending on how often the beliefs need to be updated.)

Based on these beliefs about the occupant preferences for TV programs, the task for the agent is to recommend programs that all three occupants would be interested in watching together. The simplest idea is that the agent should only recommend programs that all three occupants currently like. But it is possible that less stringent conditions might also be acceptable; for example, it might be sufficient that two of the occupants currently like a program but that the third has liked the program in the past (even if they do not like it at the present time). A (simplified) predicate rewrite system suitable for giving an hypothesis predicate language for such an acquisition task is as follows.

$$\mathit{top} \rightarrowtail \wedge_3\ \mathit{top}\ \mathit{top}\ \mathit{top}$$
$$\mathit{top} \rightarrowtail \vee_2\ \mathit{top}\ \mathit{top}$$
$$\mathit{top} \rightarrowtail \boldsymbol{B}_i \mathit{likes} \quad \% \text{ for each } i \in \{a, b, c\}$$
$$\mathit{top} \rightarrowtail \blacklozenge \boldsymbol{B}_i \mathit{likes} \quad \% \text{ for each } i \in \{a, b, c\}.$$

Here, the function $\mathit{top} : a \rightarrow \Omega$ is defined by $(\mathit{top}\ x) = \top$, for each $x$. The function

$$\wedge_3 : (a \rightarrow \Omega) \rightarrow (a \rightarrow \Omega) \rightarrow (a \rightarrow \Omega) \rightarrow a \rightarrow \Omega$$

is defined by $\wedge_3\ p_1\ p_2\ p_3\ x = (p_1\ x) \wedge (p_2\ x) \wedge (p_3\ x)$, for each $x$. The function $\vee_2$, which defines 'disjunction' at the predicate level for two arguments, is defined analogously.

Let $group\_likes : Program \rightarrow \Omega$ be the function that the agent needs to acquire. Thus the informal meaning of $group\_likes$ is that it is true for a program iff the occupants collectively like the program. (This may involve a degree of compromise by some of the occupants.) The training predicate language is $\mathbf{B}_{Program}$, so that training examples for this task look like

$$\boldsymbol{B}_m \forall x.(((= P_1)\ x) \longrightarrow (group\_likes\ x) = \top)$$
$$\boldsymbol{B}_m \forall x.(((= P_2)\ x) \longrightarrow (group\_likes\ x) = \bot),$$

where $P_1$ and $P_2$ are particular programs. The definition of a typical function that might be acquired from training examples and the hypothesis predicate language given by the above predicate rewrite system is as follows.

$$\begin{aligned}\boldsymbol{B}_m \forall x.\, &((group\_likes\ x) = \\ &\quad \mathit{if}\ ((\wedge_3\ \blacklozenge \boldsymbol{B}_a likes\ \boldsymbol{B}_b likes\ \boldsymbol{B}_c likes)\ x)\ \mathit{then}\ \top \\ &\quad \mathit{else\ if}\ ((\wedge_3\ \boldsymbol{B}_c likes\ (\vee_2\ \boldsymbol{B}_a likes\ \boldsymbol{B}_b likes)\ top)\ x)\ \mathit{then}\ \top \\ &\quad \mathit{else}\ \bot).\end{aligned}$$

Now let $P$ be some specific program. Suppose that a computation shows that $\boldsymbol{B}_m((group\_likes\ P) = \bot)$ is a consequence of the belief base of the agent. On this basis, the agent will presumably not recommend to the occupants that they watch program $P$ together.

## 5 Reasoning with Beliefs

As well as representing knowledge, it is necessary to reason with it. The reasoning system for the logic combines a theorem prover and an equational reasoning system. The theorem prover is a fairly conventional tableau theorem prover for modal higher-order logic similar to what is proposed in [12]. The equational reasoning system is, in effect, a computational system that significantly extends existing declarative programming languages by adding facilities for computing with modalities. The proof component and the computational component are tightly integrated, in the sense that either can call the other. Furthermore, this synergy between the two makes possible all kinds of interesting reasoning tasks. For agent applications, the most common reasoning task is a computational one, that of evaluating a function call. In this case, the theorem-prover plays a subsidiary role, usually that of performing some rather straightforward modal theorem-proving tasks. However, in other applications it can just as easily be the other way around with the computational system performing subsidiary equational reasoning tasks for the theorem prover.

Here we concentrate on computation. As motivation for what computation actually means, consider the problem of determining the meaning of a term $t$ in the intended interpretation (for some application). If a formal definition of the intended interpretation is available, then this problem can be solved (under some finiteness assumptions). However, we assume here that the intended interpretation is not available, as is usually the case, so that the problem cannot be

solved directly. Nevertheless, there is still a lot that can be done if the theory $\mathcal{T}$ of the application is available and enough of it is in equational form. Intuitively, if $t$ can be 'simplified' sufficiently using $\mathcal{T}$, its meaning may become apparent even in the absence of detailed knowledge of the intended interpretation. For example, if $t$ can be simplified to a term containing only data constructors, then the meaning of $t$ will generally be obvious.

More formally, the *computation problem* is as follows.

> Given a theory $\mathcal{T}$, a term $t$, and a sequence $\Box_{j_1} \cdots \Box_{j_r}$ of modalities, find a 'simpler' term $t'$ such that $\Box_{j_1} \cdots \Box_{j_r} \forall (t = t')$ is a consequence of $\mathcal{T}$.

Thus $t$ and $t'$ have the same meaning in all worlds accessible from the point world in the intended interpretation according to the modalities $\Box_{j_1} \cdots \Box_{j_r}$.

Here now is the definition of a mechanism that addresses the computational problem by employing equational reasoning to rewrite terms to 'simpler' terms that have the same meaning. To simplify matters, we only consider the case when the computation does not need to call on the theorem prover. (This is the rank 0 case in [8].) In the following definition, a modal path to a subterm is the sequence of indices of modalities whose scope one passes through when going down to the subterm. A substitution is admissible if any term that replaces a free occurrence of a variable that is in the scope of a modality is rigid.

**Definition 5.** *Let $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ be a theory. A* computation using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{T}$ *is a sequence $\{t_i\}_{i=1}^{n}$ of terms such that the following conditions are satisfied.*

1. *For $i = 1, \ldots, n-1$, there is*
   (a) *a subterm $s_i$ of $t_i$ at occurrence $o_i$, where the modal path to $o_i$ in $t_i$ is $k_1 \ldots k_{m_i}$,*
   (b) *i. a formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall (u_i = v_i)$ in $\mathcal{L}$, or*
   *ii. a formula $\forall (u_i = v_i)$ in $\mathcal{G}$, and*
   (c) *a substitution $\theta_i$ that is admissible with respect to $u_i = v_i$*

   *such that $u_i\theta_i$ is $\alpha$-equivalent to $s_i$ and $t_{i+1}$ is $t_i[s_i / v_i\theta_i]_{o_i}$.*

   *The term $t_1$ is called the* goal *of the computation and $t_n$ is called the* answer*.*
   *Each subterm $s_i$ is called a* redex*.*
   *Each formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall (u_i = v_i)$ or $\forall (u_i = v_i)$ is called an* input equation*.*
   *The formula $\Box_{j_1} \cdots \Box_{j_r} \forall (t_1 = t_n)$ is called the* result *of the computation.*

The treatment of modalities in a computation has to be carefully handled. The reason is that even such a simple concept as applying a substitution is greatly complicated in the modal setting by the fact that constants generally have different meanings in different worlds and therefore the act of applying a substitution may not result in a term with the desired meaning. This explains the restriction to admissible substitutions in the definition of computation. It also explains why, for input equations that are local assumptions, the sequence

of modalities $\Box_{k_1} \cdots \Box_{k_{m_i}}$ whose scopes are entered going down to the redex must appear in the modalities at the front of the input equation. (For input equations that are global assumptions, in effect, every sequence of modalities that we might need is implicitly at the front of the input equation.)

In the general case, an input equation can also be a theorem that was proved by the theorem-proving component of the reasoning system, as the examples below show.

Here are two examples to illustrate various aspects of computation.

*Example 5.* Consider a belief base for an agent that contains the definition

$$\boldsymbol{B}\ \forall x.((f\ x) = \\ \quad \mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0),$$

where $A, B, C : \sigma$, $f : \sigma \to \mathit{Nat}$ and $\boldsymbol{B}$ is the belief modality for the agent. With such a definition, it is straightforward to compute in the 'forward' direction. Thus $(f\ B)$ can be computed in the obvious way to produce the answer 21 and the result $\boldsymbol{B}((f\ B) = 21)$.

Less obviously, the definition can be used to compute in the 'reverse' direction. For example, consider the computation of $\{x \mid (f\ x) = 42\}$ in Figure 1, which produces the answer $\{A, C\}$. The redexes selected are underlined. This computation makes essential use of the equations

$$(w\ \ \mathit{if}\ x\ \mathit{then}\ y\ \mathit{else}\ z) = \mathit{if}\ x\ \mathit{then}\ (w\ y)\ \mathit{else}\ (w\ z)$$
$$(\mathit{if}\ x\ \mathit{then}\ y\ \mathit{else}\ z\ \ w) = \mathit{if}\ x\ \mathit{then}\ (y\ w)\ \mathit{else}\ (z\ w)$$

from the standard equality theory.

*Example 6.* This example illustrates computation using a belief base that has been obtained by incremental belief acquisition and that exploits modalities acting on arbitrary terms. Consider an agent with belief modality $\boldsymbol{B}$ and a belief base that includes definitions of the function $f : \sigma \to \mathit{Nat}$ at the current time and some recent times. Suppose at the current time the part of the belief base concerning $f$ is as follows.

$$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_4\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ \mathit{if}\ (p_5\ x)\ \mathit{then}\ 84\ \mathit{else}\ 0)$$
$$\bullet\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_3\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ 0)$$
$$\bullet^2\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_1\ x)\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ (p_2\ x)\ \mathit{then}\ 21\ \mathit{else}\ 0)$$
$$\bullet^3\boldsymbol{B}\ \forall x.((f\ x) = 0).$$

Three time steps ago, the function $f$ was 0 everywhere. Two time steps ago, the definition

$$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_1\ x)\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ (p_2\ x)\ \mathit{then}\ 21\ \mathit{else}\ 0)$$

for $f$ was acquired. Then, one time step ago, the definition

$$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ (p_3\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ 0)$$

$\{x \mid \underline{(f\ x)} = 42\}$

$\{x \mid \underline{((=\ \ \mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \underline{(\mathit{if}\ x = A\ \mathit{then}\ (=\ 42)\ \mathit{else}\ (=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \underline{(42 = 42)}\ \mathit{else}\ ((=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)\ 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \underline{((=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \underline{(\mathit{if}\ x = B\ \mathit{then}\ (=\ 21)\ \mathit{else}\ (=\ \ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \underline{(21 = 42)}\ \mathit{else}\ ((=\ \ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)\ 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \underline{((=\ \ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)} 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \underline{(\mathit{if}\ x = C\ \mathit{then}\ (=\ 42)\ \mathit{else}\ (=\ 0)\ \ 42)}\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \underline{(42 = 42)}\ \mathit{else}\ (0 = 42)\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \underline{(0 = 42)}\}$

$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \bot\}$

**Fig. 1.** Computation using $\boldsymbol{B}$ of $\{x \mid (f\ x) = 42\}$

for $f$ was acquired. This definition states that, on the region defined by $p_3$, $f$ is the same as the $f$ at the last time step; and, otherwise, $f$ is 0. Finally, we come to the current definition, which on the region defined by $p_4$ is the same as the $f$ at the last time step; on the region defined by $p_5$ is 84; and, otherwise, $f$ is 0. Definitions like these which use earlier definitions arise naturally in incremental belief acquisition. A technical device needed to achieve incrementality is to admit values of the form $(\bullet^k f\ x)$, so that earlier definitions become available for use. In turn this depends crucially on being able to apply modalities to arbitrary terms, in this case, functions.

Now suppose $t$ is a rigid term of type $\sigma$ and consider the computation using $\boldsymbol{B}$ of $(f\ t)$ in Figure 2. Note how earlier definitions for $f$ get used in the computation: at the step $\bullet\underline{(f\ t)}$, the definition at the last time step gets used, and at the step $\bullet^2\underline{(f\ t)}$, the definition from two time steps ago gets used.

Also needed in this computation is the instance $(\bullet f\ \ t) = \bullet(f\ \ t)$ of the global assumption discussed in Section 2. Incidentally, the assumption that the argument to a function like $f$ is rigid is a weak one; in typical applications, the argument will naturally be rigid.

It is assumed that the belief base of the agent contains the global assumption

$$\bullet \boldsymbol{B}\varphi \longrightarrow \boldsymbol{B}\bullet\varphi.$$

Using this assumption, it can be proved that

$$\boldsymbol{B}\bullet\ \forall x.((f\ x) = \mathit{if}\ (p_3\ x)\ \mathit{then}\ (\bullet f\ x)\ \mathit{else}\ 0)$$

and

$$\boldsymbol{B}\bullet^2\ \forall x.((f\ x) = \mathit{if}\ (p_1\ x)\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ (p_2\ x)\ \mathit{then}\ 21\ \mathit{else}\ 0)$$

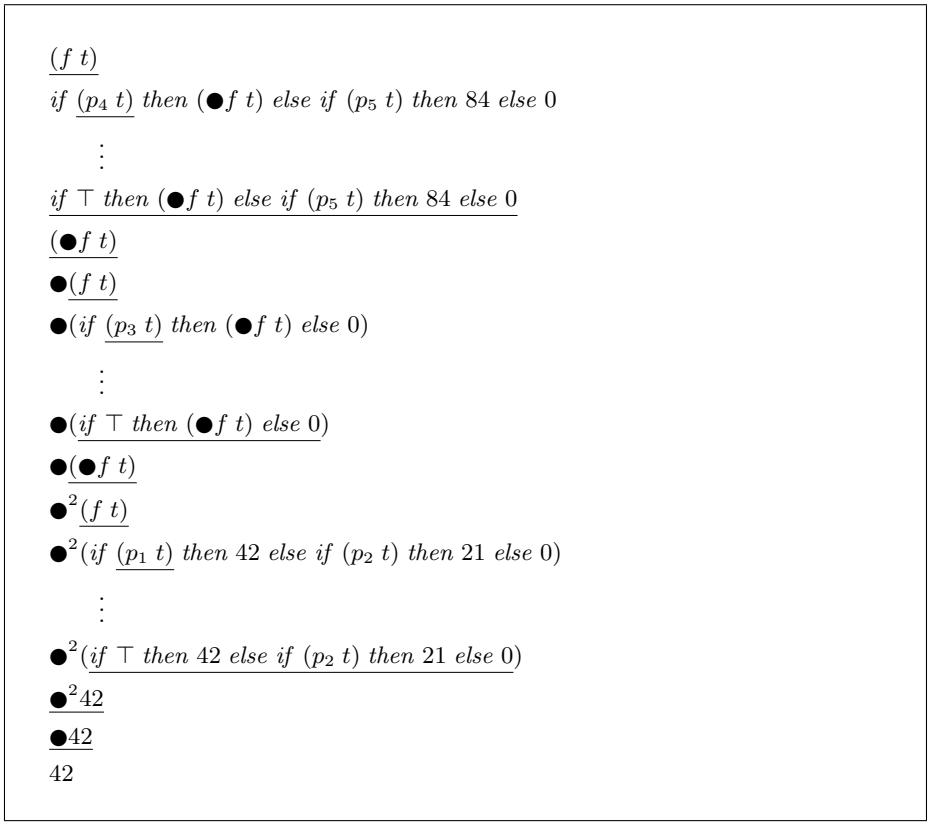



**Fig. 2.** Computation using $\boldsymbol{B}$ of $(f\ t)$

are consequences of the belief base. These can then be used as input equations in the computation.

The computation shows that $\boldsymbol{B}((f\ t) = 42)$ is a consequence of the belief base. Thus the agent believes that the value of $(f\ t)$ is 42; on the basis of this and other similar information, it will select an appropriate action.

## 6 Conclusion

In this paper, we have reflected on some issues concerning beliefs for agents. The main conclusion we draw from this is the value of using a highly expressive logic for representing beliefs. Temporal and epistemic modalities allow beliefs to capture information about an environment that can be crucial when an agent is trying to select an appropriate action. For beliefs, propositional logic is not particularly useful and so it is necessary to move beyond the propositional case; we argue for the use of higher-order logic because of its extra expressive power.

## References


1. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press (1995)
2. Gabbay, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-Dimensional Modal Logics: Theory and Applications. Studies in Logic and The Foundations of Mathematics, Volume 148. Elsevier (2003)
3. Alchourrón, C., Gärdenfors, P., Makinson, D.: On the logic of theory change: Partial meet contraction and revision functions. Journal of Symbolic Logic **50**(2) (1985) 510–530
4. Mitchell, T.: Machine Learning. McGraw-Hill (1997)
5. Lloyd, J.: Logic for Learning. Cognitive Technologies. Springer (2003)
6. Lloyd, J., Sears, T.: An architecture for rational agents. In Baldoni, M., *et al*, eds.: Declarative Agent Languages and Technologies (DALT 2005), Springer, LNAI 3904 (2006) 51–71
7. Lloyd, J., Ng, K.: Learning modal theories. In Muggleton, S., Otero, R., eds.: Proceedings of the 16th International Conference on Inductive Logic Programming (ILP2006), Springer, LNAI (to appear April, 2007) `http://csl.anu.edu.au/~jwl`.
8. Lloyd, J.: Knowledge representation and reasoning in modal higher-order logic. submitted for publication. `http://csl.anu.edu.au/~jwl` (2006)
9. Lloyd, J., Ng, K.: Belief acquisition for agents. In preparation (2007)
10. Rivest, R.: Learning decision lists. Machine Learning **2**(3) (1987) 229–246
11. Cole, J., Gray, M., Lloyd, J., Ng, K.: Personalisation for user agents. In Dignum, F., *et al*, eds.: Fourth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 05). (2005) 603–610
12. Fitting, M.: Types, Tableaus, and Gödel's God. Kluwer Academic Publishers (2002)

# Composing high-level plans for declarative agent programming

Felipe Meneguzzi and Michael Luck

Department of Computer Science
King's College London
felipe.meneguzzi@kcl.ac.uk
michael.luck@kcl.ac.uk

**Abstract.** Research on practical models of autonomous agents has largely focused on a *procedural* view of goal achievement. This allows for efficient implementations, but prevents an agent from reasoning about alternative courses of action for the achievement of its design objectives. In this paper we show how a procedural agent model can be modified to allow an agent to compose existing plans into new ones at runtime to achieve desired world states. This new agent model can be used to implement a declarative goals interpreter, since it allows designers to specify *only* the desired world states in addition to an agent's basic capabilities, enhancing the agent's ability to deal with failures. Moreover our approach allows the new plans to be included in the plan library, effectively enabling the agent to improve its runtime performance over time.

## 1 Introduction

The notion of autonomous intelligent agents has become increasingly relevant in recent years both in relation to numerous real applications and in drawing together different artificial intelligence techniques. Perhaps the best known and most used family of agent architectures is that based around the notions of beliefs, desires and intentions, which is exemplified by such systems as PRS, dMARS and AgentSpeak [1]. For reasons of efficiency and real-time operation, these architectures have been based around the inclusion of a plan library consisting of predefined *encapsulated procedures*, or *plans*, coupled with information about the context in which to use them [2]. However, designing agents in this way severely limits an agent's runtime flexibility, as the agent depends entirely on the designer's previous definition of all possible courses of action associated with proper contextual information to allow the agent to adopt the right plans in the right situations.

Typically, agent interpreters select plans using more or less elaborate algorithms, but these seldom have any knowledge of the contents of the plans, so that plan selection is ultimately achieved using fixed rules, with an agent adopting *black box* plans based solely on the contextual information that accompanies them. Alternatively, some agent interpreters allow for plan modification rules to

allow plans to be modified to suit the current situation [3], but this approach still relies on a designer establishing a set of rules that considers all potentially necessary modifications for the agent to achieve its goals. The problem here is that for some domains, an agent description must either be extremely extensive (requiring a designer to foresee every possible situation the agent might find itself in), or will leave the agent unable to respond under certain conditions.

This *procedural* response to goal achievement has been favoured to enable the construction of practical systems that are usable in real-world applications. However, it also causes difficulties in cases of failure. When a procedural agent selects a plan to achieve a given goal it is possible that the selected plan may fail, in which case the agent typically concludes that the goal has also failed, regardless of whether other plans to achieve the same goal might have been successful. By neglecting the *declarative* aspect of goals in not considering the construction of plans on-the-fly, agents lose the ability to reason about alternative means of achieving a goal, making it possible for poor plan selection to lead to an otherwise avoidable failure.

In this paper we describe how a procedural agent model can be modified to allow an agent to build new plans at runtime by chaining existing fine-grained plans from a plan library into high-level plans. We demonstrate the applicability of this approach through a modification to the AgentSpeak architecture, allowing for a combination of declarative and procedural aspects. This modification requires no change to the plan language, allowing designers to specify predefined procedures for known tasks under ideal circumstances, but also allowing the agent to form new plans when unforeseen situations arise. Though we demonstrate this technique for AgentSpeak, it can be easily applied to other agent architectures with an underlying procedural approach to reasoning, such as JADEX or the basic 3APL [4]. The key contribution is a method to augment an agent's runtime flexibility, allowing it to add to its plan library to respond to new situations without the need for the designer to specify all possible combinations of low-level operators in advance. The paper is organised as follows: in Section 2 we briefly review relevant aspects of AgentSpeak, in order to introduce the planning capability in Section 3; in Section 4 a classic example is provided to contrast our approach to that of traditional AgentSpeak; in Section 5 we compare our work with similar or complementary approaches that also aim to improve agent autonomy; finally, in Section 6 a summary of the contribution is provided along with further work that can be carried out to improve our system.

## 2 AgentSpeak

AgentSpeak [2] is an agent language that allows a designer to specify a set of procedural plans which are then selected by an interpreter to achieve the agent's design goals. It evolved from a series of procedural agent languages originally developed by Rao and Georgeff [5]. In AgentSpeak an agent is defined by a set of beliefs and a set of plans, with each plan encoding a procedure that is assumed to bring about a desired state of affairs, as well as the context in which

a plan is relevant. Goals in AgentSpeak are implicit, and plans intended to fulfil them are invoked whenever some triggering condition is met in a certain context, presumably the moment at which this implicit goal becomes relevant.

The control cycle of an AgentSpeak interpreter is driven by events on data structures, including the addition or deletion of goals and beliefs. These events are used as triggering conditions for the adoption of plans, so that adding an achievement goal means that an agent desires to fulfil that goal, and plans whose triggering condition includes that goal (*i.e.* are *relevant* to the goal) should lead to that goal being achieved. Moreover, a plan includes a logical condition that specifies when the plan is *applicable* in any given situation. Whenever a goal addition event is generated (as a result of the currently selected plan having subgoals), the interpreter searches the set of relevant plans for applicable plans; if one (or more) such plan is found, it is pushed onto an intention structure for execution. Elements in the intention structure are popped and handled by the interpreter. If the element is an action this action is executed, while if the element is a goal, a new plan is added into the intention structure and processed. During this process, failures may take place either in the execution of actions, or during the processing of subplans. When such a failure takes place, the plan that is currently being processed also fails. Thus, if a plan selected for the achievement of a given goal fails, the default behaviour of an AgentSpeak agent is to conclude that the goal that caused the plan to be adopted is not achievable. This control cycle is illustrated in the diagram of Figure 1,[1] and strongly couples plan execution to goal achievement.

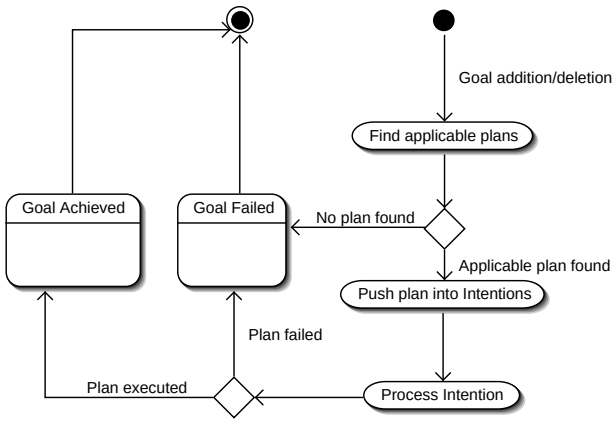



**Fig. 1.** AgentSpeak control cycle.

[1] For a full description of AgentSpeak, refer to d'Inverno *et al.* [1]

The control cycle of Figure 1 allows for situations in which the poor selection of a plan leads to the failure of a goal that would otherwise be achievable through a different plan in the plan library. While such limitations can be mitigated through meta-level [6] constructs that allow goal addition events to cause the execution of applicable plans in sequence, and the goal to fail only when *all* plans fail, AgentSpeak still regards goal achievement as an implicit side-effect of a plan being executed successfully.

## 3 Planning in an AgentSpeak interpreter

In response to these limitations, we have created an extension of AgentSpeak that allows an agent to explicitly specify the world-state that should be achieved by the agent. In order to transform the world to meet the desired state, the agent uses a propositional planner to form high-level plans through the composition of plans already present in its plan library. This propositional planner is invoked by the agent through a regular AgentSpeak action, and therefore requires no change in the language definition. The only assumption we make is the existence of plans that abide by certain restrictions in order to be able to compose higher-level plans taking advantage of planning capabilities introduced in the interpreter. Whenever an agent needs to achieve a goal that involves planning, it uses a special planning action that converts the low-level procedural plans of AgentSpeak into STRIPS operators and invokes the planning module. If the planner succeeds in finding a plan, it is converted back into a high-level AgentSpeak plan and added to the intention structure for execution. Here, we liken the low-level procedural plans of AgentSpeak to STRIPS operators, connecting the agent interpreter to the planner by converting one formalism into the other and *vice versa*. We have chosen to use STRIPS as the planning language in this paper for simplicity reasons, and this approach would not lose applicability if one was to use PDDL [7] (or another language) as the planning language.

### 3.1 The planning action

In order to describe the connection of the planning component with AgentSpeak, we need to review the main constructs of this agent language. As we have seen, an AgentSpeak interpreter is driven by events on the agent's data structures that may trigger the adoption of plans. Additions and deletions of goals and beliefs are represented by the plus (+) and minus (−) sign respectively. Goals are distinguished into *test goals* and *achievement goals*, denoted by a preceding question mark (?), or an exclamation mark (!), respectively. For example, the addition of a goal to achieve $g$ would be represented by $+!g$. Belief additions and deletions arise as the agent perceives the environment, and are therefore outside its control, while goal additions and deletions only arise as part of the execution of an agent's plans.

In our approach, in addition to the traditional way of encoding goals for an AgentSpeak agent implicitly as triggering events consisting of achievement goals

| $+goal\_conj(Goals) : true \leftarrow plan(Goals).$ |
|---|

**Table 1:** Planner invocation plan.

($!goal$), we allow desires including multiple beliefs $(b_1, \ldots, b_n)$ describing a desired world-state in the form $goal\_conj([b_1, \ldots, b_n])$. An agent desire description consists of a conjunction of beliefs the agent wishes to be true simultaneously at a given point in time. The execution of the planner component is triggered by an event $+goal\_conj([b_1, \ldots, b_n])$ as shown in Table 1.

Now, the key to our approach to planning in AgentSpeak is the introduction of a special *planning action*, denoted $plan(G)$, where $G$ is a conjunction of desired goals. This action is bound to an implementation of a planning component, and allows all of the process regarding the conversion between formalisms to be encapsulated in the action implementation, making it completely transparent to the remainder of the interpreter.

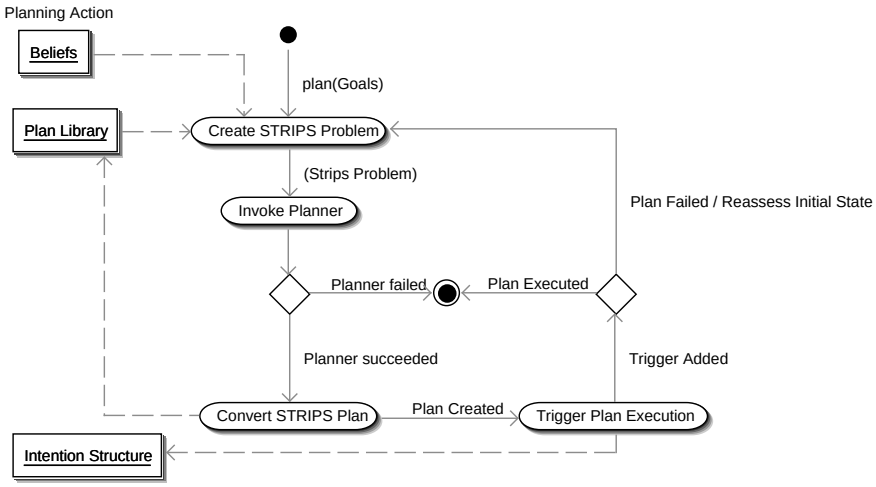



**Fig. 2.** Operation of the planning action.

As illustrated in Figure 2, the internal action to plan takes as an argument the desired world-state, and uses this, along with the current belief database and the plan library, to generate a STRIPS [8] planning problem. This action then invokes a planning algorithm; if a plan is found, the planning action succeeds, otherwise the planning action fails. If the action successfully yields a plan, it converts the resulting STRIPS plan into a new AgentSpeak plan to be added to the plan library, and immediately triggers the adoption of the new plan. If the

$$+!move\_to(A, B) : available(car) \leftarrow get(car); drive(A, B).$$

$$+!move\_to(A, B) : \neg available(car) \leftarrow walk(A, B).$$

**Table 2:** Movement plans.

newly created plan fails, the planner may then be invoked again to try and find another plan to achieve the desired state of affairs, taking into consideration any changes in the agent beliefs.

### 3.2 Chaining plans into higher-level plans

The design of a traditional AgentSpeak plan library follows a similar approach to programming in procedural languages, where a designer typically defines fine-grained actions to be the building blocks of more complex operations. These building blocks are then assembled into higher-level procedures to accomplish the main goals of a system. Analogously, an AgentSpeak designer traditionally creates fine-grained *plans* to be the building blocks of more complex operations, typically defining more than one plan to satisfy the same goal (*i.e.* sharing the same trigger condition), while specifying the situations in which it is applicable through the context part of each plan. Here, we are likening STRIPS actions to low-level AgentSpeak *plans*, since the effects of primitive AgentSpeak actions are not explicitly defined in an agent description. For example, an agent that has to move around in a city could know many ways of going from one place to another depending on which vehicle is available to it, such as by walking or driving a car, as shown in Table 2.

Modelling STRIPS operators to be supplied to a planning algorithm is similar to the definition of these building-block procedures. In both cases, it is important that operators to be used sequentially *fit*. That is, the results from applying one operator should be compatible with the application of the possible subsequent operators, matching the effects of one operator to the preconditions of the next operator.

Once the building-block procedures are defined, higher-level operations must be defined to fulfil the broader goals of a system by combining these building blocks. In a traditional AgentSpeak plan library, higher-level plans to achieve broader goals contain a series of goals to be achieved by the lower-level operations. This construction of higher-level plans that make use of lower-level ones is analogous to the planning performed by a propositional planning system. By doing the *planning themselves*, *designers* must cope with every foreseeable situation the agent might find itself in, and generate higher-level plans combining lower-level tasks accordingly. Moreover, the designer must make sure that the

subplans being used do not lead to conflicting situations. This is precisely the responsibility we intend to delegate to a STRIPS planner.

Plans resulting from propositional planning can then be converted into sequences of AgentSpeak achievement goals to comprise the body of new plans available within an agent's plan library. In this approach, an agent can still have high-level plans pre-defined by the designer, so that routine tasks can be handled exactly as intended. At the same time, if an unforseen situation presents itself to the agent, it has the flexibility of finding novel ways to solve problems, while augmenting the agent's plan library in the process.

Clearly, lower-level plans defined by the designer can (and often will) include the invocation of *atomic actions* intended to generate some effect on the environment. Since the effects of these actions are not usually explicitly specified in AgentSpeak (another example of reasoning delegated to the designer), an agent cannot reason about the consequences of these actions. When designing agents using our model, we expect designers to explicitly define the consequences of executing a given AgentSpeak plan in terms of belief additions and deletions in the plan body as well as atomic action invocations. The conversion process can then ignore atomic action invocations when generating a STRIPS specification.

### 3.3 Translating AgentSpeak into STRIPS

Once the need for planning is detected, the plan in Table 1 is invoked so that the agent can tap into a planner component. The process of linking an agent to a propositional planning algorithm includes converting an AgentSpeak plan library into propositional planning operators, declarative goals into goal-state specifications, and the agent beliefs into the initial-state specification for a planning problem. After the planner yields a solution, the ensuing STRIPS plan is translated into an AgentSpeak plan in which the operators resulting from the planning become subgoals. That is, the execution of each operator listed in the STRIPS plan is analogous to the insertion of the AgentSpeak plan that corresponded to that operator when the STRIPS problem was created.

Plans in AgentSpeak are represented by a header comprising a triggering condition and a context, as well as a body describing the steps the agent takes when a plan is selected for execution. If $e$ is a triggering event, $b_1, \ldots, b_m$ are belief literals, and $h_1, \ldots, h_n$ are goals or actions, then $e : b_1 \& \ldots \& b_m \leftarrow h_1; \ldots; h_n.$ is a plan. As an example, let us consider a triggering plan for accomplishing `!move(A,B)` corresponding to a movement from A to B, where:

- $e$ is `!move(A,B)`;
- `at(A) & not at(B)` are belief literals; and
- `-at(A); +at(B).` is the plan body, containing information about belief additions and deletions.

The plan is then as follows:

```
+!move(A,B) : at(A) & not at(B)
    <- -at(A);
       +at(B).
```

When this plan is executed, it results in the agent believing it is no longer in position A, and then believing it is in position B. For an agent to rationally want to move from A to B, it must believe it is at position A and not already at position B.

In the classical STRIPS notation, operators have four components: an identifier, a set of preconditions, a set of predicates to be added (*add*), and a set of predicates to be deleted (*del*). For example, the same `move` operator can be represented in STRIPS following the correspondence illustrated in Figure 3, in which we convert the AgentSpeak invocation condition into a STRIPS operator header, a context condition into an operator precondition, and the plan body is used to derive add and delete lists.

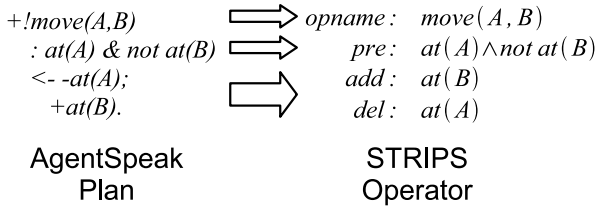



**Fig. 3.** Correspondence between an AgentSpeak plan and a STRIPS operator.

A relationship between these two definitions is not hard to establish, and we define the following algorithm for converting AgentSpeak plans into STRIPS operators. Let $e$ be a triggering event, $b_1 \& \ldots \& b_m$ a conjunction of belief literals representing a plan's context, and $a_1, \ldots, a_n$ be belief addition actions and $d_1, \ldots, d_o$ be belief deletion actions within a plan's body. All of these elements can be represented in a single AgentSpeak plan. Moreover let *opname* be the operator name and parameters, *pre* be the preconditions of the operator, *add* the predicate addition list and *del* the predicate deletion list. Mapping an AgentSpeak plan into STRIPS operators is accomplished as follows:

1. $opname = e$
2. $pre = b_1 \& \ldots \& b_m$
3. $add = a_1, \ldots, a_n$
4. $del = d_1, \ldots, d_o$

In Section 3.1 we introduced the representation of a conjunction of desired goals as the predicate $goal\_conj([b_1, \ldots, b_n])$. The list $[b_1, \ldots, b_n]$ of desires is directly translated into the goal state of a STRIPS problem. Moreover, the initial state specification for a STRIPS problem is generated directly from the agent's belief database.

$$+goal\_conj(Goals) : true \leftarrow !op_1; \ldots; !op_n.$$

**Table 3:** AgentSpeak plan generated from a STRIPS plan.

### 3.4 Executing generated plans

The STRIPS problem generated from the set of operators, initial state and goal state is then processed by a propositional planner. If the planner fails to generate a propositional plan for that conjunction of literals, the plan in Table 1 fails immediately and this goal is deemed unachievable, otherwise the resulting propositional plan is converted into an AgentSpeak plan and added to the intention structure.

A propositional plan from a STRIPS planner is in the form of a sequence $op_1, \ldots, op_n$ of operator names and instantiated parameters. We define a new AgentSpeak plan in Table 3, where $goal\_conj(Goals)$ is the event that initially caused the planner to be invoked.

Immediately after adding the new plan to the plan library, the event $goal\_conj(Goals)$ is reposted to the agent's intention structure, causing the generated plan to be executed. Plans generated in this fashion are admittedly simple, since the development of a complete process of plan generalisation is not a trivial matter since, for instance, it involves solving the issue of deriving the context condition adequately. An extremely simple solution for this problem uses the entire belief base of the agent as context for that plan, but this solution includes a great number of beliefs that are probably irrelevant to the goal at hand, severely limiting this plan's future applicability. Another solution involves replicating the preconditions of the first operator for the new plan, but this could also lead the agent to fail to execute the plan later on. We have developed an algorithm to derive a minimal set of preconditions, which we omit here due to space constraints, showing instead the simple solution of using a constantly true context. Another possible refinement to the conversion of a STRIPS plan into an AgentSpeak plan is to allow the same generated plan to be reused to handle side-effects of the set of goals that led to its generation. For example, a plan for a conjunction of goals $g$ can be used to achieve any subset $g'$ of $g$.

In the ensuing execution of the generated plan, the fact that multiple concurrent plans might be stacked in an agent's intentions structure must also be addressed. There are multiple ways of addressing this issue, namely:

- delegate the analysis and resolution of conflicting interaction between plans to the designer;
- implement provisions to ensure that the plans used by the planner process are executed atomically;
- drop the entire intention structure before plan adoption and prevent new intentions from being adopted during plan execution; and

– analyse the current intention structure and prospective plan steps during planning to ensure they do not interfere with each other.

The first way of resolving concurrency problems is the traditional solution in an AgentSpeak context, but it is clearly not acceptable, since the main goal of our extension is to diminish the amount of designer tasks. On the other hand, the last alternative involves the introduction of a complex analysis procedure to solve a very limited number of potential conflicts. In our work we considered the second and third ways of dealing with concurrency problems, and in the prototype described in Section 4 we opted to enable the agent to execute dynamically generated plans atomically (by preventing other intentions to be selected from the stack while a dynamic plan was being executed).

## 4 Experiments and Results

We have implemented the planning action described in Section 3 using Jason [9], which is an open-source Java implementation of AgentSpeak that includes a number of extensions, such as facilities for communication and distribution. In addition to providing an interpreter for the agent language, Jason has an object-oriented API for the development of *actions* available to the agents being developed. Since planning is to be performed as part of a regular AgentSpeak plan, the planning action encapsulates the conversion process of Section 3.3 using Jason's *internal actions*.

This implementation was used in a number of toy problems, such as the Blocks world used with the original STRIPS planner [8], as well as some examples from the AgentSpeak literature [2]. Solutions for these problems were created using both a procedural approach characteristic of traditional AgentSpeak agents, and a declarative one, in which high-level plans are omitted and left to be derived by the planning system. This switch in the method for describing agents results in a reduction of the plan description size, as it is no longer necessary to enumerate relevant combinations of lower-level plans for the agent to be able to react to different situations.

In terms of complexity the most computationally demanding part of our architecture is the planning process, which can vary significantly depending on the specific planner being used. The complexity of solving propositional planning problems depends on the number of pre-conditions and post-conditions of the operators in a certain domain [10], varying from polynomial to NP-complete and PSPACE-complete complexity. On the other hand, the conversion process into STRIPS is clearly very simple, having linear complexity on the number of pre-conditions and post-conditions of the operators being converted. The same linear complexity applies to the conversion from a STRIPS plan into an AgentSpeak plan.

Rao [2] uses a simple example agent to describe the derivations performed by an AgentSpeak interpreter. This agent detects when waste appears in a particular road lane, and disposes of it in a waste bin. The original plan library for the agent is as follows:

```
% Plan 1
+location(waste, X)
              : location(robot,X) &
                location(bin,Y)
              <- pick(waste);
                 !location(robot,Y);
                 drop(waste).
% Plan 2
+!location(robot, X)
              : location(robot,X)
              <- true.
% Plan 3
+!location(robot, X)
              : location(robot,Y) &
                not X = Y &
                adjacent(Y,Z)&
                not location(car,Z)
              <- move(Y, Z);
                 !location(robot, X).
```

Using Plan 1, whenever an agent detects waste in its current position, the agent will pick up the waste, move to the location of the waste bin and drop it. In this plan library, the agent's movement is achieved by an internal action, `move(Y,Z)`, and the agent has no way of explicitly reasoning about it. Moreover, if an agent has to perform multiple moves, recursive instantiations of Plan 3 in this library are stacked in the agent's intention structure, until the recursion stop condition is reached in Plan 2.

In order to be able to call a planner we need to modify the portion of the plan library responsible for the agent's movement (*i.e.* the last two plans) into a declarative description yielding the following plan library:

```
+location(waste, X)
       : location(robot, X) &
         location(bin, Y)
       <- pick(waste);
          +goal_conj([location(robot,Y)]);
          drop(waste).

+!move(X,Y)
       : location(robot,X) &
         not X = Y &
         not location(car,Y) &
              adjacent(X,Y)
       <- -location(robot,X);
          +location(robot,Y);
          move(X,Y).
```

The new plan library includes a description of the preconditions and effects of the `move(X,Y)` action. This is the action that is to be handled by the planning process, and the agent derives the sequence of movements required to reach the waste bin by *desiring* to be in the position of the bin. In order to specify this desire, the plan to dispose of the waste includes a step to add the desire `+goal_conj([location(robot,Y)])`, which causes the planner to be invoked. Here, the atomic

action to `move(X,Y)` is also included in the plan specification so that when `!move(X,Y)` is invoked, the agent not only updates its beliefs about the movement, but actually moves in the environment. Unlike the original plan library, however, the agent can plan its movements before starting to execute them, and will only start carrying out these actions if it has found the entire sequence of movements required to reach the desired location.

## 5 Related Work

Work on the declarative nature of goals as a means to achieve greater autonomy for an agent is being pursued by a number of researchers. Here we consider the approaches to declarative goals currently being investigated, namely those of Hübner *et al.* (Jason) [11], van Riemsdijk *et al.* [12] and Meneguzzi *et al.* [13]. There are multiple interpretations as to the requirements and properties of declarative goals for an agent interpreter, and while some models consist of an agent that performs planning from first principles whenever a goal is selected, others argue that the only crucial aspect of an architecture that handles declarative goals is the specification of target world states that can be reached using the traditional procedural approach.

### 5.1 Jason

A notion of declarative goals for AgentSpeak that takes advantage of the context part of the plans (representing the moment an implicit goal becomes relevant) was defined by Hübner *et al.* [11], and implemented in Jason [9]. More specifically, plans that share the same triggering condition refer to the achievement of the same goal, so that a goal can only be considered impossible for a given agent if all plans with the same triggering condition have been attempted and failed. In this extended AgentSpeak interpreter, these plans are modified so that the last action of every plan consists of testing for the fulfilment of the declared goal, and then the plans are grouped and executed in sequence until one finishes successfully. A plan only succeeds if at the end of its execution an agent can verify that its intended goal has been achieved. This approach retains the explicitly procedural approach to agent operation (a pre-compiled plan library describing sequences of steps that the agent can perform to accomplish its goals), only adding a more robust layer for handling plan-failure.

### 5.2 X-BDI

X-BDI [14] was the first agent model that includes a recognisably declarative goal semantics. An X-BDI agent is defined by a set of beliefs, a set of desires, and a set of operators that manipulate the world. The agent refines the set of desires through various constraints on the viability of each desire until it generates a set containing the highest priority desires that are possible and mutually consistent. During this process the agent selects the operators that will be applied to the

world in order to fulfil the selected desires in a process that is analogous to planning. The key aspect of X-BDI is that desires express *world-states* rather than triggers for the execution of pre-defined plans, leaving the composition of plans from world-changing operators to the agent interpreter.

### 5.3 Formalisations of Declarative Goals

Several researchers have worked on a family of declarative agent languages and investigated possible semantics for these languages [15, 12]. All of these languages have in common the notion that an agent is defined in terms of beliefs, goals and capabilities, which are interpreted in such a way as to select and apply capabilities in order to fulfil an agent's goals. These approaches have evolved from GOAL [15] into a declarative semantics very similar to that of X-BDI [14], in which an agent's desires express *world-states* which must be achieved by the agent selection and application of capabilities.

### 5.4 Discussion

In addition to the models described in this section, variations of the way an agent interpreter handles declarative goals have also been described. These approaches advocate the use of fast propositional planners to verify the existence of a sequence of actions that fulfil a declarative goal [13]. The planning process in this setting allows the consideration of the entire set of available operators to create new plans, providing a degree of flexibility to the agent's behaviour. Our research has not dealt with multi-agent issues so far, but the approach taken by Coo-BDI [16] to share plans between agents might provide an interesting extension to our architecture. The exchange of new plans might offset the sometimes significant time needed to create plans from scratch by allowing agents to request the help of other planning-capable agents.

The approaches in Sections 5.1 and 5.3 deal with important aspects of declarative goals in agent systems, such as the verification of accomplishment and logical properties of such systems. However, support for declarative goals in Jason still requires a designer to specify high-level plans, while the formalisms described by van Riemsdijk lack any analysis of the practicality of their implementation. Though X-BDI implements a truly declarative agent specification language, the language is very far from mainstream acceptance, and the underlying logic system used in X-BDI suffers from a stream of efficiency problems.

## 6 Concluding Remarks

In this paper we have demonstrated how the addition of a planning component can augment the capabilities of a plan library-based agent. In order to exploit the planning capability, the agent uses a special planning action to create high-level plans by composing specially designed plans within an agent's plan library. This assumes no modification in the AgentSpeak language, and allows an agent

to be defined so that *built-in* plans can still be defined for common tasks, while allowing for a degree of flexibility for the agent to act in unforseen situations. Our system can also be viewed as a way to extend the declarative goal semantics proposed by Hübner *et al.* [11], in that it allows an agent designer to specify only desired world-states and basic capabilities, relying on the planning component to form plans at runtime. Even though the idea of translating BDI states into STRIPS problems is not new [13], our idea of an encapsulated planning action allows the usage of any other planning formalism sufficiently compatible with the BDI model.

Recent approaches to the programming of agents based on declarative goals rely on mechanisms of plan selection and verification. However, we argue that a declarative model of agent programming must include not only constructs for verifying the accomplishment of an explicit world-state (which is an important capability in any declarative agent), but also a way in which an agent designer can specify *only* the world states the agent has to achieve and the description of atomic operators allowing an underlying *engine* to derive plans at runtime. In this paper we argue that propositional planning can provide one such engine, drawing on agent descriptions that include atomic actions and desired states, and leaving the derivation of actual plans for the agent at runtime.

The addition of a planning component to a BDI agent model has been recently revisited by other researchers, especially by Sardina *et al.* [17] and Walczak *et al.* [18]. The former describes a BDI programming language that incorporates Hierarchical Task Networks (HTN) planning by exploring the similarities between these two formalisms, but this approach fails to address the fact that designers must specify rules for HTN planning in the same way in which they would decompose multiple plans in a traditional BDI agent. The latter approach is based on a specially adapted planner to support the agent, preventing the model from taking advantage of novel approaches to planning.

The prototype implemented for the evaluation of the extensions described in this paper has been empirically tested for a number of small problems, but, further testing and refinement of this prototype is still required, for instance, to evaluate how interactions between the addition of new plans will affect the existing plan library. The system can also be improved in a number of ways in order to better exploit the underlying planner component. For example, the effort spent on planning can be moderated by a quantitative model of control, so that an agent can decide to spend a set amount of computational effort into the planning process before it concludes the goal is not worth pursuing. This could be implemented by changing the definition of $goal\_conj(Goals)$ to include a representation of motivational model $goal\_conj(Goals, Motivation)$, which can be used to tune the planner and set hard limits to the amount of planning effort devoted to achieving that specific desire.

As indicated above, the key contribution of this paper is a technique that allows procedural agent architectures to use state-space (and hence, declarative) planners to augment flexibility at runtime, thus leveraging advances in planning algorithms. It is important to point out that previous efforts exploring the use

of HTN planning do not change the essential procedural mode of reasoning of the corresponding agent architectures, as argued by Sardina *et al.* [17]. State-space planners operate on a declarative description of the desired goal state, and our conversion process effectively allows a designer to use an AgentSpeak-like language in a declarative way, something which previous planning architectures do not allow. Finally, we are currently working on addressing some of the limitations we have identified regarding the generation and execution of concurrent plans for multiagent scenarios.

**Acknowledgments.** The first author is supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) of the Brazilian Ministry of Education. We would like to thank Rafael Bordini and Jomi Hübner for their support regarding the programming of AgentSpeak agents in their Jason implementation, as well as the discussion of many issues regarding planning and declarative goals.

## References

1. d'Inverno, M., Luck, M.: Engineering AgentSpeak(L): A formal computational model. Journal of Logic and Computation **8**(3) (1998) 233–260
2. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In de Velde, W.V., Perram, J.W., eds.: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World. Volume 1038 of LNCS. Springer, Eindhoven, The Netherlands (1996) 42–55
3. van Riemsdijk, B., van der Hoek, W., Meyer, J.J.C.: Agent programming in dribble: from beliefs to goals using plans. In: AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, Melbourne, Australia, ACM Press (2003) 393–400
4. Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E.: Multi-Agent Programming: Languages, Platforms and Applications. Volume 15 of Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer (2005)
5. Rao, A.S., Georgeff, M.P.: BDI-agents: from theory to practice. In: Proceedings of the First International Conference on Multiagent Systems ICMAS-95, San Francisco (1995) 312–319
6. Georgeff, M.P., Ingrand, F.F.: Monitoring and control of spacecraft systems using procedural reasoning. In: Proceedings of the Space Operations and Robotics Workshop, Houston, USA (1989)
7. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research **20** (2003) 61–124
8. Fikes, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2**(3-4) (1971) 189–208
9. Bordini, R.H., Hübner, J.F., Vieira, R.: Jason and the golden fleece of agent-oriented programming. In Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E., eds.: Multi-Agent Programming: Languages, Platforms and Applications. Springer (2005) 3–37
10. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence **69**(1-2) (1994) 165–204

11. Hübner, J., Bordini, R.H., Wooldridge, M.: Programming declarative goals using plan patterns. In: Proceedings of the 2006 Workshop on Declarative Agent Languages and Technologies. (2006)
12. van Riemsdijk, M.B., Dastani, M., Meyer, J.J.C.: Semantics of declarative goals in agent programming. In: AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, Utrecht, The Netherlands, ACM Press (2005) 133–140
13. Meneguzzi, F.R., Zorzo, A.F., Móra, M.D.C.: Propositional planning in BDI agents. In: Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, ACM Press (2004) 58–63
14. Móra, M.d.C., Lopes, J.G.P., Vicari, R.M., Coelho, H.: BDI models and systems: Bridging the gap. In: Intelligent Agents V, Agent Theories, Architectures, and Languages, Fifth International Workshop, ATAL '98. Volume 1555 of LNCS. Springer, Paris, France (1999) 11–27
15. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.J.C.: Agent programming with declarative goals. In: Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop, ATAL 2000. Volume 1986 of LNCS. Springer, Boston, USA (2001) 228–243
16. Ancona, D., Mascardi, V.: Coo-BDI: Extending the BDI Model with Cooperativity. In Leite, J.A., Omicini, A., Sterling, L., Torroni, P., eds.: Proceedings of the First Declarative Agent Languages and Technologies Workshop (DALT'03), Springer-Verlag (2004) 109–134 LNAI 2990.
17. Sardina, S., de Silva, L., Padgham, L.: Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In: AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, Hakodate, Japan, ACM Press (2006) 1001–1008
18. Walczak, A., Braubach, L., Pokahr, A., Lamersdorf, W.: Augmenting BDI Agents with Deliberative Planning Techniques. In: The Fifth International Workshop on Programming Multiagent Systems (PROMAS-2006). (2006)

# Modelling Agents' Choices in Temporal Linear Logic

Duc Q. Pham, James Harland, and Michael Winikoff

School of Computer Science and Information Technology
RMIT University
GPO Box 2476V, Melbourne, 3001, Australia
{qupham,jah,winikoff}@cs.rmit.edu.au

**Abstract.** Decision-making is a fundamental feature of agent systems. Agents need to respond to requests from other agents, to react to environmental changes, and to prioritize and pursue their goals. Such decisions can have ongoing effects, as the future behavior of an agent may be heavily dependent on choices made earlier. In this paper we investigate a formal framework for modeling the choices of an agent. In particular, we show how the use of a choices calculus based on *temporal linear logic* can be used to capture distribution, temporal and dependency aspects of choices.

## 1 Introduction

Agents are increasingly becoming accepted as a suitable paradigm for conceptualizing, designing, and implementing the sorts of distributed complex dynamic systems that can be found in a range of domains, such as telecommunications, banking, crisis management, and business transactions [1].

A fundamental theme in agent systems is *decision-making*. Agents have to decide which resources to use, which actions to perform, which goals and commitments to attend to next etc. in order to fulfill their design objectives as well as to respond to other agents in an open and dynamic operating environment. Very often, agents are confronted with choices. Decisions on choices made now may very well affect future achievement of goals or other threads of interactions. In a sense, agents have to make informed and wise decisions on choices. This can be thought of as how to enable agents to act on choices, subject to any possible constraints and with a global consideration to future advantages.

Moreover, in open and dynamic environments, changes from the environment occur frequently and often are unpredictable, which can hinder accomplishment of agents' goals. How agents cope with changes remains an open and challenging problem. On the one hand, agents should be enabled to reason about the current changes and act flexibly. On the other hand, agents should be equipped with a reasoning ability to best predict changes and act accordingly.

These characteristics are desirable for a single agent. However, no agent is an island, and decisions of an agent are not made in isolation, but in the context of decisions made by other agents, as part of interactions between the agents. Thus, the challenging setting here is that in negotiation and other forms of agent interaction, decision making is *distributed*. In particular, key challenges in modeling decision making in agent interaction are:

– *Distribution:* choices are distributed among agents, and changes from the environments affect each agent in different ways. How to capture these choices, their dependencies and the effects of different strategies for their decisions as well as to reason about the global changes at the individual level in agent systems are important.
– *Time:* decision making by agents occurs in time. So do the choices to be made and the changes in the environment. Then it is necessary to deal with them in a time dependent manner.
– *Dependencies:* i.e. capturing that certain decisions depend on other decisions.

The central importance of decision-making in agent systems makes it natural to use logic as a basis for a formal framework for agents. This means that we can model the current state of an agent as a collection of formulas, and the consequences of a particular action on a given state can be explored via standard reasoning methods. In this paper, we explore how to extend this approach to include decisions as well as actions. Hence, for logic-based agents, whose reasoning and decision making is based on a declarative logical formalism, it is important to model the decision making on choices as well as on the environment changes.

This paper tackles the modeling of agent decisions in a way that allows distribution, dependencies, and time of choices to be captured. We discuss specific desirable properties of a formal model of agent choices (section 3) and then present a formal *choice calculus* (section 4). We then consider an application of the choice calculus. Specifically, by ensuring that the choices are made in multiple different formulas consistently, the choice calculus allows us to turn an interaction concerning a goal into multiple concurrent and distributed threads of interaction on its subgoals. This is also based on a mechanism to split a formula $\Gamma$ which contains $A$ into two formulas, one of which contains $A$, the other contains the results of "subtracting" $A$ from $\Gamma$.

In [2], it was shown how *Temporal Linear Logic* (TLL) can be used to model agent interactions to achieve flexibility, particularly due to its ability to model resources and choices, as well as temporal constraints. This paper can be seen as further developing this line of work to include explicit considerations of the choices of each agent and the strategies of dealing with them.

The remainder of this paper is structured as follows. Section 2 briefly reviews temporal linear logic, and the agent interaction framework. The following two sections motivate and present the choice calculus. Section 5 presents an application of the choice calculus to distributed concurrent problem solving. We then conclude in section 6.

## 2 Background

### 2.1 Temporal Linear Logic

Temporal Linear Logic (**TLL**) [3] is the result of introducing temporal logic into linear logic. While linear logic provides advantages to modeling and reasoning about resources, temporal logic addresses the description and reasoning about the changes of truth values of logic expressions over time [4]. Hence, TLL is resource-conscious as well as dealing with time.

In particular, linear logic [5] is well-known for modeling resources as well as updating processes. It has been considered in agent systems to support agent negotiation and planning by means of proof search [6, 7].

In multi-agent systems, utilization of resources and resource production and consumption processes are of fundamental consideration. In such logic as classical or temporal logic, however, a direct mapping of resources onto formulas is troublesome. If we model resources like A as "one dollar" and B as "a chocolate bar", then $A, A \Rightarrow B$ in classical logic is read as "given one dollar we can get a chocolate bar". The problem is that $A$ - one dollar - remains afterward. In order to resolve such resource - formula mapping issues, Girard proposed treating formulas as resources and hence they will be used exactly once in derivations.

As a result of such constraint, classical conjunction (and) and disjunction (or) are recast over different uses of contexts - multiplicative as combining and additive as sharing to come up with four connectives. In particular, A $\otimes$ A (*multiplicative conjunction*) means that one has two As at the same time, which is different from $A \wedge A = A$. Hence, $\otimes$ allows a natural expression of proportion. A $\parr$ B (*multiplicative disjunction*) means that if not A then B or vice versa but not both A and B.

The ability to specify choices via the additive connectives is also a particularly useful feature of linear logic. If we consider formulas on the left hand side of $\vdash$ as what are provided (program formulas), then A $\&$ B (*additive conjunction*) stands for one's own choice, either of A or B but not both. A $\oplus$ B (*additive disjunction*) stands for the possibility of either A or B, but we don't know which. In other words, while $\&$ refers to inner determinism, $\oplus$ refers to inner non-determinism. Hence, $\&$ can be used to model an agent's own choices (*internal choices*) whereas $\oplus$ can be used to model *indeterminate possibilities* (or external choices) in the environment. The duality between $\&$ and $\oplus$, being respectively an internal and an external choice, is a well-known feature of linear logic [5].

Due to the duality between formulas on two sides of $\vdash$, formulas on the right side can be regarded as goal formulas, i.e. what to be derived. A goal $A \,\&\, B$ means that after deriving this goal, one can choose between $A$ or $B$. In order to have this ability to choose, one must prepare for both cases - being able to derive $A$ and derive $B$. On the other hand, a goal $A \oplus B$ means that it is not determined which goal between $A$ and $B$. Hence, one can choose to derive either of them. In terms of deriving goals, $\&$ and $\oplus$ among goal formulas act as introducing indeterminate possibilities and introducing an internal choice respectively.

The temporal operators used are $\bigcirc$ (next), $\Box$ (anytime), and $\Diamond$ (sometime) [3]. Formulas with no temporal operators can be considered as being available only at present. Adding $\bigcirc$ to a formula A, i.e. $\bigcirc$A, means that A can be used only at the next time point and exactly once. Similarly, $\Box$A means that A can be used at any time (exactly once, since it is linear). $\Diamond A$ means that A can be at some time (also exactly once). Whilst the temporal operators have their standard meanings, the notions of internal and external choice can be applied here as well, in that in that $\Box A$ means that $A$ can be used at any time (but exactly once) with the choice of time being internal to the agent, and $\Diamond A$ means that $A$ can be used at some time with the choice of time being external to the agent.

The semantics of TLL connectives and operators as above are given via its sequent calculus, since we take a proof-theoretic approach in modeling agent interaction.

### 2.2 A Model for Agent Interaction

In [8], an interaction modeling framework which uses TLL as a means of specifying interaction protocols is used as TLL is natural to model resources, internal choices and indeterminate possibilities with respect to time. Various concepts such as resource, capability and commitment/goal are encoded in TLL. The symmetry between a formula and its negation in TLL is explored as a way to model resources and commitments/goals. In particular, formulas to be located on the left hand side of $\vdash$ can be regarded as formulas in supply (resources) while formulas to be located on the right hand side of $\vdash$ as formulas in demand (goals).

A unit of consumable *resources* is then modeled as a proposition in linear logic and can be preceded by temporal operators to address time dependency. For example, listening to music after (exactly) three time points is denoted as $\bigcirc\ \bigcirc\ \bigcirc music$. A shorthand is $\bigcirc^3 music$.

The *capabilities* of agents refer to producing, consuming, relocating and changing ownership of resources. Capabilities are represented by describing the state before and after performing them. The general representation form is $\Gamma \multimap \Delta$, in which $\Gamma$ describes the conditions before and $\Delta$ describes the conditions after. The linear implication $\multimap$ ensures that the conditions before will be transformed into the conditions after.

To take an example, consider a capability of producing music using music player to play music files. There are two options available at the agent's own choice, one is using mp3 player to play mp3 files, the other is using CD player to play CD files. The encoding is:

$$\Box[[(mp3 \otimes mp3\_player) \oplus (CD \otimes CD\_player)] \multimap music]^1$$

where $\Box$ means that the capability can be applied at any time, $\oplus$ indicates an internal choice (not $\&$, as it is located on the left hand side of $\multimap$).

## 3 Desiderata for a Choice Calculus

Unpredictable changes in the environment can be regarded as a set of possibilities which the agents do not know the outcomes. There are several strategies for dealing with unpredictable changes. A safe approach is to prepare for all the possible scenarios, at the cost of extra reservation and/or consumption of resources. Other approaches are more risky in which agents make a closest possible prediction of which possibilities to occur and act accordingly. If the predictions are correct, agents achieve the goals with resource efficiency. Here, there is a trade-off between resource efficiency and safety.

[1] In the modeling, formulas representing media players are consumed away, which does not reflect the persistence of physical objects. However, we focus on modeling how resources are utilized, not their physical existences and hence simplify the encoding since it is not necessary to have the media players retained for later use.

In contrast to indeterminate possibilities, internal choices are what agents can decide by themselves to their own advantage. Decisions on internal choices can be based on what is best for the agents' current and local needs. However, it is desirable that they consider internal choices in the context of other internal choices that have been or will be made. This requires an ability to make an informed decision on internal choices. If we put information for decision making on internal choices as constraints associated with those internal choices then what required is a modeling of internal choices with their associated constraints such that agents can reason about them and decide accordingly. Also, in such a distributed environment as multi-agent systems, such a modeling should take into account as constraints the dependencies among internal choices.

In addition, as agents act in time, decisions can be made precisely at the required time or can be well prepared in advance. When to decide and act on internal choices should be at the agents' autonomy. The advantages of deciding internal choices in advance can be seen in an example as resolving a goal of $\bigcirc^3(A \oplus B)$. This goal involves an internal choice ($\oplus$) to be determined at the third next time point ($\bigcirc^3$). If the agent decides now to choose A and commits to making the same decision at the third next time point, then from now, the agent only has to focus a goal of $\bigcirc^3 A$. This also means that resources used for other goals can be guaranteed to be exclusive from the requirements of $\bigcirc^3(A \oplus B)$, which might not be the case otherwise when $\bigcirc^3(A \oplus B)$ is decided as $\bigcirc^3 B$ at the third next time point.

The following example illustrates various desirable strategies of agents.

*Peter intends to organize an outdoor party in two days' time. He has a goal of providing music at the party. He has a CD burner and a blank CD onto which he can burn music in CD or mp3 format. His friend, John, can help by bringing a CD player or an mp3 player to the party but Peter will not know which until tomorrow. David then informs Peter that he would like to borrow Peter's CD burner today.*

In this situation, to Peter, there is an internal choice on the music format and an indeterminate possibility regarding the player. We consider two strategies. If Peter does not let David borrow the CD burner, he can wait until tomorrow to find out what kind of player John will bring to the party and choose the music format accordingly at that time. Otherwise, he can not delay burning the CD until tomorrow and so has to make a prediction on which player John will bring to the party and then decide the internal choice on the music format early (now), burn the CD and let David borrow the CD burner. The question is then how to make such strategies available for Peter to explore.

One solution is using formalisms such as logic to enable agent reasoning on those internal choices and indeterminate possibilities. Linear Logic is highly suitable here, because it allows us to distinguish between internal determinism and non-determinism. *Temporal* linear logic (TLL) further puts such modeling of them in a time dependent context. Indeed, internal choices and external choices (inner non-determinism) have been modeled previously using Linear Logic [6, 9] and TLL [8, 2].

An important observation is that although (temporal) linear logic captures the notions of internal choice and indeterminate possibility, its sequent rules constrain agents to specific strategies and make each decision on internal choices in isolation (subject only to local information). Specifically, consider the following rules of standard sequent

calculus:

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \oplus B \vdash \Delta} \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \,\&\, B, \Delta}$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \,\&\, B \vdash \Delta} \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \,\&\, B \vdash \Delta} \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \oplus B, \Delta} \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \oplus B, \Delta}$$

where for the formulas on the left hand side of $\vdash$, $\&$ remarks internal choice and $\oplus$ remarks indeterminate possibility and vice versa for the formulas on the right hand side of $\vdash$.

The first set of rules require agents to prepare for both outcomes of the indeterminate possibility. Though this strategy is safe, it demands extra and unnecessary resources and actions. Moreover, this strategy does not take into account an agents' prediction of the environment or whether it is willing to take risks.

More importantly, according to the last set of rules, the free (internal) choice agents have is determined *locally*, i.e. without a global awareness. Hence decisions on these free choices may not be optimal. In particular, if the formula $A\&B$ (on the left hand side of $\vdash$) is not used in any proof now, without this kind of local information, the decision on this internal choice becomes unguided. Hence, if there is further information about possible future goals or about dependencies on other internal choices, this information should be considered and the agent should be enabled to decide the internal choice accordingly. Moreover, the rule does not allow agents to explore the strategy of deciding internal choices in advance.

Referring to our running example, in the first strategy, Peter does not let David borrow the CD burner, and so Peter can then find a proof using standard sequent rules to achieve the goal of providing music at the party two days later. However, in the second strategy, the search using standard TLL sequent calculus for a proof of the goal fails as it requires to have music in both formats (mp3 and CD) so as to be matched with the future possibility of the media player.

Hence, in this paper, we investigate how TLL not only allows us to model the difference between internal choice and indeterminate possibility with respect to time, but also allows us to capture *dependencies* among internal choices, constraints on how internal choices can be made as well as predictions and decisions of indeterminate possibilities. Such constraints may also reflect global consideration of other goals and other threads of interaction. We further consider strategies that can be used to deal with internal choices with respect to time, reflecting how cautious the agents are and whether the agents deal with them in advance. However, we will not discuss how agents can predict the environment outcomes correctly.

## 4 A Choice Calculus

If we assume that the order of operants is unchanged throughout the process of formulas manipulation, in other words, ignoring the commutative property of $\oplus$ and $\&$, then the decision on choices and indeterminate possibilities can be regarded as selecting the left hand side or the right hand side of the connective. For simplicity, we shall refer to both internal choices (choices with inner determinism) and indeterminate possibilities (choices with non-determinism) simply as choices.

As the decision on a choice is revealed at the time point associated with the choice, before that time point, the decision of the choice is unknown. We encode the decisions on choices using TLL constants.

We need to consider the base values for choice decisions, how to specify choices of the same decisions, choices that are dependent on other choices and also how standard sequent calculus rules are modified to reflect decisions on choices.

We use the notation $\overset{\&_x}{\hookrightarrow}$ or $\overset{\oplus_x}{\hookrightarrow}$ to indicate the result of the decision making of $\&_x$ and $\oplus_x$ respectively. The subscript indicates the ID of the connective. The base values for their decision can be encoded by TLL constants $L$, and $R$. For example, the result of the decision on the choice in $A \,\&\, B$ is $L$ if $A$ results from $A \,\&\, B$ and is $R$ if $B$ results. For internal choices, their decisions can be regarded as variables as agents can decide the assignment of values. Formally, we write $\vdash \overset{\&_1}{\hookrightarrow} \multimap L$ or $\overset{\&_1}{\hookrightarrow} \vdash L$ to denote that the left subformula of $\&_1$ was selected.

Decisions on indeterminate possibilities could also be represented as variables. However, we will explicitly represent that agents can not decide the outcomes of indeterminate possibilities and that the decisions on them (by external factors) have not been made by using the form $L \oplus R$ (or $L \,\&\, R$). For example, given the indeterminate possibility $\bigcirc^n(A \oplus_x B)$, we represent their decision by $\bigcirc^n(L \oplus_x R)$, where $n$ is the time point associated with the choice and $\oplus_x$ is the same connective as that in $\bigcirc^n(A \oplus_x B)$.

By modeling the choices explicitly, we can state constraints between them. For example, if two choices, $\overset{\&_x}{\hookrightarrow}$ and $\overset{\&_y}{\hookrightarrow}$, need to be made consistently — either both right or both left — then this can be stated as $\overset{\&_x}{\hookrightarrow} = \overset{\&_y}{\hookrightarrow}$ or, in logic encoding, $\overset{\&_x}{\hookrightarrow} \vdash \overset{\&_y}{\hookrightarrow}$, $\overset{\&_x}{\hookrightarrow} \dashv \overset{\&_y}{\hookrightarrow}$. More generally, we can state that a given choice $\&_x$ should depend on a combination of other choices or some external constraints. We use $condL_x$ (respectively $condR_x$) to denote the condition that should hold for the left (respectively right) side of the choice to be taken. Clearly, $condL_x$ and $condR_x$ should always be mutually exclusive. These conditions, in their presence, completely determine the results of the choices' decisions. In their absence, the internal choices become truly free choices and we are getting back to the normal case as of standard sequent rules. These conditions are encoded as TLL sequents so that sub-conditions (sequents) can be found via proof search.

Given a formula $\Gamma$ which contains a sub-formula $A$, we can compress the sequence of decisions that need to be made in order to obtain $A$ from $\Gamma$ into a single *representative choice*. For example, if $\Gamma = B \,\&_1\, \bigcirc^a(\bigcirc^b(A \oplus_2 C) \,\&_3\, D)$ then, in order to obtain $A$ from $\Gamma$ we need to decide on the right side of $\&_1$, then, $a$ time units later, decide on the left side of $\&_3$, and $b$ time units after that, have the left side of $\oplus_2$ be selected by the environment (an indeterminate possibility). Formally, the notion of representative is defined as below.

**Definition 1.** *A **representative** choice $\&_r$ with respect to a formula A in a compound program formula (respectively goal formula) $\Gamma$ is a choice $\bigcirc^x A \,\&_r\, \bigcirc^y 1$ (respectively $\bigcirc^x A \oplus_r \bigcirc^y 1$) whose decision is $L$ if A is chosen from $\Gamma$ and is $R$ otherwise, where $x, x \geq 0$ is the time associated with A in $\Gamma$ and $y, y \geq 0$ is is the time point associated with* 1.

Note that at the time of representing the choice $\&_r$ (or $\oplus_r$), the value of $y$ is not known. It will be known after all the decisions of internal choices and indeterminate possibilities in $\Gamma$ are revealed.

In the previous example, such a sequence of decisions on internal choices and indeterminate possibilities on $\Gamma$ to obtain $A$ can be captured by the sequent:
$\vdash (\overset{\&_1}{\hookrightarrow} \multimap R) \otimes \bigcirc^a (\overset{\&_3}{\hookrightarrow} \multimap L) \otimes \bigcirc^{a+b} (\overset{\oplus_2}{\hookrightarrow} \multimap L)$.
This is the determining condition for $A$ to be obtained from $\Gamma$. Observe that we can compress $\Gamma$ into a representative choice for $A$ of the form $\bigcirc^{a+b} A \,\&_r\, \bigcirc^y)$ such that the choice $\&_r$ is decided left if $\bigcirc^{a+b} A$ results from $F$ and is decided right otherwise. The condition above then corresponds to $condL_r$ of $\&_r$. As being mutually exclusive, $condR_r$ is captured as:
$\vdash (\overset{\&_1}{\hookrightarrow} \multimap L) \oplus \bigcirc^a (\overset{\&_3}{\hookrightarrow} \multimap R) \oplus \bigcirc^{a+b} (\overset{\oplus_2}{\hookrightarrow} \multimap R)$.

We now come to determine sequent calculus rules for various strategies on choices.

### 4.1 Extended Sequent Calculus

We take a general assumption that regarding indeterminate possibilities, the environment (or external factors) determines the outcomes of the possibilities at their associated times. For example, given the indeterminate possibility $\bigcirc^4 (A \oplus B)$, after four time points, the environment determines the possibility such that $\bigcirc^4 (A \oplus B)$ becomes $\bigcirc^4 A$ or $\bigcirc^4 B$ and only at this time, the outcome becomes known to agents. This assumption is based on the inherent modeling of TLL that formulas denoted at a specific time point last only in that time point.

The standard sequent calculus rules for indeterminate possibilities, which demand that agents prepare for all possibilities, correspond to a safe approach. However, if the agent chooses a strategy of risk taking, it then makes predictions of the outcome of the indeterminate possibility and follows the search path corresponding to the predicted one. The sequent rules for such a strategy are ($\vdash_{cc}$ means $\vdash$ in choice calculus context):

$$\frac{\Gamma \vdash_{cc} F, \Delta \quad [L \vdash L \,\&_n\, R]}{\Gamma \vdash_{cc} F \,\&_n\, G, \Delta} \qquad \frac{\Gamma \vdash_{cc} G, \Delta \quad [R \vdash L \,\&_n\, R]}{\Gamma \vdash_{cc} F \,\&_n\, G, \Delta}$$

$$\frac{\Gamma, F \vdash_{cc} \Delta \quad [L \oplus_n R \vdash L]}{\Gamma, F \oplus_n G \vdash_{cc} \Delta} \qquad \frac{\Gamma, G \vdash_{cc} \Delta \quad [L \oplus_n R \vdash R]}{\Gamma, F \oplus_n G \vdash_{cc} \Delta}$$

where formulas in square brackets are the conditions (predictions) and those outside brackets are the main formulas. The conditions are evaluated independently from the main formulas and at the time associated with the indeterminate possibility, when the environment reveals the outcomes. If there is a proof of the main formulas, and if the conditions are also satisfied, then the proof search is successful. If the conditions can not be satisfied even though there is a proof among the main formulas, then the search for proof fails on this branch associated with the conditions.

Moreover, if the agent further decides upon its prediction of an indeterminate possibility before the time associated with the possibility, it also can bring out the possibility's outcome earlier in the search:

$$\frac{\Gamma \vdash_{cc} \bigcirc^x F, \Delta \quad [L \vdash L \,\&_n\, R]}{\Gamma \vdash_{cc} \bigcirc^x (F \,\&_n\, G), \Delta} \qquad \frac{\Gamma \vdash_{cc} \bigcirc^x G, \Delta \quad [R \vdash L \,\&_n\, R]}{\Gamma \vdash_{cc} \bigcirc^x (F \,\&_n\, G), \Delta}$$

$$\frac{\Gamma, \bigcirc^x F \vdash_{cc} \Delta \quad [L \oplus_n R \vdash L]}{\Gamma, \bigcirc^x (F \oplus_n G) \vdash_{cc} \Delta} \qquad \frac{\Gamma, \bigcirc^x G \vdash_{cc} \Delta \quad [L \oplus_n R \vdash R]}{\Gamma, \bigcirc^x (F \oplus_n G) \vdash_{cc} \Delta}$$

Internal choices are decided by the owner agent at the time associated with the choice, subject to any constraints ($condL_n$ or $condR_n$) imposed on them. The following sequent rules reflect that:

$$\frac{\Gamma, F \vdash_{cc} \Delta \quad (\vdash condL_n)}{\Gamma, F \,\&_n\, G \vdash_{cc} \Delta} \qquad \frac{\Gamma, G \vdash_{cc} \Delta \quad (\vdash condR_n)}{\Gamma, F \,\&_n\, G \vdash_{cc} \Delta}$$

$$\frac{\Gamma \vdash_{cc} F, \Delta \quad (\vdash condL_n)}{\Gamma \vdash_{cc} F \oplus_n G, \Delta} \qquad \frac{\Gamma \vdash_{cc} G, \Delta \quad (\vdash condR_n)}{\Gamma \vdash_{cc} F \oplus_n G, \Delta}$$

where $condL_n$ and $condR_n$ are conditions imposed on the internal choice $n$ for the choice to be decided left or right. These conditions may or may not be present.

Moreover, if the agent is to decide the choice priorly, it can bring out the choice's outcome earlier in the search:

$$\frac{\Gamma, \bigcirc^x F \vdash_{cc} \Delta \quad (\vdash condL_n)}{\Gamma, \bigcirc^x (F \,\&_n\, G) \vdash_{cc} \Delta} \qquad \frac{\Gamma, \bigcirc^x G \vdash_{cc} \bigcirc^x \Delta \quad (\vdash condR_n)}{\Gamma, \bigcirc^x (F \,\&_n\, G) \vdash_{cc} \bigcirc^x \Delta}$$

$$\frac{\Gamma \vdash_{cc} \bigcirc^x F, \Delta \quad (\vdash condL_n)}{\Gamma \vdash_{cc} \bigcirc^x (F \oplus_n G), \Delta} \qquad \frac{\Gamma \vdash_{cc} G, \bigcirc^x \Delta \quad (\vdash condR_n)}{\Gamma \vdash_{cc} \bigcirc^x (F \oplus_n G), \Delta}$$

These above sequent rules, together with standard TLL sequent rules, form the *choice calculus*.

Considering our running example, recall that if Peter is to let David borrow the CD burner now, he needs to decide on the music format (the internal choice $\&_1$) now. This involves making a prediction on the player that John will possibly bring. For instance, Peter predicts that John will provide an mp3 player (i.e. $L \oplus_3 R \vdash L$). Using the choice calculus, this is captured by the following inference:

$$\frac{\Gamma, \bigcirc^2 mp3\_player \vdash_{cc} \bigcirc^2 music \quad [L \oplus_3 R \vdash L]}{\Gamma, \bigcirc(\bigcirc mp3\_player \oplus_3 \bigcirc CD\_player) \vdash_{cc} \bigcirc^2 music}$$

Based on this prediction, agent Peter decides early on the choice of music format $\&_1$ (mp3 format now) and burns the blank CD accordingly. By taking this risk on the prediction, agent Peter then successfully obtains a proof of $\bigcirc^2 music$ (given below). If the prediction $L \oplus_3 R \vdash L$ is provable at the next two days, then the goal is achieved.

For the purposes of presenting the proof we make the following abbreviations.

Let $B$ (for "Burn") denote the formula

$$\Box[Blank\_CD \otimes CD\_Burner \multimap CD\_Burner \otimes (\Box mp3 \,\&_1\, \Box CD)]$$

i.e. one can convert a blank CD to either an mp3 or music format CD (internal choice of which).

Let $P$ (for "Play") denote the formula

$$\Box[[(mp3 \otimes mp3\_player) \oplus_2 (CD \otimes CD\_player)] \multimap music]$$

i.e. at any time, either using mp3 player on mp3 music or CD player on a CD, one can produce music (the choice $\oplus_2$ here is internal).

Let $R$ (for "Resources") denote the formula

$$\Box Blank\_CD \otimes \Box CD\_Burner.$$

Let $J$ (for "John", i.e. the music player that John will provide) denote the formula

$$\bigcirc[\bigcirc mp3\_player \oplus_3 \bigcirc CD\_player]$$

i.e. either an mp3 player or CD player will be provided after two days. $\oplus_3$ is an indeterminate possibility to Peter and will be revealed tomorrow.

We also abbreviate $music$ to $m$, and $player$ to $p$, e.g. $mp3\_player$ becomes $mp3p$, then we have the following proof of $\bigcirc^2 music$ where some inferences combine a number of rule applications, and where (for space reasons) we have left out giving the CD burner to David at the rule marked "$\otimes, \Box, \multimap$". As there is no imposed condition for $\&_1$ ($condL_1 = 1$), it is omitted in the proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{mp3 \vdash mp3} \quad \dfrac{}{mp3p \vdash mp3p}\otimes}{mp3, mp3p \vdash mp3 \otimes mp3p}\otimes}{mp3, mp3p \vdash (mp3 \otimes mp3p) \oplus_2 (cd \otimes cdp)}\oplus_2}{\Box mp3, P, \bigcirc^2 mp3p \vdash \bigcirc^2 m}\bigcirc^2, \Box, \multimap}{\Box mp3 \,\&_1\, \Box cd, P, \bigcirc^2 mp3p \vdash \bigcirc^2 m}\&_1}{R, P, B, \bigcirc^2 mp3p \vdash \bigcirc^2 m}\otimes, \Box, \multimap \qquad [L \oplus_3 R \vdash L]}{R, F, P, B \vdash \bigcirc^2 m}\bigcirc\oplus_3$$

In this example we begin (bottom-most inference) by making an "in-advance" decision of the choice $\oplus_3$, specifically we predict that John will provide an MP3 player. We then use standard TLL sequent rules to burn an MP3 format CD. When the time comes to make a decision for $\oplus_2$ we can select to use the MP3 player to produce music. As can be seen from the example, internal choices and indeterminate possibilities are properly modeled with respect to time. Moreover, several strategies are enabled at agent Peter due to the use of choice calculus. If Peter is to take a safe approach, he should delay deciding the music format until tomorrow and ignores David's request. If Peter is willing to take risks, he can predict the indeterminate possibility of which player John will bring to the party and act accordingly. Peter can also decide the choice on music early so as to lend David the CD burner.

Hence, these sequent calculus rules are in place to equip agents with various strategies for reasoning to deal with indeterminate possibilities and internal choices. These strategies make it more flexible to deal with changes and handle exceptions with global awareness and dependencies among choices. In the next section, we explore an application of such modeling of choices and their coping strategies, especially dependencies among choices, to distributed problem solving in a flexible interaction modeling TLL framework [8]. But first, we show that proofs using the additional rules are, in a sense, equivalent to proofs in the original TLL sequent calculus.

The intuition behind the soundness and completeness properties of proofs using these additional rules with respect to proofs which only use original TLL sequent calculus is that eventually indeterminate possibilities like between $A$ and $B$ will be revealed as the outcome turns out to be one of the two. The soundness and completeness properties are then evaluated and proved in this context. In particular, we introduce the concept of a revealed proof, which is a proof in which all the internal choices and possibilities are revealed and replaced by the actual respective outcomes. As a result of such

replacements, all of the additional rules added in our choice calculus collapse to sequents, leaving only the standard TLL rules. Note that the proofs using choice calculus require that all the assumptions will turn out to be correct. Clearly, if the assumptions turn out to be unfounded, then the proofs are not valid.

**Definition 2.** *The **revealed proof** corresponding to a given proof of $\Gamma \vdash \Delta$ is the proof resulting from replacing all occurrences of choices with the actual outcomes of these choices. That is, any formula $F \oplus G$ corresponding to an indeterminate possibility is replaced by either $F$ or $G$, corresponding to the decision that was made by the environment; and any formula $F \,\&\, G$ corresponding to an internal choice is replaced by either $F$ or $G$, corresponding to the choice that was made by the agent.*

**Theorem 1 (Soundness).**

*A revealed proof of a proof using the TLL sequent rules augmented with the additional choice calculus rules is a valid proof under standard TLL sequent calculus rules.*
***Proof sketch:*** *All of the additional rules introduced by the choice calculus disappear when the proof is made into a revealed proof. For example, consider the rules (on the left) which are replaced in a revealed proof, where $F \,\&\, G$ is replaced by $F$, by the identities on the right.*

$$\frac{\Gamma \vdash_{cc} F, \Delta \quad [L \vdash L \,\&_n\, R]}{\Gamma \vdash_{cc} F \,\&_n\, G, \Delta} \qquad \frac{\Gamma \vdash F, \Delta}{\Gamma \vdash F, \Delta}$$

$$\frac{\Gamma \vdash_{cc} \bigcirc^x F, \Delta \quad [L \vdash L \,\&_n\, R]}{\Gamma \vdash_{cc} \bigcirc^x (F \,\&_n\, G), \Delta} \qquad \frac{\Gamma \vdash \bigcirc^x F, \Delta}{\Gamma \vdash \bigcirc^x F, \Delta}$$

$$\frac{\Gamma, F \vdash_{cc} \Delta}{\Gamma, F \,\&_n\, G \vdash_{cc} \Delta} \qquad \frac{\Gamma, F \vdash \Delta}{\Gamma, F \vdash \Delta}$$

As a result of this theorem, it then becomes that a proof under choice calculus is sound if the assumptions (predictions) it relies on are correct.

Moreover, as choice calculus also contains standard TLL sequent calculus rules, the completeness property holds trivially.

**Theorem 2 (Completeness).** *A proof using standard TLL sequent calculus rules is also a proof under choice calculus.*

## 5 Splitting a Formula

Interaction between agents is often necessary for the achievement of their goals. In the above example with Peter and John, if Peter had a CD player of his own, he would not need to interact with John in order to have music at the party. In general, it will be necessary for an agent to co-ordinate interaction with many different agents, the precise number and identity of which may not be known in advance. In order to achieve this, in this section we investigate a mechanism for partial achievement of a goal. In particular, this is a process of decomposing a given TLL goal formula into concurrent subgoals.

For example, assume that Peter now has the additional goal of having either Chinese or Thai food at the party. Deriving which goal - Chinese food (abbreviated as $C$) or Thai food (abbreviated as $T$) - is an internal choice ($\oplus_3$). Peter's goal is then

$$CD\_Burner \otimes \bigcirc^2[music \otimes (C \oplus_3 T)]$$

However, Peter can not provide food, but his friends, Ming and Chaeng, can make Chinese food and Thai food respectively. Hence, this goal can not be fulfilled by Peter alone but involves interaction with John and David as above and also Ming or Chaeng. If this goal is sent as a request to any one of them, none would be able to fulfill the goal in its entirety. Hence, it is important that the goal can be split up and achieved partially via concurrent threads of interaction. In this case, we would split this into the sub-goal $CD\_Burner \otimes \bigcirc^2 music$, which is processed as above, the sub-goal $\bigcirc^2 C \oplus_4 \bigcirc^2 1$, which is sent as a request to Ming, and the sub-goal $\bigcirc^2 1 \oplus_4 \bigcirc^2 T$, which is sent as a request to Chaeng. The choice $\oplus_4$ will be later determined consistently with $\oplus_3$.

Hence we need to be able to split a goal into sub-goals, and to keep track of which parts have been achieved. In particular, it is useful to isolate a sub-goal from the rest of the goal. We do this by taking the overall formula $\Gamma$ and separating from it a particular sub-formula $A$. We show how this can be done on the fragment which contains the connectives $\otimes, \oplus, \&, \bigcirc$.

The split-ups of a formula $\Gamma$ with respect to the formula A that $\Gamma$ contains are the two formulas $\widehat{\Gamma - A}$ and $\widehat{A}$, which are defined below.

$\widehat{\Gamma - A}$ is the formula $\Gamma$ which has undergone a single removal or substitution of (one occurrence of) A by 1 while the rest is kept unchanged. Specifically, where A resides in the structure of $\Gamma$, the following mapping is applied to A and its directly connected formulas $\Delta$. $\Delta$ is any TLL formula and $x \geq 0$.

1. $A \mapsto 1$
2. $\bigcirc^x A \mapsto \bigcirc^x 1$
3. $\bigcirc^x A \; op \; \Delta \mapsto \bigcirc^x 1 \; op \; \Delta$ for $op \in \{\otimes, \&, \oplus\}$

We also apply the equivalence $1 \otimes \Delta \equiv \Delta$, so that $\bigcirc^x A \otimes \Delta \mapsto \Delta$.

The formula $\widehat{A}$ is determined recursively according to structure of $\Gamma$ as below, by examining the structure of $\Gamma$:

- If $\Gamma^1 = \bigcirc^x A$, then $\widehat{A^1} = \bigcirc^x A$
- If $\Gamma^1 = \bigcirc^x A \; op_m \; \Delta$, then $\widehat{A^1} = \bigcirc^x A \; op_m \; 1$
- If $\Gamma^n = \bigcirc^x \Gamma^{n-1}$, then $\widehat{A^n} = \bigcirc^x \widehat{A^{(n-1)}}$
- If $\Gamma^n = \Gamma^{n-1} \; op_n \; \Delta$, then $\widehat{A^n} = \widehat{A^{n-1}} \; op_n \; 1$

where $\Gamma^i, \Delta$ are formulas of the fragment and $\Gamma^i$ contains A. $op_n, op_m \in \{\otimes, \&, \oplus\}$ and $n, m$ are the IDs. We also again apply the equivalence $1 \otimes \Delta \equiv \Delta$, so that when $\Gamma^1 = \bigcirc^x A \otimes \Delta$, then $\widehat{A^1} = \bigcirc^x A$.

Another view is that $\widehat{A}$ is obtained by recursively replacing formulas that rest on the other side of connective (to the formula that contains A) by 1 if the connective is $\oplus$ or $\&$ and remove them if the connective is $\otimes$.

It can be seen from the formulation of $\widehat{\Gamma - A}$ and $\widehat{A}$ that there are requirements of choice dependencies among the split ups. Indeed, all the corresponding choices and

possibilities in them must be consistent. In particular, decisions made on the corresponding choices and possibilities in $\widehat{\Gamma - A}$, and $\widehat{A}$ should be the same as those that would have been made on the corresponding ones in $\Gamma$. Indeed, if A is ever resulted from $\Gamma$ as a result of a sequence of choices and possibilities in $\Gamma$ being decided, then those decisions also make $\widehat{A}$ become A.

As an example, we return to our running example and consider Peter's goal formula. The goal $G = CD\_Burner \otimes \bigcirc^2[music \otimes (C \oplus_3 T)]$ can be split into:
$[\widehat{G - C}] = CD\_Burner \otimes \bigcirc^2[music \otimes (1 \oplus_3 T)]$ and $\widehat{C} = \bigcirc^2(C \oplus_3 1)$.
Subsequently, $\widehat{G - C}$ can be split into:
$[\widehat{G - C - T}] = CD\_Burner \otimes \bigcirc^2 music$ and $\widehat{T}$ of $\widehat{G - C}$ is $\bigcirc^2(1 \oplus_3 T)$.

Indeed, $\widehat{A}$ can result in $\bigcirc^x A$ or $\bigcirc^y 1$, $x, y \geq 0$, as a result of having all the choices in $\widehat{A}$ decided. In the following theorem, we show that $\widehat{A}$ can be compressed into a representative choice (of A in $\widehat{A}$) of the form $\bigcirc^x A \,\&_r\, \bigcirc^y 1$ if $\widehat{A}$ is a program formula, or $\bigcirc^x A \oplus_r \bigcirc^y 1$ if $\widehat{A}$ is a goal formula.

**Theorem 3.** *$\widehat{A} \vdash_{cc} \bigcirc^x A \,\&_r\, \bigcirc^y 1$ if $\widehat{A}$ is a program formula, and $\widehat{A} \vdash_{cc} \bigcirc^x (A \oplus_r 1)$ if $\widehat{A}$ is a goal formula, where $x, x \geq 0$ is the time associated with the occurrence of A in $\widehat{A}$ and for some value of $y \geq 0$. Additionally, $\bigcirc^x A \,\&_r\, \bigcirc^y 1 \vdash_{cc} \widehat{A}$ and $\bigcirc^x (A \oplus_r 1) \vdash_{cc} \widehat{A}$ (proof omitted).*

***Proof**: by induction on the structure of $\widehat{A}$. We highlight a few cases of the proof for $\widehat{A} \vdash_{cc} \bigcirc^x A \,\&_r\, \bigcirc^y 1$. The others are similar.*

***Base step**: $\widehat{A} = A$, hence $x = 0, condL_r = 1$. The choice is decided left and we have $A \vdash_{cc} A$.*

***Induction step**: Assume the hypothesis is true for $n$, so that $\widehat{A^n} \vdash_{cc} \bigcirc^n A \,\&_n\, \bigcirc^y 1$ is provable, which means the following (upper) sequents are also provable:*

$$\frac{\widehat{A^n} \vdash_{cc} \bigcirc^n A\ [\vdash condL_n]}{\widehat{A^n} \vdash_{cc} \bigcirc^n A \,\&_n\, \bigcirc^y 1}\ \&_n \qquad \frac{\widehat{A^n} \vdash_{cc} \bigcirc^y 1\ [\vdash condR_n]}{\widehat{A^n} \vdash_{cc} \bigcirc^n A \,\&_n\, \bigcirc^y 1}\ \&_n$$

*We show the case for $\widehat{A^{n+1}} = \widehat{A^n} \&_1 1$ below, the others $\bigcirc\widehat{A^n}$, and $\widehat{A^n} \oplus_2 1$ are similar. In this case, we need to prove $\widehat{A^n} \,\&_1\, 1 \vdash_{cc} \bigcirc^n A \,\&_{n+1}\, \bigcirc^y 1$, where $condL_{n+1} = condL_n \otimes (\overset{\&_1}{\hookrightarrow} L)$; and $condR_{n+1} = condR_n \oplus (\overset{\&_1}{\hookrightarrow} R)$*

$$\frac{\dfrac{\widehat{A^n} \vdash_{cc} \bigcirc^n A\ [\vdash condL_n]}{\widehat{A^n} \,\&_1\, 1 \vdash_{cc} \bigcirc^n A\ [\vdash condL_n \otimes (\overset{\&_1}{\hookrightarrow} L)]}\ \&_{n+1}}{\widehat{A^n} \,\&_1\, 1 \vdash_{cc} \bigcirc^n A \,\&_{n+1}\, \bigcirc^y 1}\ \&_1 \qquad \frac{\dfrac{\widehat{A^n} \vdash_{cc} \bigcirc^y 1\ [\vdash condR_n] \quad \dfrac{1 \vdash_{cc} 1(y=0)}{1 \vdash_{cc} \bigcirc^y 1}}{\widehat{A^n} \,\&_1\, 1 \vdash_{cc} \bigcirc^y 1\ [\vdash condR_n \oplus (\overset{\&_1}{\hookrightarrow} R)]}\ \&_{n+1}}{\widehat{A^n} \,\&_1\, 1 \vdash_{cc} \bigcirc^n A \,\&_{n+1}\, \bigcirc^y 1}\ \&_1$$

*where the value of y is assigned as appropriately in the proof. Both cases of the decision on $\&_1$ are proved.*

Applying this theorem to the above example, we can obtain further results:
$\widehat{C} = \bigcirc^2(C \oplus_3 1) = \bigcirc^2 C \oplus_4 \bigcirc^2 1$,
$\widehat{T}$ (of $\widehat{G - C}$) $= \bigcirc^2(1 \oplus_3 T) = \bigcirc^2 1 \oplus_4 \bigcirc^2 T$, where $\oplus_4$ is the representative choice and is of the same decision as $\oplus_3$ at the next two time points.

The equivalence relationship between $\Gamma$ and its split ups, $\widehat{\Gamma - A}$ and $\widehat{A}$, is established by the following theorems.

**Theorem 4.** $\widehat{A}, \widehat{\Gamma - A} \vdash_{cc} \Gamma$.
*(From the multiplicative conjunction of the split ups of $\Gamma$ via $A$ — $\widehat{A}, \widehat{\Gamma - A}$ — we can derive $\Gamma$).*
***Proof (sketch)**: by induction on the structure of $\Gamma$. We highlight a few cases of the proof. The others are similar.*

***Base step**:*
***Case** $\Gamma = A \oplus_1 \Delta$. We need to prove $A \oplus_1 1, 1 \oplus_1 \Delta \vdash_{cc} A \oplus_1 \Delta$. Both choices for $\oplus_1$ fulfill this, as below.*

$$\frac{\dfrac{A \vdash_{cc} A}{A, 1 \vdash_{cc} A}}{A \oplus_1 1, 1 \oplus_1 \Delta \vdash_{cc} A \oplus_1 \Delta} \oplus R \quad \frac{\dfrac{\Delta \vdash_{cc} \Delta}{1, \Delta \vdash_{cc} \Delta}}{A \oplus_1 1, 1 \oplus_1 \Delta \vdash_{cc} A \oplus_1 \Delta} \oplus R$$

***Induction step**: Assume the hypothesis is true for $n$, so that $\widehat{A^n}, \widehat{[\Gamma - A]^n} \vdash_{cc} \Gamma^n$. We need to prove that this holds for $n+1$. We show the case for $\Gamma^{n+1} = \Gamma^n \&_1 \Delta$ below; the others ($\bigcirc^x \Gamma^n$, $\Gamma^n \otimes \Delta$ and $\Gamma^n \oplus_2 \Delta$) are all similar. In this case we have $\widehat{[\Gamma - A]^{n+1}} = \widehat{[\Gamma - A]^n} \&_1 \Delta$, and $\widehat{A^{n+1}} = \widehat{A^n} \&_1 1$.*

$$\frac{\dfrac{[R \vdash L \&_1 R]}{\widehat{A^n}, \widehat{[\Gamma - A]^n} \vdash_{cc} \Gamma^n \; [R \vdash L \&_1 R]}}{\widehat{A^n} \&_1 1, \widehat{[\Gamma - A]^n} \&_1 \Delta \vdash_{cc} \Gamma^n \&_1 \Delta} \&R \quad \frac{\dfrac{\dfrac{[L \vdash L \&_1 R]}{\Delta \vdash_{cc} \Delta \; [L \vdash L \&_1 R]}}{1, \Delta \vdash_{cc} \Delta \; [L \vdash L \&_1 R]}}{\widehat{A^n} \&_1 1, \widehat{[\Gamma - A]^n} \&_1 \Delta \vdash_{cc} \Gamma^n \&_1 \Delta} \&R$$

*Hence, both cases of the decision on $\&_1$ are proved.*

One further point to note is the use of $\bot$. In our modeling context, $\bot$ does not produce any resource nor consume any other resource. We make an assumption that $\bot$ can be removed from agents' states of resources. This is formalized as a new axiom:

$$\frac{}{\Gamma, \bot \Vdash_{cc} \Gamma}$$

where $\Vdash_{cc}$ denotes $\vdash_{cc}$ under this assumption. Based on this assumption, we derive

**Theorem 5.** $\Gamma, \widehat{A^\bot}, \widehat{A} \Vdash_{cc} \widehat{\Gamma - A} \otimes \widehat{A}$.
*That is, from $\Gamma$ and its split up on A, $\widehat{A}$, as well as $\widehat{A^\bot}$ with the same structure as of $\widehat{A}$, we can derive a multiplicative conjunction of its split ups $\widehat{A}$ and $\widehat{\Gamma - A}$.*
***Proof (sketch)**: by induction on the structure of $\Gamma$, where $\widehat{A^\bot}$ is obtained from $\widehat{A}$ by replacing the single copy of A by $A^\bot$. The proof can be obtained similarly from the proof of theorem 4 and is omitted here for space reason.*

Hence, $\Gamma, \widehat{A^\bot}, \widehat{A} \Vdash_{cc} \widehat{\Gamma - A} \otimes \widehat{A} \vdash_{cc} \Gamma$.

As $\widehat{A}, \widehat{A^\bot} \vdash_{cc} \bot$, in terms of resources, the concurrent presence of both $\widehat{A}$ and its consumption $\widehat{A^\bot}$ does not consume any resource nor produce any. Hence, the presence of both does not make any effect and hence can be ignored. In terms of resources, using $\Gamma$, one can derive $\widehat{\Gamma - A} \otimes \widehat{A}$.

Theorems 4 and 5 lay important foundation of splitting up resources and goals in agent interaction. Particularly, if a goal $\Gamma$ contains a formula A that the current interaction can derive, then $\Gamma$ can be split into $\widehat{A}$ and $\widehat{\Gamma - A}$. If A is ever chosen in $\Gamma$, then the

goal $\widehat{A}$ becomes a goal of A which can be achieved immediately by the current interaction. Similarly, if a resource $\Gamma$, which contains A, is available for use in an interaction that only uses A than the resource $\Gamma$ can be split into two resources $\widehat{\Gamma - A}$ and $\widehat{A}$, of which $\widehat{A}$ can be used right away if A is ever chosen in $\Gamma$.

Returning to our example, the above theorems can be applied so that Peter can turn its goal into concurrent sub-goals $CD\_Burner \otimes \bigcirc^2 music \otimes (\bigcirc^2 C \oplus_4 \bigcirc^2 1) \otimes (\bigcirc^2 1 \oplus_4 \bigcirc^2 T)$, where the decision on $\oplus_4$ now is the same as that of $\oplus_3$ at the next two days. Therefore, agent Peter can achieve the two sub-goals $CD\_Burner \otimes \bigcirc^2 music$ as above and sends the subgoal $(\bigcirc^2 C \oplus_4 \bigcirc^2 1)$ as a request to Ming and the subgoal $(\bigcirc^2 1 \oplus_4 \bigcirc^2 T)$ as a request to Chaeng.

If Ming makes Chinese food, then $\bigcirc^2 C\ [\overset{\oplus_4}{\hookrightarrow}\vdash L]$ is resulted. As the choice $\oplus_4$ is decided left, the other subgoal $(\bigcirc^2 1 \oplus_4 \bigcirc^2 T)$ becomes $\bigcirc^2 1$, which is also readily achievable. If Ming does not make Chinese food, there is a proof of $\bigcirc^2 1$, where $[\overset{\oplus_4}{\hookrightarrow}\vdash R]$. This decision on the choice $\oplus_4$ (choosing right) makes the subgoal $(\bigcirc^2 1 \oplus_4 \bigcirc^2 T)$ becomes $\bigcirc^2 T$. Thus, if all the subgoals are successful, this mechanism ensures that only one kind of food is made.

Hence, such splitting up of formulas allows Peter to concurrently and partially achieve its goal via different threads of interaction.

## 6 Discussion and Conclusion

The paper addresses issues in agents' decision making when it comes to agents' choices and indeterminate possibilities in a distributed environment. A modeling of internal choices and indeterminate possibilities as well as their decisions is presented via choice calculus. The modeling supports decisions across time, decisions based on predictions of changes in the environment, as well as dependencies and distribution among choices with respect to time.

Temporal linear logic has been used in our modeling due to its natural role in supporting agent planning in concurrent and resource-conscious agent systems. Its limitation that the standard sequent calculus rules only provide a strategy of being safe by always taking all future options into account is overcome. Indeed, our choice calculus provides agents with various strategies at each decision making point when it comes to internal choices and future possibilities. In particular, agents can make predictions of future events and/or can decide early future decisions and act accordingly. The combinations of these strategies reflect how cautious the agents are when dealing future changes, how agents strike the balance between safety and resource efficiency, how agents match up their plans with the future via predictions and how agents shape their future actions by early decisions. Moreover, as these strategies add flexibility into agents' decision making to deal with choices and changes, this is a step forward in providing flexible agent interaction.

Furthermore, the ability to deal with dependencies among distributed choices opens up another area for enhancing the quality of agents' decision making. Indeed, consideration of other or future choices or events can be specified as constraints to be satisfied on current choices. Hence, decision making by agents on choices is not carried out locally but with global and temporal awareness, and in a distributed manner.

Our second contribution is deriving a mechanism for agent reasoning to divide tasks into multiple subtasks which can be attempted concurrently in a distributed manner. In other words, rather than having human designers specify the distribution of concurrent tasks for agents, we can have agents construct a distributed model of task resolution by themselves. The mechanism is based on transferring inner dependencies into outer dependencies among distributed formulas. This is well suited to the nature of systems composed of multiple independent agents interacting with each other.

The mechanism also supports the notion of arbitrary partial achievement of goals and partial utilization of resources. This removes the need to pre-specify subgoals for various threads of interaction and lets agents work out the partial achievement of the goals and what remain. Interaction then can take place at agents' discretion, so long as it is beneficial to agents' goals. This further provides agents with an autonomy in interacting in open systems.

Our further work includes extending the choice calculus to other temporal operators like $\Box$ and $\Diamond$. We will also explore variations of the splitting up of formulas which directly encode various strategies of agents in dealing with choices. Furthermore, deriving an implementation platform using choice calculus and splitting up mechanisms for such a modeling of flexible agent interaction using TLL as [2] is also considered. Finally, there is scope for investigating the relationship between our approach for modeling choices, and the use of Computational tree logic (CTL).

## Acknowledgments

We would like to acknowledge the support of the Australian Research Council under grant DP0663147 and also thank the reviewers for their helpful comments.

## References

1. Munroe, S., Miller, T., Belecheanu, R.A., Pechoucek, M., McBurney, P., Luck, M.: Crossing the agent technology chasm: Experiences and challenges in commercial applications of agents. Knowledge Engineering Review **21**(4) (2006)
2. Pham, D.Q., Harland, J.: Temporal linear logic as a basis for flexible agent interactions. In: AAMAS '07: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems. (2007)
3. Hirai, T.: Temporal Linear Logic and Its Applications. PhD thesis, Graduate School of Science and Technology, Kobe University (2000)
4. Emerson, E.A.: Temporal and modal logic. Handbook of Theoretical Computer Science **B, Chapter 16** (1990) 995–1072
5. Girard, J.Y.: Linear logic. Theoretical Computer Science **50** (1987) 1–102
6. Harland, J., Winikoff, M.: Agent negotiation as proof search in linear logic. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2002) 938–939
7. Küngas, P.: Linear logic, partial deduction and cooperative problem solving. In Leite, J.A., Omicini, A., Sterling, L., Torroni, P., eds.: Declarative Agent Languages and Technologies, First International Workshop, DALT 2003. Melbourne, Victoria, July 15th, 2003. Workshop Notes. (2003) 97–112

8. Pham, D.Q., Harland, J.: Flexible agent protocols via temporal and resource-based reasoning. In: AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM Press (2006) 235–237
9. Küngas, P., Matskin, M.: Symbolic negotiation with linear logic. In: CLIMA IV: the Fourth International Workshop on Computational Logic in Multi-Agent Systems, New York, NY, USA (January 2004)

## A Sequent Calculus for TLL (extract)

$$\frac{}{p \vdash p}\ axiom \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma', A \vdash \Gamma'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ cut$$

$$\frac{\Gamma, A, B, \Gamma' \vdash \Delta}{\Gamma, B, A, \Gamma' \vdash \Delta}\ EL \qquad \frac{\Gamma \vdash \Delta, A, B, \Delta'}{\Gamma \vdash \Delta, B, A, \Delta'}\ ER \qquad \frac{\Gamma \vdash \Delta}{\Gamma, 1 \vdash \Delta}\ 1L \qquad \frac{}{\vdash 1}\ 1R$$

$$\frac{}{\bot \vdash}\ \bot L \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\ \bot R \qquad \frac{}{\Gamma, 0 \vdash \Delta}\ 0L \qquad \frac{}{\Gamma \vdash \top, \Delta}\ \top R$$

$$\frac{\Gamma \vdash A, \Delta}{\Gamma, A^{\bot} \vdash \Delta}\ -^{\bot}L \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash A^{\bot}, \Delta}\ -^{\bot}R$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \otimes B \vdash \Delta}\ \otimes L \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma' \vdash B, \Delta'}{\Gamma, \Gamma' \vdash A \otimes B, \Delta, \Delta'}\ \otimes R$$

$$\frac{\Gamma, A \vdash \Delta}{\Gamma, A \,\&\, B \vdash \Delta}\ \&L \qquad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \,\&\, B \vdash \Delta}\ \&L \qquad \frac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \,\&\, B, \Delta}\ \&R$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \oplus B \vdash \Delta}\ \oplus L \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \oplus B, \Delta}\ \oplus R \qquad \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \oplus B, \Delta}\ \oplus R$$

$$\frac{\Gamma, A \vdash \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \parr B \vdash \Delta, \Delta'}\ \parr L \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \parr B, \Delta}\ \parr R$$

$$\frac{\Gamma \vdash A, \Delta \quad \Gamma', B \vdash \Delta'}{\Gamma, \Gamma', A \multimap B \vdash \Delta, \Delta'}\ \multimap L \qquad \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \multimap B, \Delta}\ \multimap R$$

$$\frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x.A \vdash \Delta}\ \forall L \qquad \frac{\Gamma \vdash A[y/x], \Delta}{\Gamma \vdash \forall x.A, \Delta}\ \forall R \qquad \frac{\Gamma, A[y/x] \vdash \Delta}{\Gamma, \exists x.A \vdash \Delta}\ \exists L \qquad \frac{\Gamma \vdash A[t/x], \Delta}{\Gamma \vdash \exists x.A, \Delta}\ \exists R$$

$$\frac{A, \Gamma \vdash \Delta}{\Box A, \Gamma \vdash \Delta}\ \Box L \qquad \frac{!\Gamma, \Box\Delta \vdash A, \Diamond\Lambda, ?\Sigma}{!\Gamma, \Box\Delta \vdash \Box A, \Diamond\Lambda, ?\Sigma}\ \Box R$$

$$\frac{!\Gamma, \Box\Delta, A \vdash \Diamond\Lambda, ?\Sigma}{!\Gamma, \Box\Delta, \Diamond A \vdash \Diamond\Lambda, ?\Sigma}\ \Diamond L \qquad \frac{\Gamma \vdash A.\Delta}{\Gamma \vdash \Diamond A, \Delta}\ \Diamond R$$

$$\frac{!\Gamma, \Box\Delta, \Xi \vdash A, \Phi, \Diamond\Lambda, ?\Pi}{!\Gamma, \Box\Delta, \bigcirc\Xi \vdash \bigcirc A, \overline{\bigcirc}\,\Phi, \Diamond\Lambda, ?\Pi}\ \bigcirc \qquad \frac{!\Gamma, \Box\Delta, \Xi, A \vdash \Phi, \Diamond\Lambda, ?\Pi}{!\Gamma, \Box\Delta, \bigcirc\Xi, \overline{\bigcirc}\,A \vdash \overline{\bigcirc}\,\Phi, \Diamond\Lambda, ?\Pi}\ \overline{\bigcirc}$$

$$\frac{!\Gamma, \Box\Delta, \Xi \vdash \Phi, \Diamond\Lambda, ?\Pi}{!\Gamma, \Box\Delta, \bigcirc\Xi \vdash \overline{\bigcirc}\,\Phi, \Diamond\Lambda, ?\Pi}\ \bigcirc \to \overline{\bigcirc}$$

# Extending Propositional Logic with Concrete Domains in Multi-issue Bilateral Negotiation

Azzurra Ragone[1], Tommaso Di Noia[1], Eugenio Di Sciascio[1], Francesco M. Donini[2]

[1] SisInfLab, Politecnico di Bari, Bari, Italy
{a.ragone,t.dinoia,disciascio}@poliba.it
[2] Università della Tuscia , Viterbo, Italy
donini@unitus.it

**Abstract.** We present a novel approach to knowledge-based automated one-shot multi-issue bilateral negotiation handling, in a homogeneous setting, both numerical features and non-numerical ones. The framework makes possible to formally represent typical situations in real e-marketplaces such as "*if I spend more than 20000 € for a sedan then I want a navigator pack included*" where both numerical (price) and non-numerical (sedan, navigator pack) issues coexist. To this aim we introduce $\mathcal{P}(\mathcal{N})$, a propositional logic extended with concrete domains, which allows to: model relations among issues (both numerical and not numerical ones) via logical entailment, differently from well-known approaches that describe issues as uncorrelated; represent buyer's request, seller's supply and their respective preferences as formulas endowed with a formal semantics. By modeling preferences as formulas it is hence possible to assign a utility value also to a bundle of issues, which is obviously more realistic than the trivial sum of utilities assigned to single elements in the bundle itself. We illustrate the theoretical framework, the logical language, the one-shot negotiation protocol we adopt, and show we are able to compute Pareto-efficient outcomes, using a mediator to solve a multi objective optimization problem.

## 1 Introduction

Bilateral negotiation between agents is a challenging problem, which finds applications in a number of different scenarios, each one with its own peculiarities and issues. In this work we focus on automated negotiation in e-marketplaces [30]. Clearly, in such domains we do not simply deal with undifferentiated products (commodities as oil, cement, etc.) or stocks, where only price, time or quantity have to be taken into account. In fact also other features have to be considered during the negotiation process. When a potential buyer browses an automobile e-marketplace, she looks for a car fulfilling her needs and/or wishes, so not only the price is important, but also warranty or delivery time, as well as look, model, comfort and so on. In such domains it is harder to model not only the negotiation process, but also the request/offer descriptions, as well as finding the best suitable agreement. Recently, there has been a growing interest toward multi-issue negotiation, also motivated by the idea that richer and expressive descriptions of demand and supply can boost e-marketplaces (see *e.g.*, [29] for a reasonable set of motivations) but –to the best of our knowledge– also in recent literature,

issues are usually described as uncorrelated terms, without considering any underlying semantics. Notable exceptions are discussed in Section 8. In our approach we use knowledge representation in two ways: (1) exploiting a logic theory to represent relations among issues and (2) assigning utilities to formulas to represent agents having preferences over different bundles of issues. For what concerns the former, we introduce a logical theory that allows to represent, *e.g.*, through logical implication, that a Ferrari is an Italian car ($\texttt{Ferrari} \Rightarrow \texttt{ItalianMaker}$) or that an Italian car is not a German car ($\texttt{ItalianMaker} \Rightarrow \neg\texttt{GermanMaker}$). Furthermore we can express agent preferences over bundle of issues, *e.g.*, the buyer can state she would like to have a car with navigator pack, where the meaning of navigator pack is in the Theory ($\texttt{NavigatorPack} \Leftrightarrow \texttt{SatelliteAlarm} \wedge \texttt{GPS\_system}$). In this case, the utility assigned to a bundle is obviously not necessarily the sum of utilities assigned to single elements in the bundle itself. Moreover issues are often inter-dependent: the selection of one issue depends on the selection made for other issues: in our framework agents can express conditional preferences as *I would like a car with leather seats if its color is black* ($\texttt{ExternalColorBlack} \Rightarrow \texttt{Leather\_seats}$). In this work we introduce an extended propositional logic, $\mathcal{P}(\mathcal{N})$ enriched with concrete domains, which allows –as it is in the real world– to take into account preferences involving both numerical features and not numerical ones, *e.g.*, the seller can state that if you want a car with a GPS system you have to wait at least one month: ($\texttt{GPS\_system} \Rightarrow \texttt{deliverytime} \geq 31$) as well as preferences can involve only numerical ones: *e.g.*, the buyer can state that she can accept to pay more than 25000€ for a sedan only if there is more than a two years warranty ($\texttt{price} > 25000 \Rightarrow \texttt{year\_warranty} > 2$). Contributions of this paper include: the framework for automated multi-issue bilateral negotiation, the logical language to represent existing relations between issues and preferences as formulas, which is able to handle both numerical features and not numerical ones as correlated issues w.r.t. a logical Theory and the one-shot protocol we adopt, which allows to compute Pareto-efficient agreements, exploiting a mediator that solves a multi objective optimization problem. The rest of the paper is structured as follows: next section discusses the scenario and the assumptions we make; then we illustrate the modeling of issues through our logical language and the negotiation mechanism. Section 4 presents the multi-issue bilateral negotiation problem, Section 5 describes the computation of utilities for numerical fetures. Section 6 shows how to compute Pareto-efficient agreement and Section 7 summarizes the bargaining process. Related work and discussion close the paper.

## 2 Negotiation Scenario

We start introducing the negotiation mechanism and the assumptions characterizing our framework. So, in accordance with [25], we define: the *Space of possible deals*, the *Negotiation Protocol* and the *Negotiation Strategy*. For what concerns the *Space of possible deals*, since we solve a multi objective optimization problem, possible deals are all the solutions of the problem that satisfy the constraints, even if they do not maximize the objective function (the so called *feasible region* [11]). The *Negotiation Protocol* we adopt is a *one-shot* protocol with the presence of a mediator. Differently from the clas-

sical *Single-shot* bargaining [23], where one player proposes a deal and the other player may only accept or refuse it [2], in our framework we hypothesize the presence of an electronic mediator, that may automatically explore the negotiation space and discover Pareto-efficient agreements to be proposed to both parties. Such parties may then accept or refuse them. We recall that, basically, two different approaches to automated negotiation exist: *centralized* and *distributed* ones. In the first ones, agents elicit their preferences and then a mediator, or some central entity, selects the most suitable deal based on them. In the latter ones, agents negotiate through various negotiation steps reaching the final deal by means of intermediate deals, without any external help [5]. Distributed approaches do not allow the presence of a mediator because – as stated in [14, p.25] – agents cannot agree on any entity, so they do not want to disclose their preferences to a third party, that, missing any relevant information, could not help agents. In dynamic system a predefined conflict resolution cannot be allowed, so the presence of a mediator is discouraged. On the other hand the presence of a mediator can be extremely useful in designing negotiation mechanisms and in practical important commerce settings. As stated in [17], negotiation mechanisms often involve the presence of a mediator [3], which collects information from bargainers and exploit them in order to propose an efficient negotiation outcome. In Section 8 some approaches adopting a centralized approach are described. Although the main target of an agent is reaching a satisfying agreement, this alone it is not enough, since knowing if this agreement is also Pareto-efficient is a matter that cannot be left out. It is fundamental to assess *how hard* is to find Pareto-efficient agreements and check whether a given agreement is also Pareto-efficient. The presence of a trusted third party can help the parties to reach a Pareto-efficient agreement. As pointed out in [24, p.311], usually, bargainers may not want to disclose their preferences or utilities to the other party, but they can be more willing to reveal these information to a trusted – automated – mediator, helping negotiating parties to achieve efficient and equitable outcomes. The presence of a mediator and the one-shot protocol is an incentive for the two parties to reveal the true preferences, because they can trust in the mediator and they have a single possibility to reach the agreement with that counterpart. Therefore in our framework we propose a one-shot protocol with the intervention of a *mediator* with a proactive behavior: it suggests to each participant a *fair* Pareto-efficient agreement. For what concerns *strategy*, the players reveal their preferences to the mediator and then, once it has computed a solution, they can accept or refuse the agreement proposed to them; they refuse if they think possible to reach a better agreement looking for another partner, or another shot, or for a different set of bidding rules. Notice that here we do not consider the influence of the *outside options* in the negotiation strategy [18].

## 3 Representation of issues

We divide issues involved in a negotiation in two categories. Some issues may express properties that are true or false, like, *e.g.*, in an automotive domain, `ItalianMaker`,

[3] The most well known –and running– example of mediator is eBay site, where a mediator receives and validates bids, as well as presenting the current highest bid and finally determining the auction winner [17].

or `AlarmSystem`. We represent them as propositional atoms $A_1, A_2, \ldots$ from a finite set $\mathcal{A}$. Other issues involve numerical features like `deliverytime`, or `price` represented as variables $f_1, f_2, \ldots$, each one taking values in its specific domain $D_{f_1}, D_{f_2}, \ldots$, such as $[0, 90]$ (days) for `deliverytime`, or $[1{,}000, 20{,}000]$ (euros), for `price`. The variables representing numerical features are always constrained by comparing them to some constant, like $\texttt{price} < 20{,}000$, or $\texttt{deliverytime} \geq 30$, and such constraints can be combined into complex propositional requirements – also involving propositional issues – *e.g.*, $\texttt{ItalianMaker} \wedge (\texttt{price} \leq 25{,}000) \wedge (\texttt{deliverytime} < 30)$ (representing a car made in Italy, costing no more than 25,000 euros, delivered in less than 30 days), or $\texttt{AlarmSystem} \Rightarrow (\texttt{deliverytime} > 30)$ (expressing the seller's requirement "if you want an alarm system mounted you'll have to wait more than one month"). We now give precise definitions for the above intuitions, borrowing from a previous formalization of so-called *concrete domains* [1] from Knowledge Representation languages.

**Definition 1 (Concrete Domains, [1]).** *A* concrete domain $D$ *consists of a finite set* $\Delta_c(D)$ *of numerical values, and a set of predicates* $C(D)$ *expressing numerical constraints on* $D$.

For our numerical features, predicates will always be the binary operators $C(D) = \{\geq, \leq, >, <, =, \neq\}$, whose second argument is a constant in $\Delta_c(D)$[4]. We note that in some scenarios other concrete domains could be possible, *e.g.*, colors as RGB vectors in an agricultural market, when looking for or selling fruits.

Once we have defined a concrete domain and constraints, we can formally extend propositional logic in order to handle numerical features. We call this language $\mathcal{P}(\mathcal{N})$.

**Definition 2 (The language $\mathcal{P}(\mathcal{N})$).** *Let $\mathcal{A}$ be a set of propositional atoms, and $F$ a set of pairs $\langle f, D_f \rangle$ each made of a feature name and an associated concrete domain $D_f$, and let $k$ be a value in $D_f$. Then the following formulas are in $\mathcal{P}(\mathcal{N})$:*

1. *every atom $A \in \mathcal{A}$ is a formula in $\mathcal{P}(\mathcal{N})$*
2. *if $\langle f, D_f \rangle \in F$, $k \in D_f$, and $c \in \{\geq, \leq, >, <, =, \neq\}$ then $(fck)$ is a formula in $\mathcal{P}(\mathcal{N})$*
3. *if $\psi$ and $\varphi$ are formulas in $\mathcal{P}(\mathcal{N})$ then $\neg\psi$, $\psi \wedge \varphi$ are formulas in $\mathcal{P}(\mathcal{N})$. We also use $\psi \vee \varphi$ as an abbreviation for $\neg(\neg\psi \wedge \neg\varphi)$, $\psi \Rightarrow \varphi$ as an abbreviation for $\neg\psi \vee \varphi$, and $\psi \Leftrightarrow \varphi$ as an abbreviation for $(\psi \Rightarrow \varphi) \wedge (\varphi \Rightarrow \psi)$.*

In order to define a formal semantics of $\mathcal{P}(\mathcal{N})$ formulas, we consider interpretation functions $\mathcal{I}$ that map propositional atoms into $\{\mathsf{true}, \mathsf{false}\}$, feature names into values in their domain, and assign propositional values to numerical constraints and composite formulas according to the intended semantics.

**Definition 3 (Interpretation and models).** *An interpretation $\mathcal{I}$ for $\mathcal{P}(\mathcal{N})$ is a function (denoted as a superscript $\cdot^{\mathcal{I}}$ on its argument) that maps each atom in $\mathcal{A}$ into a truth value $A^{\mathcal{I}} \in \{\mathsf{true}, \mathsf{false}\}$, each feature name $f$ into a value $f^{\mathcal{I}} \in D_f$, and assigns truth values to formulas as follows:*

[4] So, strictly speaking, $C(D)$ would be a set of unary predicates with an infix notation, *e.g.*, $x > 5$ is in fact a predicate $P_{>5}(x)$ which is $\mathsf{true}$ for all values of $D_x$ greater than 5 and $\mathsf{false}$ otherwise; however, this distinction is not necessary in our formalization.

– $(fck)^{\mathcal{I}} =$ *true iff* $f^{\mathcal{I}}ck$ *is true in* $D_f$, $(fck)^{\mathcal{I}} =$ *false otherwise*
– $(\neg\psi)^{\mathcal{I}} =$ *true iff* $\psi^{\mathcal{I}} =$ *false*, $(\psi \wedge \varphi)^{\mathcal{I}} =$ *true iff both* $\psi^{\mathcal{I}} =$ *true and* $\varphi^{\mathcal{I}} =$ *true, etc., according to truth tables for propositional connectives.*

*Given a formula* $\varphi$ *in* $\mathcal{P}(\mathcal{N})$*, we denote with* $\mathcal{I} \models \varphi$ *the fact that* $\mathcal{I}$ *assigns true to* $\varphi$*. If* $\mathcal{I} \models \varphi$ *we say* $\mathcal{I}$ *is a* model *for* $\varphi$*, and* $\mathcal{I}$ *is a model for a set of formulas when it is a model for each formula.*

Clearly, an interpretation $\mathcal{I}$ is completely defined by the values it assigns to propositional atoms and numerical features.

*Example 1.* Let $\mathcal{A} = \{\texttt{Sedan}, \texttt{GPL}\}$ be a set of propositional atoms, $D_{\texttt{price}} = \{0, \ldots, 60000\}$ and $D_{\texttt{year\_warranty}} = \{0, 1, \ldots, 5\}$ be two concrete domains for the features `price`, `year_warranty`, respectively. A model $\mathcal{I}$ for both formulas:

$$\left\{ \begin{array}{l} \texttt{Sedan} \wedge (\texttt{GPL} \Rightarrow (\texttt{year\_warranty} \geq 1)), \\ (\texttt{price} \leq 5{,}000) \end{array} \right\}$$

is $\texttt{Sedan}^{\mathcal{I}} =$ true, $\texttt{GPL}^{\mathcal{I}} =$ false, $\texttt{year\_warranty}^{\mathcal{I}} = 0$, $\texttt{price}^{\mathcal{I}} = 4{,}500$.

Given a set of formulas $\mathcal{T}$ in $\mathcal{P}(\mathcal{N})$ (representing an ontology), we denote *model* for $\mathcal{T}$ as $\mathcal{I} \models \mathcal{T}$. An ontology is *satisfiable* if it has a model. $\mathcal{T}$ logically implies a formula $\varphi$, denoted by $\mathcal{T} \models \varphi$ iff $\varphi$ is true in all models of $\mathcal{T}$. We denote with $\mathcal{M}_{\mathcal{T}} = \{\mathcal{I}_1, \ldots, \mathcal{I}_n\}$, the set of all models for $\mathcal{T}$, and omit the subscript when no confusion arises.

The following remarks are in order for the concrete domains of our e-marketplace-oriented scenarios:

1. domains are *discrete*, with a *uniform* discretization step $\epsilon$. If the seller states he cannot deliver a car before one month, he is saying that the delivery time will be at least in one month and one day ($\texttt{deliverytime} \geq 32$), where $\epsilon = 1$ (in days).
2. domains are *finite*; we denote with $\max(D_f)$ and $\min(D_f)$ the maximum and minimum values of each domain $D_f$.
3. even for the same feature name, concrete domains are *marketplace dependent*. Let us consider `price` in two different marketplace scenarios: pizzas and cars. For the former one, the discretization step $\epsilon$ is the €-cent: the price is usually something like 4.50 or 6.00 €. On the other hand, specifying the price of a car we usually have 10,500 or 15,000 €; then the discretization step in this case can be fixed as 100 €.

The above Point 1 and the propositional composition of numerical constraints imply that the operators $\{\geq, \leq, >, <, =, \neq\}$ can be reduced only to $\geq, \leq$.

**Definition 4 (successor/predecessor).** *Given two contiguous elements* $k_i$ *and* $k_{i+1}$ *in a concrete domain* $D$ *we denote by:*

– $s : D \to D$ *the successor function:* $s(k_i) = k_{i+1} = k_i + \epsilon$
– $p : D \to D$ *the predecessor function:* $p(k_{i+1}) = k_i = k_{i+1} - \epsilon$

Clearly, $\max(D_f)$ has no successor and $\min(D_f)$ has no predecessor. Based on the above introduced notions, we can reduce $C_m(D)$ to $\{\leq, \geq\}$ using the following transformations:

$$f = k \longrightarrow (f \leq k) \wedge (f \geq k) \quad (1)$$

$$f \neq k \longrightarrow (f < k) \vee (f > k) \quad (2)$$

$$f > k \longrightarrow f \geq (k + \epsilon) \longrightarrow f \geq s(k) \quad (3)$$

$$f < k \longrightarrow f \leq (k - \epsilon) \longrightarrow f \leq p(k) \quad (4)$$

## 4 Multi Issue Bilateral Negotiation in $\mathcal{P}(\mathcal{N})$

Following [21], we use logic formulas in $\mathcal{P}(\mathcal{N})$ to model the buyer's demand and the seller's supply. Relations among issues, both propositional and numerical, are represented by a set $\mathcal{T}$ – for Theory – of $\mathcal{P}(\mathcal{N})$ formulas.

In a typical bilateral negotiation scenario, the issues within both the buyer's request and the seller's offer can be split into *strict requirements* and *preferences*. Strict requirements represent what the buyer and the seller want to be necessarily satisfied in order to accept the final agreement – in our framework we call strict requirements *demand/supply*. Preferences denote issues they are willing to negotiate on – this is what we call *preferences*.

**Example 1** *Suppose to have a buyer's request like "I would like a sedan with leather seats. Preferably I would like to pay less than 12,000 € furthermore I'm willing to pay up to 15,000 € if warranty is greater or equal than 3 years. (I don't want to pay more than 17,000 € and I don't want a car with a warranty less than 2 years)". In this example we identify:*
***demand**: I want a sedan with leather seats. I don't want to pay more than 17,000 €. I don't want a car with a warranty less than 2 years*
***preferences**: Preferably I would like to pay less than 12,000 , furthermore I'm willing to pay up to 15,000 € if warranty is greater or equal than 3 years.*

**Definition 5 (Demand, Supply, Agreement).** *Given an ontology $\mathcal{T}$ represented as a set of formulas in $\mathcal{P}(\mathcal{N})$ representing the knowledge on a marketplace domain*

- *a buyer's* demand *is a formula $\beta$ (for Buyer) in $\mathcal{P}(\mathcal{N})$ such that $\mathcal{T} \cup \{\beta\}$ is satisfiable.*
- *a seller's* supply *is a formula $\sigma$ (for Seller) in $\mathcal{P}(\mathcal{N})$ such that $\mathcal{T} \cup \{\sigma\}$ is satisfiable.*
- *$\mathcal{I}$ is a* possible deal *between $\beta$ and $\sigma$ iff $\mathcal{I} \models \mathcal{T} \cup \{\sigma, \beta\}$, that is, $\mathcal{I}$ is a model for $\mathcal{T}$, $\sigma$, and $\beta$. We also call $\mathcal{I}$ an* agreement.

The seller and the buyer model in $\sigma$ and $\beta$ the minimal requirements they accept for the negotiation. On the other hand, if seller and buyer have set strict attributes that are in conflict with each other, that is $\mathcal{M}_{\mathcal{T} \cup \{\sigma,\beta\}} = \emptyset$, the negotiation ends immediately because, it is impossible to reach an agreement. If the participants are willing to avoid the *conflict deal* [25], and continue the negotiation, it will be necessary they revise their strict requirements.

In the negotiation process both the buyer and the seller express some preferences on attributes, or their combination. The utility function is usually defined based on these preferences. We start defining buyer's and seller's preferences and their associated utilities: $u_\beta$ for the buyer, and $u_\sigma$ for the seller.

**Definition 6 (Preferences).** *The buyer's* negotiation preferences $\mathcal{B} \doteq \{\beta_1, \ldots, \beta_k\}$ *are a set of formulas in $\mathcal{P}(\mathcal{N})$, each of them representing the subject of a buyer's preference, and a utility function $u_\beta : \mathcal{B} \to \Re^+$ assigning a utility to each formula, such that $\sum_i u_\beta(\beta_i) = 1$.*
*Analogously, the seller's* negotiation preferences $\mathcal{S} \doteq \{\sigma_1, \ldots, \sigma_h\}$ *are a set of formulas in $\mathcal{P}(\mathcal{N})$, each of them representing the subject of a seller's preference, and a utility function $u_\sigma : \mathcal{S} \to \Re^+$ assigning a utility to each formula, such that $\sum_j u_\sigma(\sigma_j) = 1$.*

Buyer's request in Example 1 is then formalized as:

$$\begin{aligned}\beta &= \texttt{Sedan} \wedge \texttt{Leather\_seats} \wedge (\texttt{price} \leq 17{,}000) \wedge \\ &\quad (\texttt{year\_warranty} \geq 2) \\ \beta_1 &= (\texttt{price} \leq 12{,}000) \\ \beta_2 &= (\texttt{year\_warranty} \geq 3) \wedge (\texttt{price} \leq 15{,}000)\end{aligned}$$

As usual, both agents' utilities are normalized to 1 to eliminate outliers, and make them comparable. Since we assumed that utilities are additive, the *preference utility* is just a sum of the utilities of preferences satisfied in the agreement.

**Definition 7 (Preference Utilities).** *Let $\mathcal{B}$ and $\mathcal{S}$ be respectively the buyer's and seller's preferences, and $\mathcal{M}_{\mathcal{T}\cup\{\alpha,\beta\}}$ be their agreements set. The* preference utility *of an agreement $\mathcal{I} \in \mathcal{M}_{\mathcal{T}\cup\{\alpha,\beta\}}$ for a buyer and a seller, respectively, are defined as:*

$$u_{\beta,\mathcal{P}(\mathcal{N})}(\mathcal{I}) \doteq \Sigma\{u_\beta(\beta_i) \mid \mathcal{I} \models \beta_i\}$$
$$u_{\sigma,\mathcal{P}(\mathcal{N})}(\mathcal{I}) \doteq \Sigma\{u_\sigma(\sigma_j) \mid \mathcal{I} \models \sigma_j\}$$

*where $\Sigma\{\ldots\}$ stands for the sum of all elements in the set.*

Notice that if one agent *e.g.*, the buyer, does not specify soft preferences, but only strict requirements, it is as $\beta_1 = \top$ and $u_{\beta,\mathcal{P}(\mathcal{N})}(\mathcal{I}) = 1$, which reflects the fact that an agent accepts whatever agreement not in conflict with its strict requirements. From the formulas related to Example 1, we note that while considering numerical features, it is still possible to express strict requirements and preferences on them. A strict requirement is surely the ***reservation value*** [24]. In Example 1 the buyer expresses two reservation values, one on price *"more than 17,000 €"* and the other on warranty *"less than 2 years"*.

Both buyer and seller have their own reservation values on each feature involved in the negotiation process. It is the maximum (or minimum) value in the range of possible feature values to reach an agreement, *e.g.*, the maximum price the buyer wants to pay for a car or the minimum warranty required, as well as, from the seller's perspective the minimum price he will accept to sell the car or the minimum delivery time. Usually, each participant knows its own reservation value and ignores the opponent's one. Referring to price and the two corresponding reservation values $r_{\beta,\texttt{price}}$ and $r_{\sigma,\texttt{price}}$ for the buyer and the seller respectively, if the buyer expresses $\texttt{price} \leq r_{\beta,\texttt{price}}$ and the seller $\texttt{price} \geq r_{\sigma,\texttt{price}}$, in case $r_{\sigma,\texttt{price}} \leq r_{\beta,\texttt{price}}$ we have $[r_{\sigma,\texttt{price}}, r_{\beta,\texttt{price}}]$ as a **Z**one **O**f **P**ossible **A**greement — $ZOPA(\texttt{price})$, otherwise no agreement is possible [24]. More formally, given an agreement $\mathcal{I}$ and a feature $f$, $f^{\mathcal{I}} \in ZOPA(f)$ must hold.

Keeping the price example, let us suppose that the maximum price the buyer is willing to pay is 15,000, while the seller minimum allowable price is 10,000, then we can set the two reservation values: $r_{\beta,\texttt{price}}$ = 15,000 and $r_{\sigma,\texttt{price}}$ = 10,000, so the *agreement price* will be in the interval $ZOPA(\texttt{price}) = [10000, 15000]$.

Obviously, the reservation value is considered as private information and will not be revealed to the other party, but will be taken into account by the mediator when the

agreement will be computed. Since setting a reservation value on a numerical feature is equivalent to set a strict requirement, then, once the buyer and the seller express their strict requirements, reservation values constraints have to be added to them (see Example 1).

In order to formally define a Multi-issue Bilateral Negotiation problem in $\mathcal{P}(\mathcal{N})$, the only other elements we still need to introduce are the *disagreement thresholds*, also called disagreement payoffs, $t_\beta$, $t_\sigma$. They are the minimum utility that each agent requires to pursue a deal. Minimum utilities may incorporate an agent's attitude toward concluding the transaction, but also overhead costs involved in the transaction itself, *e.g.*, fixed taxes.

**Definition 8 (MBN-$\mathcal{P}(\mathcal{N})$).** *Given a $\mathcal{P}(\mathcal{N})$ set of axioms $\mathcal{T}$, a demand $\beta$ and a set of buyer's preferences $\mathcal{B}$ with utility function $u_{\beta,\mathcal{P}(\mathcal{N})}$ and a disagreement threshold $t_\beta$, a supply $\sigma$ and a set of seller's preferences $\mathcal{S}$ with utility function $u_{\sigma,\mathcal{P}(\mathcal{N})}$ and a disagreement threshold $t_\sigma$, a **M**ulti-issue **B**ilateral **N**egotiation problem (MBN) is finding a model $\mathcal{I}$ (agreement) such that all the following conditions hold:*

$$\mathcal{I} \models \mathcal{T} \cup \{\sigma, \beta\} \tag{5}$$

$$u_{\beta,\mathcal{P}(\mathcal{N})}(\mathcal{I}) \geq t_\beta \tag{6}$$

$$u_{\sigma,\mathcal{P}(\mathcal{N})}(\mathcal{I}) \geq t_\sigma \tag{7}$$

Observe that not every agreement $\mathcal{I}$ is a solution of an MBN, if either $u_\sigma(\mathcal{I}) < t_\sigma$ or $u_\beta(\mathcal{I}) < t_\beta$. Such an agreement represents a deal which, although satisfying strict requirements, is not worth the transaction effort. Also notice that, since reservation values on numerical features are modeled in $\beta$ and $\sigma$ as strict requirements, for each feature $f$, the condition $f^{\mathcal{I}} \in ZOPA(f)$ always holds by condition (5).

## 5 Utilities for Numerical Features

Buyer's/seller's preferences are used to evaluate how good is a possible agreement and to select the best one. On the other hand, also preferences on numerical features have to be considered, in order to evaluate agreements and how good an agreement is w.r.t. another one. Let us explain the idea considering the demand and buyer's preferences in Example 1.

*Example 2.* Referring to $\beta$, $\beta_1$ and $\beta_2$ in Example 1 let us suppose to have the offer [5]:

$$\sigma = \texttt{Sedan} \wedge (\texttt{price} \geq 15{,}000) \wedge (\texttt{year\_warranty} \leq 5)$$

Three possible agreements between the buyer and the seller are, among others:

$$\begin{aligned}\mathcal{I}_1 : \{&\texttt{Sedan}^{\mathcal{I}_1} = \text{true}, \texttt{Leather\_seats}^{\mathcal{I}_1} = \text{true},\\ &\texttt{price}^{\mathcal{I}_1} = 17{,}000, \texttt{year\_warranty}^{\mathcal{I}_1} = 3\}\\ \mathcal{I}_2 : \{&\texttt{Sedan}^{\mathcal{I}_2} = \text{true}, \texttt{Leather\_seats}^{\mathcal{I}_2} = \text{true},\end{aligned}$$

[5] For illustrative purpose, in this example we consider an offer where only strict requirements are explicitly stated. Of course, in the most general case also the seller can express his preferences.

$$\texttt{price}^{\mathcal{I}_2} = 16{,}000, \texttt{year\_warranty}^{\mathcal{I}_2} = 4\}$$
$$\mathcal{I}_3 : \{\texttt{Sedan}^{\mathcal{I}_3} = \text{true}, \texttt{Leather\_seats}^{\mathcal{I}_3} = \text{true},$$
$$\texttt{price}^{\mathcal{I}_3} = 15{,}000, \texttt{year\_warranty}^{\mathcal{I}_3} = 5\}$$

Looking at the values of numerical features, $\mathcal{I}_1$ is the best agreement from the seller's perspective whilst $\mathcal{I}_3$ is the best from the buyer's one. In fact, the buyer the less he pays, the happier he is and the contrary holds for the seller! The contrary is for the warranty: the buyer is happier if he gets a greater year warranty. On the other hand, $\mathcal{I}_2$ is a good compromise between buyer's and seller's requirements.

The above example highlights the need for utility functions taking into account the value of each numerical feature involved in the negotiation process. Of course, for each feature two utility functions are needed; one for the buyer — $u_{\beta,f}$, the other for the seller — $u_{\sigma,f}$. These functions have to satisfy at least the basic properties enumerated below. For the sake of conciseness, we write $u_f$ when the same property holds both for $u_{\beta,f}$ and $u_{\sigma,f}$. :

1. Since $u_f$ is a utility function, it is normalized to $[0 \dots, 1]$. Given the pair $\langle f, D_f \rangle$, it must be defined over the domain $D_f$.
2. From Example 2 we note the buyer is happier as the price decreases whilst the seller is sadder. Hence, $u_f$ has to be monotonic and whenever $u_{\beta,f}$ increases then $u_{\sigma,f}$ decreases and vice versa.
3. There is no utility for the buyer if the agreed value on price is greater or equal than its reservation value $r_{\beta,\texttt{price}}$ =17,000 and there is no utility for the seller if the price is less than or equal to $r_{\sigma,\texttt{price}}$ =15,000. Since concrete domains are finite, for the buyer the best possible price is $\min(D_{\texttt{price}})$ whilst for the seller is $\max(D_{\texttt{price}})$. The contrary holds if we refer to year warranty.

**Definition 9 (Feature Utilities).** *Let $\langle f, D_f \rangle$ be a pair made of a feature name $f$ and a concrete domain $D_f$ and $r_f$ be a reservation value for $f$. A* **feature utility function** *$u_f : D_f \to [0 \dots, 1]$ is a monotonic function such that*
*– if $u_f$ monotonically increases then (see Figure 1)*

$$\begin{cases} u_f(v) = 0, v \in [\min(D_f), r_f] \\ u_f(\max(D_f)) = 1 \end{cases} \tag{8}$$

*– if $u_f$ monotonically decreases then*

$$\begin{cases} u_f(v) = 0, v \in [r_f, \max(D_f)] \\ u_f(\min(D_f)) = 1 \end{cases} \tag{9}$$

*Given a buyer and a seller, if $u_{\beta,f}$ increases then $u_{\sigma,f}$ decreases and vice versa.*

Clearly, the simplest utility functions are the two linear functions:

$$u_f(v) = \begin{cases} 1 - \frac{v - \min(D_f)}{r_f - \min(D_f)} \ , \ v \in [\min(D_f), r_f[ \\ \qquad\qquad 0 \ , \ v \in [r_f, \max(D_f)] \end{cases} \tag{10}$$

if it monotonically decreases and

$$u_f(v) = \begin{cases} 1 - \frac{v - \max(D_f)}{r_f - \max(D_f)} \ , \ v \in [r_f, \max(D_f)[ \\ 0 \ , \ v \in [\min(D_f), r_f] \end{cases} \tag{11}$$

if it monotonically increases (see Figure 1).

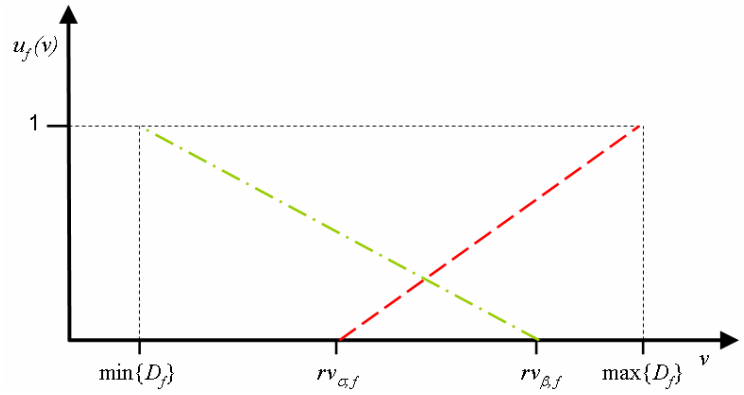



**Fig. 1.** Linear utility functions

## 6 Computing Pareto agreements in $\mathcal{P}(\mathcal{N})$

Among all possible agreements that we can compute, given a theory $\mathcal{T}$ as constraint, we are interested in agreements that are Pareto-efficient and *fair* for both the participants, in order to make them equally, and as much as possible, satisfied. We now outline how an actual solution can be found solving a multi objective optimization problem.

First of all, let $\{B_1, \ldots, B_k, S_1, \ldots, S_h\}$ be $k + h$ new propositional atoms, and let $\mathcal{T}' = \mathcal{T} \cup \{B_i \Leftrightarrow \beta_i | i = 1, \ldots, k\} \cup \{S_j \Leftrightarrow \sigma_j | j = 1, \ldots, h\}$ – that is, every preference in $\mathcal{B} \cup \mathcal{S}$ is equivalent to a new atom in $\mathcal{T}'$.

### 6.1 Objective functions

Here we define functions to be maximized to find a solution to a multi objective optimization problem. In order to formulate functions to be maximized involving preferences expressed as formulas in $\mathcal{P}(\mathcal{N})$, let $\{b_1, \ldots, b_k\}$ the (0,1)-variables one-one with $\{B_1, \ldots, B_k\}$ and similarly $\{s_1, \ldots, s_h\}$ for $\{S_1, \ldots, S_h\}$. The functions representing respectively buyer's and seller's utility over preferences can hence be defined as:

$$u_{\beta, \mathcal{P}(\mathcal{N})} = \sum_{i=1}^{k} b_i u_\beta(\beta_i) \tag{12}$$

$$u_{\sigma,\mathcal{P}(\mathcal{N})} = \sum_{j=1}^{h} s_j u_\sigma(\sigma_j) \tag{13}$$

As highlighted in Section 5, also utilities over numerical features have to be taken into account while finding the best solution for both the buyer and the seller. Hence, for each feature $f_t$ involved in the negotiation process we have a ***feature utility function*** for the buyer $u_{\beta,f_t}$ and one for the seller $u_{\sigma,f_t}$. For instance, if we consider `price` and the linear function in equations (10) and (11) we likely will have:

$$u_{\beta,\texttt{price}}(v) = \begin{cases} 1 - \frac{v - \max(D_{\texttt{price}})}{r_{\beta,\texttt{price}} - \max(D_{\texttt{price}})} \\ 0 \end{cases}$$

$$u_{\sigma,\texttt{price}}(v) = \begin{cases} 1 - \frac{v - \min(D_{\texttt{price}})}{r_{\sigma,\texttt{price}} - \min(D_{\texttt{price}})} \\ 0 \end{cases}$$

### 6.2 The Multi Objective Optimization Problem

Given the objective functions to be optimized – the *feature* utility functions and the *preference* utility functions – in order to compute a Pareto agreement we reduce to a multi objective optimization problem (MOP). The functions to be optimized are utility functions both for the buyer and the seller, as we want them equally satisfied.

In addition to the set of functions to maximize (or minimize), in a MOP there are a set of constrained numerical variables. In our setting, we have three different sets of constraints:

1. the (modified) ontology $\mathcal{T}'$ —see the beginning of Section 6
2. strict requirements $\beta$ and $\sigma$, including reservation values over numerical features
3. conditions (6) and (7) of an MBN on disagreement thresholds $t_\beta$ and $t_\sigma$ — see the definition of MBN-$\mathcal{P}(\mathcal{N})$ at the end of Section 4

Notice that the ones involving disagreements thresholds are already linear constraints. In order to model as linear constraints also the ones described in points 1 and 2 of the above enumeration, proceed as follows.

**Clause reduction** Obtain a set of clauses $\mathcal{T}''$ s.t. each clause contains only one single numerical constraint and $\mathcal{T}''$ is satisfiable iff $\mathcal{T}' \cup \{\sigma, \beta\}$ does. In order to have such clauses, if after using standard transformations in clausal form [16] you find a clause with two numerical constraints $\chi : A \vee \ldots (f_i c_i k_i) \vee (f_j c_j k_j)$ pick up a new propositional atom $\overline{A}$ and replace $\chi$ with the set of two clauses[6]

$$\left\{ \begin{array}{l} \chi_1 : \overline{A} \vee A \vee \ldots \vee (f_i c_i k_i), \\ \chi_2 : \neg\overline{A} \vee A \vee \ldots \vee (f_j c_j k_j) \end{array} \right\}$$

As a final step, for each clause, replace $\neg(f \leq k)$ with $(f \geq s(k))$ and $\neg(f \geq k)$ with $(f \leq p(k))$ (see (3) and 4).

[6] It is well know that such a transformation preserves logical entailment[27].

*Example 3.* Suppose to have the clause

$$\chi : \texttt{ItalianMaker} \vee \neg\texttt{AirConditioning} \vee (\texttt{year\_warranty} \geq 3) \vee \neg(\texttt{price} \geq 20,500)$$

First of all split the clause in the following two

$$\chi_1 : \overline{A} \vee \texttt{ItalianMaker} \vee \neg\texttt{AirConditioning} \vee (\texttt{year\_warranty} \geq 3)$$
$$\chi_2 : \neg\overline{A} \vee \texttt{ItalianMaker} \vee \neg\texttt{AirConditioning} \vee \neg(\texttt{price} \geq 20,500)$$

then change the second one in

$$\chi_2 : \neg\overline{A} \vee \texttt{ItalianMaker} \vee \neg\texttt{AirConditioning} \vee (\texttt{price} \leq 20,000)$$

Here we consider $\epsilon = 500$ for the concrete domain $D_{\texttt{price}}$.

**Encoding clauses into linear inequalities** Use a modified version of well-known encoding of clauses into linear inequalities (*e.g.*, [19, p.314]) so that every solution of the inequalities identifies a model of $\mathcal{T}''$. If we identify true with values in $[1 \ldots \infty]$ and false with values in $[0 \ldots 1[$ each clause can be rewritten in a corresponding inequality.

- map each propositional atom $A$ occurring in a clause $\chi$ with a (0,1)-variable $a$. If $A$ occurs negated in $\chi$ then substitute $\neg A$ with $(1 - a)$, otherwise substitute $A$ with $a$.
- replace $(f \leq k)$ with $\frac{1}{\max(D_f)-k}(\max(D_f) - f)$ and $(f \geq k)$ with $\frac{1}{k}f$.

After this rewriting it is easy to see that, considering $\vee$ – logical *or* – as classical addition, in order to have a clause true the evaluation of the corresponding expression must be a value grater or equal to 1.

*Example 4.* If we consider $\max(D_{\texttt{price}}) = 60,000$, continuing Example 3 we have from $\chi_1$ and $\chi_2$ the following inequalities respectively:

$$\overline{a} + i + (1 - a) + \frac{1}{3}\texttt{year\_warranty} \geq 1$$
$$(1 - \overline{a}) + i + (1 - a) + \frac{1}{60,000 - 20,000}(60,000 - \texttt{price}) \geq 1$$

where $\overline{a}$, $i$, $a$ are (0,1)-variables representing propositional terms $\overline{A}$, ItalianMaker and AirConditioning.

Looking at the example below, it should be clear the reason why only one numerical constraint is admitted in a clause.

*Example 5.* Let us transform the following clause without splitting in the two corresponding ones

$$\overline{\chi} : \texttt{ItalianMaker} \vee (\texttt{year\_warranty} \geq 3) \vee (\texttt{price} \leq 20,000)$$

the corresponding inequality is then

$$i + \frac{1}{3}\texttt{year\_warranty} + \frac{1}{60,000 - 20,000}(60,000 - \texttt{price}) \geq 1$$

The interpretation $\{\texttt{year\_warranty} = 2, \texttt{price} = 19,500\}$ is not a model for $\overline{\chi}$ while the inequality is satisfied.

## 7 The bargaining process

Summing up, the negotiation process covers the following steps:

**Preliminary Phase**. The buyer defines strict $\beta$ and preferences $\mathcal{B}$ with corresponding utilities $u_\beta(\beta_i)$ , as well as the threshold $t_\beta$, and similarly the seller $\sigma$, $\mathcal{S}$, $u_\sigma(\sigma_j)$ and $t_\sigma$. Here we are not interested in how to compute $t_\beta$,$t_\sigma$,$u_\beta(\beta_i)$ and $u_\sigma(\sigma_j)$; we assume they are determined in advance by means of either direct assignment methods (Ordering, Simple Assessing or Ratio Comparison) or pairwise comparison methods (like AHP and Geometric Mean) [20]. Both agents inform the mediator about these specifications and the theory $\mathcal{T}$ they refer to. Notice that for each feature involved in the negotiation process, both in $\beta$ and $\sigma$ their respective reservation values are set either in the form $f \leq r_f$ or in the form $f \geq r_f$.

**Negotiation-Core phase**. For each $\beta_i \in \mathcal{B}$ the mediator picks up a new propositional atom $B_i$ and adds the axiom $B_1 \Leftrightarrow \beta_i$ to $\mathcal{T}$, similarly for $\mathcal{S}$. Then, it transforms all the constraints modeled in $\beta$, $\sigma$ and (just extended) $\mathcal{T}$ in the corresponding linear inequalities following the procedures illustrated in Section 6.2 and Section 6.2. Given the preference utility functions $u_{\beta,\mathcal{P}(\mathcal{N})} = \sum_{i=1}^{k} b_i u_\beta(\beta_i)$ and $u_{\sigma,\mathcal{P}(\mathcal{N})} = \sum_{j=1}^{h} s_j u_\sigma(\sigma_j)$, the mediator adds to this set of constraints the ones involving disagreement thresholds $u_{\beta,\mathcal{P}(\mathcal{N})} \geq t_\beta$ and $u_{\sigma,\mathcal{P}(\mathcal{N})} \geq t_\sigma$.

With respect to the above set of constraints, the mediator solves a MOP maximizing the *preference* utility functions $u_{\beta,\mathcal{P}(\mathcal{N})}$, $u_{\sigma,\mathcal{P}(\mathcal{N})}$ and for each feature $f$ involved in the negotiation process also the *feature* utility functions $u_{\beta,f}$ and $u_{\sigma,f}$. The returned solution to the MOP is the agreement proposed to the buyer and the seller. Notice that a solution to a MOP is always Pareto optimal [11], furthermore the solution proposed by the mediator is also a *fair* solution, because among all the Pareto-optimal solutions we take the one maximizing the utilities of both the buyer and the seller (see Sec. 6.1 ). From this point on, it is a *take-it-or-leave-it* offer, as the participants can either accept or reject the proposed agreement [12]. Let us present a tiny example in order to better clarify the approach. Given the toy ontology in $\mathcal{P}(\mathcal{N})$,

$$\mathcal{T} = \begin{cases} \texttt{ExternalColorBlack} \Rightarrow \neg\texttt{ExternalColorGray} \\ \texttt{SatelliteAlarm} \Rightarrow \texttt{AlarmSystem} \\ \texttt{NavigatorPack} \Leftrightarrow \texttt{SatelliteAlarm} \wedge \texttt{GPS\_system} \end{cases}$$

the buyer and the seller specify their strict requirements and preferences:

$\beta$ = $\texttt{Sedan} \wedge (\texttt{price} \leq 30{,}000) \wedge (\texttt{km\_warranty} \geq 120{,}000) \wedge (\texttt{year\_warranty} \geq 4)$
$\beta_1$ = $\texttt{GPS\_system} \wedge \texttt{AlarmSystem}$
$\beta_2$ = $\texttt{ExternalColorBlack} \Rightarrow \texttt{Leather\_seats}$
$\beta_3$ = $(\texttt{km\_warranty} \geq 140{,}000)$
$u_\beta(\beta_1)$ = 0.5

$u_\beta(\beta_2)$ = 0.2
$u_\beta(\beta_3)$ = 0.3
$t_\beta$ =0.2

$\sigma$ = $\texttt{Sedan} \wedge (\texttt{price} \geq 20{,}000) \wedge (\texttt{km\_warranty} \leq 160{,}000) \wedge (\texttt{year\_warranty} \leq 6)$
$\sigma_1$ = $\texttt{GPS\_system} \Rightarrow (\texttt{price} \geq 28{,}000)$
$\sigma_2$ = $(\texttt{km\_warranty} \leq 150{,}000) \vee (\texttt{year\_warranty} \leq 5)$
$\sigma_3$ = $\texttt{ExternalColorGray}$
$\sigma_4$ = $\texttt{NavigatorPack}$
$u_\sigma(\sigma_1)$ = 0.2
$u_\sigma(\sigma_2)$ = 0.4
$u_\sigma(\sigma_3)$ = 0.2
$u_\sigma(\sigma_4)$ = 0.2
$t_\sigma$ =0.2

Then the final agreement is:

$$\begin{aligned}\mathcal{I} : \{&\texttt{Sedan}^{\mathcal{I}} = \text{true}, \texttt{ExternalColorGray}^{\mathcal{I}} = \text{true},\\ &\texttt{SatelliteAlarm}^{\mathcal{I}} = \text{true}, \texttt{GPS\_system}^{\mathcal{I}} = \text{true},\\ &\texttt{NavigatorPack}^{\mathcal{I}} = \text{true}, \texttt{AlarmSystem}^{\mathcal{I}} = \text{true},\\ &\texttt{price}^{\mathcal{I}} = 28{,}000, k^{\mathcal{I}} = 160{,}000, \texttt{year\_warranty}^{\mathcal{I}} = 5\}\end{aligned}$$

Here, for the sake of conciseness, we omit propositional atoms interpreted as false.

## 8 Related Work and discussion

Automated bilateral negotiation among agents has been widely investigated, both in artificial intelligence and in microeconomics research communities, so this section is necessarily far from complete. Several definitions have been proposed in the literature for bilateral negotiation. Rubinstein [26] defined the *Bargaining Problem* as the situation in which "two individuals have before them several possible contractual agreements. Both have interests in reaching agreement but their interests are not entirely identical. What 'will be' the agreed contract, assuming that both parties behave rationally?" In game theory, the bargaining problem has been modeled either as *cooperative* or *non-cooperative* games [10]. AI-oriented research has been more focused on automated negotiation among agents and on designing high-level protocols for agent interaction [15]. Agents can play different roles: act on behalf of buyer or seller, but also play the role of a mediator or facilitator. Approaches exploiting a mediator include among others [8, 13, 9]. In [8] an extended alternating offers protocol was presented, with the presence of a mediator, which improves the utility of both agents. In [13] a mediated-negotiation approach was proposed for complex contracts, where inter dependency among issues is investigated. In [3] the use of propositional logic in multi-issue negotiation was investigated, while in [4] weighted propositional formulas in preference modeling were considered. However, in such papers, no semantic relation among issues is taken into account. In our approach we adopt a logical theory, *i.e.*, an ontology, which allows *e.g.*, to catch inconsistencies between demand and supply or find out a feasible agreement in

a bundle, which is fundamental to model an e-marketplace. Self-interested agents negotiating over a set of resources to obtain an optimal allocation of such resources have been studied in [7, 6, 5]. Endriss et al. [7] propose an optimal resource allocation in two different negotiation scenarios: one, with money transfer, determines an allocation with maximal social welfare; the second is a money-free framework, which results in a Pareto outcome. In [5] agents negotiate over small bundles of resources, and a mechanism of resource allocation is investigated, which maximizes the social welfare by means of a sequence of deals involving at most *k* items each. Both papers [7, 5] extend the framework proposed in [28], which focused on negotiation for (re)allocating tasks among agents. We borrow from [31] the definition of agreement as a model for a set of formulas from both agents. However, in [31] only multiple-rounds protocols are studied, and the approach leaves the burden to reach an agreement to the agents themselves, although they can follow a protocol. The approach does not take preferences into account, so that it is not possible to guarantee the reached agreement is Pareto-efficient. Our approach, instead, aims at giving an *automated* support to negotiating agents to reach, in one shot, Pareto agreements. The work presented here builds on [22], where a basic propositional logic framework endowed of a logical theory was proposed. In [21] the approach was extended and generalized and complexity issues were discussed. In this paper we further extended the framework, introducing the extended logic $\mathcal{P}(\mathcal{N})$, thus handling numerical features, and showed we are able to compute Pareto-efficient agreements, solving a multi objective optimization problem adopting a one-shot negotiation protocol.

## References


1. F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *proc. of IJCAI-91*, pages 452–457, 1991.
2. K. Binmore. *Fun and Games. A Text on Game Theory*. D.C. Heath and Company, 1992.
3. S. Bouveret, M. Lemaitre, H. Fargier, and J. Lang. Allocation of indivisible goods: a general model and some complexity results. In *Proc. of AAMAS '05*, pages 1309–1310, 2005.
4. Y. Chevaleyre, U. Endriss, and J. Lang. Expressive power of weighted propositional formulas for cardinal preference modeling. In *Proc. of KR 2006*, pages 145–152, 2006.
5. Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. Negotiating over small bundles of resources. In *Proc. of AAMAS '05*, pages 296–302, 2005.
6. P. E. Dunne, M. Wooldridge, and M. Laurence. The complexity of contract negotiation. *Artif. Intell.*, 164(1-2):23–46, 2005.
7. U. Endriss, N. Maudet, F. Sadri, and F. Toni. On optimal outcomes of negotiations over resources. In *Proc. of AAMAS '03*, pages 177–184, 2003.
8. S. Fatima, M. Wooldridge, and N.R. Jennings. Optimal agendas for multi-issue negotiation. In *Proc. of AAMAS'03*, pages 129–136, 2003.
9. N. Gatti and F. Amigoni. A decentralized bargaining protocol on dependent continuous multi-issue for approximate pareto optimal outcomes. In *Proc. of AAMAS'05*, pages 1213–1214, 2005.
10. E. H. Gerding, D. D. B. van Bragt, and J. A. La Poutre. Scientific approaches and techniques for negotiation: a game theoretic and artificial intelligence perspective. Technical report, SEN-R0005, CWI, 2000.
11. F. Hillier and G. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 2005.

12. N.R. Jennings, P. Faratin, A.R. Lomuscio, S. Parsons, M.J. Wooldridge, and C. Sierra. Automated negotiation: prospects, methods and challenges. *Int. J. of Group Decision and Negotiation*, 10(2):199 – 215, 2001.
13. M. Klein, P. Faratin, H. Sayama, and Y. Bar-Yam. Negotiating complex contracts. In *Proc. of AAMAS'02*, pages 753–757, 2002.
14. S. Kraus. *Strategic Negotiation in Multiagent Environments*. The MIT Press, 2001.
15. A. R. Lomuscio, M. Wooldridge, and N. R. Jennings. A classification scheme for negotiation in electronic commerce. *Int Journal of Group Decision and Negotiation*, 12 (1):31–56, 2003.
16. D.W. Loveland. *Automated theorem proving: A logical basis*. North-Holland, 1978.
17. J.K. MacKie-Mason and M.P. Wellman. Automated markets and trading agents. In *Handbook of Computational Economics*. North-Holland, 2006.
18. A. Muthoo. On the strategic role of outside options in bilateral bargaining. *Operations Research*, 43(2):292–297, 1995.
19. C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., 1982.
20. J.C. Pomerol and S. Barba-Romero. *Multicriterion Decision Making in Management*. Kluwer Series in Operation Research. Kluwer Academic, 2000.
21. A. Ragone, T. Di Noia, E. Di Sciascio, and F.M. Donini. A logic-based framework to compute pareto agreements in one-shot bilateral negotiation. In *Proc. of ECAI'06*, pages 230–234, 2006.
22. A. Ragone, T. Di Noia, E. Di Sciascio, and F.M. Donini. Propositional- logic approach to one-shot multi issue bilateral negotiation. *ACM SIGecom Exchanges*, 5(5):11–21, 2006.
23. H. Raiffa. *The Art and Science of Negotiation*. Harvard University Press, 1982.
24. H. Raiffa, J. Richardson, and D. Metcalfe. *Negotiation Analysis - The Science and Art of Collaborative Decision Making*. The Belknap Press of Harvard University Press, 2002.
25. J.S. Rosenschein and G. Zlotkin. *Rules of Encounter*. MIT Press, 1994.
26. A. Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50:97–109, 1982.
27. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education-Prentice Hall, 2003.
28. T. Sandholm. Contract types for satisficing task allocation: I theoretical results. In *Proceedings of the AAAI Spring Symposium*, 1998.
29. D. Trastour, C. Bartolini, and C. Priest. Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In *Proc. WWW '02*, pages 89–98, 2002.
30. M.P. Wellman. Online marketplaces. In *Practical Handbook of Internet Computing*. CRC Press, 2004.
31. M. Wooldridge and S. Parsons. Languages for negotiation. In *Proc of ECAI '04*, pages 393–400, 2000.

# Towards Alternative Approaches to Reasoning about Goals

Patricia H. Shaw and Rafael H. Bordini

Department of Computer Science,
University of Durham, U.K.
{p.h.shaw,r.bordini}@durham.ac.uk

**Abstract.** Agent-oriented programming languages have gone a long way in the level of sophistication offered to programmers, and there has also been much progress in tools to support multi-agent systems development using such languages. However, much work is still required in mechanisms that can reduce the burden, typically placed on programmers, of ensuring that agents behave *rationally*, hence being effective and as efficient as possible. One such mechanisms is *reasoning about declarative goals*, which is increasingly appearing in the agents literature; it allows agents to make better use of resources, to avoid plans hindering the execution of other plans, and to be able to take advantage of opportunities for reducing the number of plans that have to be executed to achieve certain combinations of goals. In this paper, we introduce a Petri net based approach to such reasoning, and we report on experimental results showing that this technique can have a significant impact in the agent's behaviour (even though these experiments do not yet cover reasoning about resource usage). Our long term goal is to provide a number of alternative approaches for such reasoning, and incorporate them into interpreters for agent-oriented programming languages in such a way that the most appropriate approach is used at given circumstances.

## 1 Introduction

Recent years have seen an astonishing progress in the level of sophistication and practical use of various different agent-oriented programming languages [3]. These languages provide constructs that were specifically created for the implementation of systems designed on the basis of the typical abstractions used in the area of autonomous agents and multi-agent systems, therefore of much help for the development of large-scale multi-agent systems. However, the burden of ensuring that an agent behaves *rationally* in a given application is left to programmers (even though the languages do offer some support for that task).

Clearly, it would make the work of multi-agent systems developers much easier if we could provide (semi-) automatic mechanisms to facilitate the task of ensuring such rationality, provided, of course, that they are sufficiently fast to be used in *practical* agent programming languages. One important issue for a rational agent is that of *deliberation* — that is, deciding *which goals* to adopt

in the first place (see [15, 9, 2] for some approaches to agent deliberation in the context of agent programming languages). Besides, once certain goals have been adopted, the particular choice of plans to achieve them can cause a significant impact in the agent's behaviour and performance, as particular plans may interfere with one another (e.g., through the use of particular resources, or through the effects they have in the environment). The general term for the reasoning that is required to address these issues, which requires *declarative goal* representations [25, 24], has been called *reasoning about goals*.

Much work has been published recently introducing various approaches which contribute to addressing this problem [7, 21–23, 11, 16]. In most cases, in particular in the work by Thangarajah *et al.* and Clement *et al.*, the idea of "summary information" is used in the proposed techniques for reasoning about goals. However, the size of such summary information can potentially grow exponentially on the number of goals and plans the agent happens to be committed to achieve/execute [8]. It remains to be seen how practical those approaches will be for real-world problems.

In our work, we are interested in mechanisms for goal reasoning which do not require such summary information. This, of course, does not guarantee that they will be more efficient than the existing approaches. In fact, our approach is to try and use well-known formalisms with which to attempt to model the goal reasoning problem, then experimentally evaluating the various different approaches. We aim, in future work, to combine those approaches in such a way that agents can use one mechanism or another in the circumstances where each works best, if that turns out to be practically determinable.

So far, we have been able to model the goal reasoning problem using two different approaches, neither of which requires summary information as in the existing literature on the topic (the next section gives a detailed description of such work). First, we have modelled goal-adoption decision making as a reachability problem in a Petri net [14]. Then, using the idea and method suggested in [18, 17] for translating a Hierarchical Task Network (HTN) plan into a Constraint Satisfaction Problem (CSP), we have also developed a method for, given an agent's current goals and plans (possibly including a goal the agent is considering adopting), generating an instance of a CSP which can produce a valid ordering of plans — if one exists — to help the agent avoid conflicts (and take advantage of opportunities) when attempting to achieve all its goals.

For reasons of space, in this paper we focus on presenting the Petri net based technique only, and we also give initial experimental analysis of an agent's performance when using such goal reasoning in two different scenarios; the results of the CSP-based technique will be reported in a separate paper. The remainder of this paper is organised as follows. Section 2 gives an overview of the types of goal reasoning and various approaches appearing in the literature. Then in Section 3, we look at how such reasoning can be incorporated into a Petri net. Section 4 provides an experimental analysis of the Petri-net based reasoning. Finally, we give conclusions and a summary of future work in Section 5.

## 2 Reasoning About Goals

There are multiple types of conflicts that rational agents need to be aware of; these can be internal to the individual agent, or external between two or more agents [10]. While conflicts can occur in social interactions, when attempting to delegate or collaborate over a set of given tasks [5], the main focus of this paper is to look at conflicts between goals within an individual agent.

The conflicts arise within a single agent when it has taken on two or more goals that are not entirely compatible [10]. The conflicts may be caused if there is a limited amount of resources available [23, 16], or it may be due to the effects the actions involved in achieving the goals have on the environment; the actions in the plans being executed to achieve concurrent goals can cause effects which can hinder, or even prevent altogether, the successful completion of some of those plans [21, 22].

In all the work by Thangarajah *et al.* referred above, a Goal-Plan Tree (GPT) is used to represent the structure of the various plans and sub-goals related to each goal (see Figure 1). In order for a plan within the tree to be completed, all of its sub-goals must first be completed. However, to achieve a goal or sub-goal only one of its possible plans needs to be achieved. At each node on the tree, *summary information* is used to represent the various constraints under consideration. The reasoning done in their approach is solely internal to the individual agent.

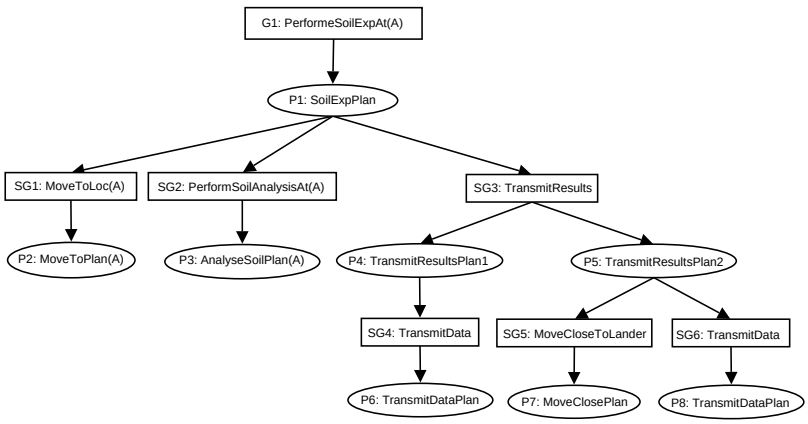



**Fig. 1.** Goal-Plan Tree for a Mars rover as used by Thangarajah *et al.* Goals and sub-goals are represented by rectangles, while plans are represented by ovals.

Reasoning about effects of actions needs to consider both positive and negative impacts in relation to other plans, and causal links that may exist between goals. In the first paper by Thangarajah *et al.* where reasoning about effects

is considered, they show how to detect and avoid negative interference between goals [21]. By using additional types of summary information, similar to those developed in [7], such as summaries for definite or potential pre-conditions and in-conditions along with post-conditions or effects, they monitor the causal links between effects produced by a plan which are used as pre-conditions of another to ensure these are not interfered with. To derive these effects, a formal notation based on set theory is defined, to allow the agent to produce the summary information in order to reason about conflicting actions between its current goals and any new goals the agent might consider adopting.

When conflicts occur, often they can be handled by scheduling the plan execution so as to protect the causal links until they are no longer required. Also in [21], the authors determine a sequence of steps for an agent to schedule plan execution so as to avoid interference, including checks that need to be performed before an agent can accept to adopt a new goal. Empirical results from experiments using the reasoning described in that paper are given in [19], comparing the performance of an agent with and without such reasoning, varying the level of interaction between goals and the amount of parallelism. The results show the improvement in number of goals successfully achieved, and only slight increase in time taken to perform the additional reasoning.

In [22], Thangarajah *et al.* focus on exploiting positive interaction between goals. This is where two or more plans cause the same effect, so rather than executing both, it might be possible to merge the two plans, thereby improving the agents' performance. To represent this form of reasoning, they again use the goal-plan tree with summary information including the *definite* and *potential* effects of the plans and goals; they also define a particular method to derive such summaries. They then describe how an agent can decide if it is feasible to merge the plans, and how to avoid waiting too long if one of the two plans selected for merging is reached considerably sooner than the other or the second plan is never reached, in case it was a "potential" merge rather than a "definite" merge. Results from experiments using this type of reasoning are once again presented in [19].

Horty and Pollack also consider positive interaction between plans [11]. In their work, an agent evaluates the various options it has between its goals within the context of its existing plans. They use estimates for the costs of plans, and where there is some commonality between some plans, those plans will be considered for merging. If the estimated merged cost is less than the sum of the two separate estimated costs, then the plans are actually merged. The example they give to illustrate this is an "important" plan for going to a shopping centre to buy a shirt, while also having a less important goal of buying a tie. Both plans involve getting money and travelling to a shopping centre, so if the overall cost of buying the tie at the same time as the shirt is less than that of buying the tie separately, then the plans will be merged, even though the goal of having a tie is not as important. In this way, they look for the least expensive execution of plans involved in achieving the goals.

When referring to reasoning about resource usage in a GPT [23], Thangarajah *et al.* consider both reusable and consumable resources. For example, a communication channel is a reusable resource, while energy or time is consumed so they cannot be reused. Summaries of the resource requirements are passed up the tree towards the goal, describing which resources are *necessary* in order to achieve the goals, and also which resources are used only *potentially*. They introduce a notation, based on set theory, allowing the derivation of summaries for the resource requirements of each goal and plan with sub-goals. These can then be used to reason about where conflicts might occur, so that they can be avoided by choosing suitable alternative plans or appropriately ordering plan execution. An algorithm is given to compute whether it is feasible to add a new goal to the existing set of goals. The initial formation of the goal-plan tree and summary information for the agent is produced at compile time, and the highlighted conflicts are then monitored at runtime in an attempt to avoid conflict.

Empirical results from experiments done using such reasoning are given in [20]. They consider goal-plan trees of depth 2 and depth 5, varying the amount of parallelism between multiple goals, and the amount of competition for the resources either by reducing the availability or increasing the number of goals competing for the same resources. The reasoning is implemented as an extension to the JACK agent development system [4]; the extended system is called X-JACK. The performance of X-JACK is compared against the performance of JACK without any of the additional reasoning, and shows an improvement in performance regarding the number of goals successfully achieved, typically with only a half-second time increase in the computation cost.

In comparison, [16] also consider the use of limited resources when deliberating and performing actions in a multi-agent environment, where coordination and negotiation with the other agents is required. In their attempt to address the problem of limited resources within meta-level control, they make use of reinforcement learning to improve the agents' performance over time.

To our knowledge, while Thangarajah *et al.* have reported on experimental results for reasoning separately about each of those types of interactions between plans and goals as well as resource usage, no results appear in the literature showing what is the performance obtained when an agent is doing all those forms of reasoning simultaneously. All results are given for the individual types, to demonstrate the sole effects from the individual reasoning and the (typically very small) amount of added computational costs associated with it. The lack of combined results seem to suggest the possibility of there being interference between the different forms of reasoning presented in their approach. For example, if one reasoning suggests that performing a particular plan will cause one type of conflict (say, lack or resources), while another reasoning suggests that the only alternative plan for that goal will also cause a conflict (say, a negative interference with another goal), the agent may be unable to decide between the two without some additional overriding reasoning. It also remains unknown if their approach is still equally efficient when the various types of reasoning are combined.

The results were also limited in the depth of trees tested. In the real world, it is likely the plans (and hence the goals) would be far more complex, leading to trees of significantly greater sizes. However, using the summary information, as a goal-plan tree grows, the amount of summary information to handle could potentially grow exponentially [8], which would have a significant impact on the performance of the agent for larger problems.

Prior to the time that the work by Thangarajah *et al.* was published, the Distributed Intelligent Agents Group led by Edmund Durfee, produced some similar research for modelling — and reasoning about — plan effects, extending their work to cover multi-agent systems rather than individual agents [6–8]. In their work, they are interested in reasoning about conflicts to coordinate the actions of agents that use HTN planning, while the work by Thangarajah was based around BDI agents (focusing on individual agents instead). In [7], Clement *et al.* present the summary information for pre-, in-, and post-conditions of plans, which is adopted by Thangarajah *et al.* and used in goal-plan trees to reason about both resources and effects.

## 3 Reasoning About Goals using Petri Nets

Petri nets are mathematical models, with an intuitive diagrammatic representation, used for describing and studying concurrent systems [14]. They consist of *places* that are connected by *arcs* to *transitions*, with *tokens* that are passed from place to place through transitions. Transitions can only *fire* when there are sufficient tokens in each of the input places, acting as pre-conditions for the transition. A token is then removed from each input place, and one is placed in each of the output places. Places are graphically represented as circles, while transitions are represented as rectangles.

There are many variations on the basic Petri net representation, and many of these have been used in a variety of agent systems [13, 1]. Arcs can have weights associated with them, the default weight being one. Greater weights on arcs either require the place to have at least that many tokens for the transition to fire, or the transition adds to the output place that number of tokens as its output. Coloured Petri Nets are able to hold tokens of different types, representing for example different data types. The weightings on the arcs then match up and select the relevant tokens to fire. Reference nets allow nets to contain sub-nets. *Renew* is a Petri net editor and simulator that is able to support high-level Petri nets such as coloured and reference nets [12].

We have developed a method to represent an agents' goals and plans using Petri nets. Essentially, we are able to represent the same problems as expressed by goal-plan trees in the work by Thangarajah *et al.* (see Figure 2 for an example). According to the method we have devised, goals and plans are represented by a series of places and transitions. A plan consists of a sequence of actions that starts with a place, and has a transition to another place to represent each of the atomic actions that occur in sequence within that plan. Goals are also set up as places with transitions linked to the available plans for each goal or subgoal. In

Figure 2, the plans are enclosed in dark boxes, while the goals and subgoals are in light boxes. The plans and subgoals are nested within each other, matching the hierarchical tree structure of the GPT.

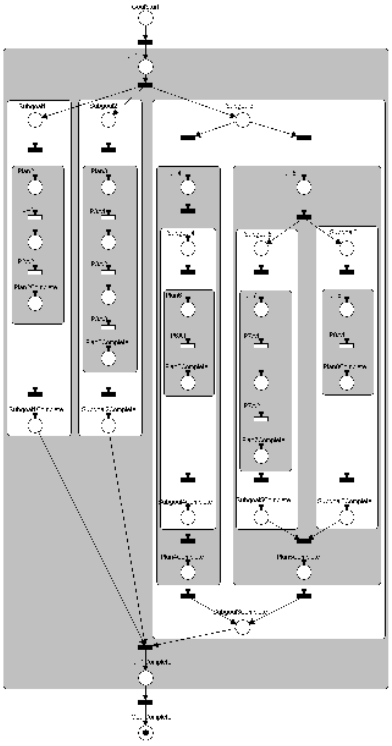
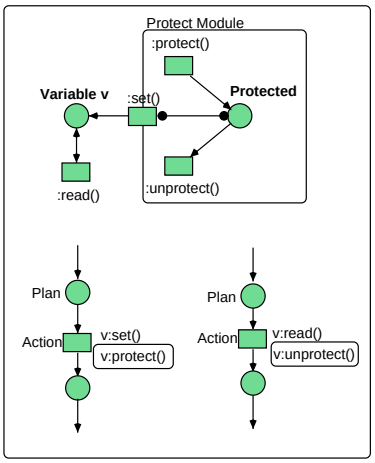


**Fig. 2.** Petri Net Representation of the Mars Rover GPT in Figure 1.

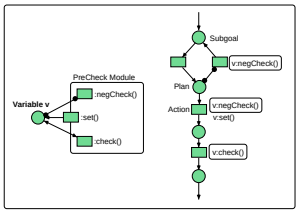
The goal reasoning that we have incorporated into the Petri nets is to allow an agent to handle both positive and negative interactions between multiple goals; we are in the process of incorporating reasoning about resources on top of these. Our aim is to be able to reason about these three aspects together whilst also avoiding the use of any "summary information" as in the work by Thangarajah *et al.* and Clement *et al.*. This reasoning and the representation of the plans and goals themselves can each be seen as an interlinked module, as will be discussed below. This modularisation of the method we use to represent goals and plans as (sub) Petri nets allows an agent to dynamically produce Petri net representations of goals and plans (and their relationship to existing goals and plans) that can then be used by an agent to reason on-the-fly about its ability to adopt a new goal given its current commitments towards existing goals.

Currently the Petri nets are being generated manually, but they have been designed in such modular way with the aim of being able to automate this process. An agent will then be able to generate new Petri nets to model new goals as the agent generates them or receive requests to achieve goals, allowing it to reason about whether it is safe to accept the new goal. If the goal is accepted then the Petri nets can be used to advise plan selection to avoid interference and to benefit from positive interactions. Figure 3 shows the main modules being used in the Petri nets. Some of the notation used in the Petri nets is specific to the *Renew* Petri net editor.

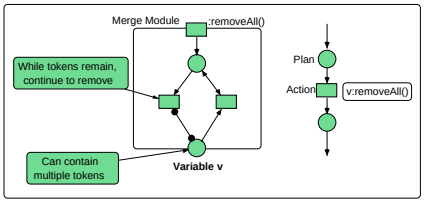
The negative interference reasoning protects the effects that have been caused in the environment until they are no longer required by the goal that caused the change. When an agent executes a plan that produces an effect in the environment, and that effect will be required by a later plan, the effect is immediately marked as protected until it is no longer required. This is done by using a protect module (Figure 3(a)) that adds a set of transitions and places to the Petri nets




(a) Protect module for negative interference.




(b) Pre-check module for positive interaction.




(c) Merge module for positive interaction.

**Fig. 3.** Petri-Net Representation of Modules for Reasoning about Goals.

so that when the relevant effect takes place, a transition is fired to protect it, then when it is no longer needed another transition is fired to release the protected effect. If another plan attempts to change something that will impact on the protected effects, then it will be stopped and forced to wait until the effects are no longer protected (i.e., until the release transition fires).

In the Mars Rover example, negative interference occurs when two or more goals require taking samples at different locations and after having moved to the first location, a second goal interferes to take the rover to another location before the sample is taken to satisfy the fist goal. To avoid this, the causal link is identified based on the effects and preconditions of the plans when Petri nets are generated, and a protect module is added to ensure other goals and plans cannot interfere with the casual link until the necessary plans have executed. In the Petri nets, the protect module is implemented by adding a place that holds a token to indicate if a variable is protected or not, with transitions that the

plan fires to protect the variable at the start of the causal link, then another transition to unprotect the variable when it is no longer required.

The positive interaction reasoning checks whether the desired effects have already been achieved (such as a Mars rover going to a specific location to perform some tests), or whether multiple goals can all be achieved by a merged plan rather than a plan for each goal, such as the Mars Rover transmitting all the data back in one go instead of transmitting separately individual results obtained by separate goals. When two or more plans achieve the same effect, only one of the plans has to be executed. This can greatly reduce the number of plans that are executed, especially if one of the plans has a large number of subgoals and plans. As a result, this can speed up the completion and reduce the costs of achieving the goals, particularly if there is a limited amount of resources.

In the Mars rover example, positive interaction can take place in both ways. Firstly, when moving to a different location the rover may have several goals all of which required going to the same location; however, only one plan needs to be actually executed to take the rover there. In the Petri nets, this is handled by a pre-check module (Figure 3(b)) that first checks whether another plan is about to, or has already, moved the rover to the new location, and if not it then fires a transition to indicate that the rover will be moving to the new location so the similar plans for other parallel goals do not need to be executed.

The second form of positive interaction is the direct merging of two or more plans. In the Mars rover scenario, this can occur when two or more goals are ready to transmit the data they have collected back to the base station. A merge module (Figure 3(c)) is added to indicate that when a goal is ready to transmit data back, it also checks to see if other goals are also ready to transmit their data. If so, all data that is ready is transmitted by the one plan rather than each goal separately executing individual plans to transmit the data.

## 4 Experimental Results and Analysis

We have used two different scenarios in our evaluation: the first is an abstract example and the other is the simple Mars rover example.

#### Scenario 1: Abstract Example

In this scenario, the goal structure in Figure 4 was used for each of the goals that were initiated. In the experiments reported here, we have opted for not considering varying structures, but this will be considered in future experiments. The experiments we conducted with Scenario 1 aimed to match, to the extent we could understand and reproduce, the settings of the experiments conducted in [19] to evaluate the GPT and summary information method that they introduced, in particular their experiments to compare the performance of JACK and X-JACK.

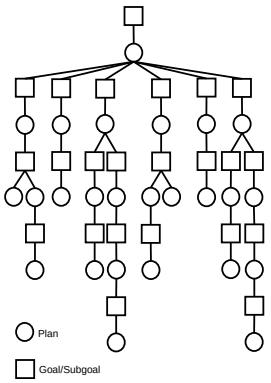



**Fig. 4.** Goal-Plan Tree Used for all Goals in Scenario 1.

In our experiments using Scenario 1, ten goal types were defined adjusting the selection of plans within the goal plan tree that would interact with those of other goals. The interaction was modelled through a set of common variables to which each goal was able to assign values. The variables and values are used to represent the different effects that plans can have in the environment.

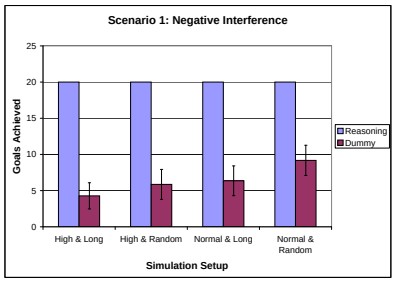
To stress-test the Petri nets, tests were set up that involved high levels of interaction, using a set of 5 variables, or low levels of interaction, using a set of 10 variables. Out of the 10 goal types, 5 of the goal types used 3 variables, while the remaining 5 goals types only altered 1 variable. During testing, 20 instantiations of the 10 possible goal types were created at random intervals and running concurrently. The Petri nets were implemented using Renew 2.1 [12], and each experiment was repeated 50 times.

Four experimental setups were used, with "High & Long" in the graphs (see Figure 5) corresponding to High Levels of Negative Interference for Long Periods, down to "Normal & Random" corresponding to Normal Levels of Negative Interference for Random Length Periods. The periods are controlled by defining the levels within the GPT that the interaction occurs at; so, for example, in the positive interaction, the duration over which the positive interaction takes place can be maximised by making plans in the top levels of the GPT with the greatest depth to interact.

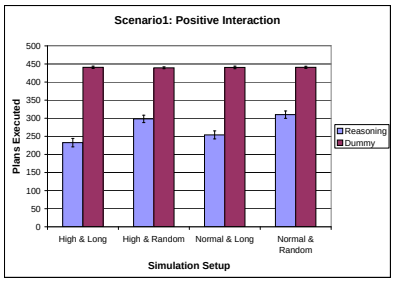
A dummy Petri net was set up using the same goal structure and set of goal types, but without any of the reasoning for positive or negative interaction. The results from running this against the Petri net where such reasoning was included could then be compared to show the improvements obtained by the reasoning.

**Negative Interference.** Each goal was given a set of 1 or 3 variables to which it was to assign a given value and then use it (recall that this represents the effects of plan execution in the environment). The positions in the goals where the variables were set and then used were varied either randomly or set to require the variables to be protected for the longest possible periods (meaning the state of the world caused by a plan is required to be preserved for longer periods before the interfering plans can be executed). The selections of plans in each goal are designed to cause interference for other goals being pursued simultaneously. This is done by ensuring a significant overlap in the variables which the goals are setting, particularly under high levels of interaction. The effect of the reasoning is measured by counting the number of goals achieved both by the "dummy" and by the "reasoning" Petri nets.

The results are shown in Figure 5(a). The graphs show the averages for the number of goals achieved by the reasoning Petri net and the dummy Petri net




(a) Experimental results for negative interference.




(b) Experimental results for nositive interaction.

**Fig. 5.** Results for Negative Interference and Positive Interaction in an Abstract Scenario.

from the 50 runs for each of the experiment sets, also showing the standard deviation. The effects of the negative reasoning are immediately obvious by the fact that the Petri nets with goal reasoning were consistently able to achieve all the goals, while the dummy Petri nets achieved, on average, very few goals, particularly when there were high levels of interference and variables that had to be protected for a long time, where it was only able to achieve approximately 21% of the goals, on average. Even at normal levels of interaction and random depth positioning, it was still only able to achieve, on average, 46% of the goals. The standard deviation shows that the performance of the dummy Petri nets was highly variable within the 50 runs of this experiment.

**Positive Interaction.** To measure the effects of reasoning about positive interactions, each goal was again given a set of 1 or 3 variables, with overlap between the goals, so that we can determine a selection of plans for each goal which can potentially be achieved by just executing one of the plans. Each goal contains 25 plans (in its GPT), of which at least 21 would have to be executed if the goal was being pursued on its own. This is due to two subgoals having a choice of plans to execute in the GPT. The scenario was set up to ensure all the goals are achievable without any reasoning, so the effects of the reasoning are measured by the number of plans that are required to execute in order to achieve all the goals.

As with the negative interference, the depth of the plans within the goal-plan structure at which merging can occur is varied. Plans with more subgoals will have a greater impact on the number of plans executed when merged than plans with no or very few subgoals. The tests were set with mergeable plans either high up in the GPT, or randomly placed within the tree.

The results are shown in Figure 5(b). The graphs show the averages for the number of plans executed by an agent using the Petri net for goal reasoning and a dummy agent; the averages are taken from the 50 runs for each of the experiment setups, and the graphs also show the standard deviations. There is

clearly a major improvement between the "dummy" and the "reasoning" agents in all of the simulation settings, with the reasoning agent requiring significantly fewer plans to be executed than the dummy, whilst still achieving the same goals. For high levels of interaction and mergeable plans at high levels in the GPT, there is an average drop of 47% in the number of plans being executed. Even with lower levels of interaction, and randomly placed mergeable plans, there is still a decrease of 30% on average. This could lead to large savings in the time and resources required by an agent to achieve its goals. While the standard deviation shows there is more variance in the performance of the reasoning agent than the dummy, this is due to the variations in depth and GPT of the merged plans. Even with the variance, the reasoning consistently made a significant improvement in the performance over the dummy agent.

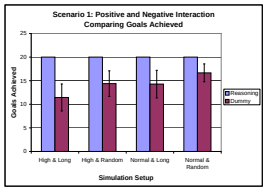
**Negative and Positive Interaction.** In this section, the two types of reasoning have been combined into one Petri net with a scenario that causes both negative interference and provides opportunities for positive interaction. To maintain exactly the same levels of interaction, both positively and negatively, the same GPT has been used again and the variables are duplicated for this abstract scenario. One set of variables is used for positive interaction, while the other is used for negative interference. This has been done, in the abstract scenario, to maintain the levels of interaction to allow for a clear comparison, but in the second scenario both forms of reasoning are applied to the same variables to represent a more realistic scenario.

Each goal is given 1 or 3 variables to assign values to for the negative interference, and the same number of variables for positive interaction. The number of goals achieved and the plans required are then measured to compare the expected performance of agent that uses the Petri-net based reasoning against a dummy agent (i.e., an agent without any goal-reasoning).

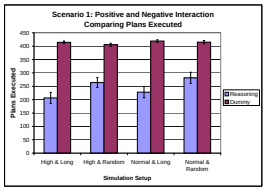
The four sets of tests were combined, in particular the negative interference at high levels of interaction over long periods was combined with the positive interference at high levels of interaction and at high levels within the GPT, while the negative interference at high levels of interaction over random periods was combined with the positive interference at high levels of interaction and at random levels within the GPT. The experiment for interaction at normal levels was combined in the same way.

The results are shown in Figure 6. These are broken down into three groups: 6(a) goals achieved, 6(b) plans executed, and 6(c) the ratio between plans executed and goals achieved. The standard deviations are also included in each of these graphs.

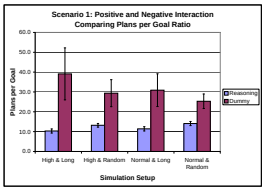
The reasoning agent is once again able to achieve all of its goals, while the dummy agent is still only able to achieve 57–83% of its goals. Not only is the dummy agent failing to achieve all its goals, it is also attempting to execute almost all its plans in an effort to find a solution. This means the effects of the positive interaction reasoning are also very obvious with a drop of 50% in the number of plans executed for high levels of negative interference with




(a) Comparison of goals achieved across the four experimental setups.




(b) Comparison of plans executed across the four experimental setups.




(c) Comparison of ratio between plans executed and goals achieved.

**Fig. 6.** Experimental Results for Combined Positive and Negative Interaction in an Abstract Scenario.

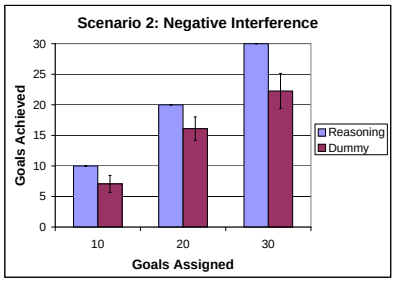
positive interaction for long periods in the GPT, while still maintaining a 32% decrease in plans at lower levels of interference. The *plan to goal ratio* shows that the reasoning agent only had to execute on average 10 plans at high levels of interaction, and 14 plans at lower levels of interaction, to achieve its goals, while the dummy agent had to execute on average 39 plans at high levels of interaction and 25 at normal levels. Recall that while in the GPT there are only 25 plans available to achieve the main goal on its own, the dummy agent was still executing plans in goals that failed, and the ratio shows all the plans executed compared to the goals achieved. The standard deviation shows that in general, the performance of the reasoning agent is very consistent, whereas the dummy agent is highly erratic, particularly when there are high levels of interaction for long periods.

### Scenario 2: Mars Rover

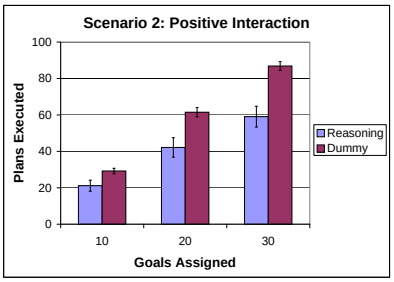
To show the reasoning being used in a more concrete example, a Mars rover scenario has also been used. In this scenario, the rover is given a set of locations and a set of tests (or tasks) to perform at each location. Each task at each location is represented by a separate goal, as shown in Figure 2, offering much opportunity for both negative and positive interactions. All of the plans contain a set of preconditions that must be true for it to be able to execute, and these preconditions are satisfied by the effects of other plans. So while there may be less plans involved than in Scenario 1, there is still a lot of interaction taking place. The preconditions lead to a partial ordering of the plans for the goal to be achieved. In our experiments, 2, 4, and 6 locations were used, with 5 tests carried out at each location, in order to evaluate the performance of the reasoning over different levels of concurrency, specifically 10, 20, or 30 goals being simultaneously pursued.

For the interests of comparison, the negative and positive reasoning have again been separated out before being combined together in the final set of experiments.

**Negative Interference.** Negative interference is caused when the rover goes to a location ready to perform its tasks, but is then interrupted by another goal that required going to a different location before the tasks required at the first location by the previous goal had been completed. The effects of the reasoning is again measured by the number of goals achieved. The results are shown in Figure 7(a).




(a) Reasoning about negative interference.




(b) Reasoning about positive interaction.

**Fig. 7.** Results for Negative Interference and Positive Interaction in the Mars Rover Example.

The results again show a definite improvement obtained by adding the reasoning about negative interference, whereby all goals were achieved, while the dummy agent is still only able to achieve on average 75% of its goals, across all the levels of goal concurrency, even at the lowest levels.

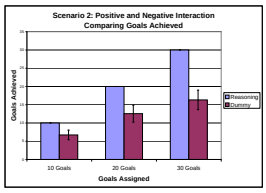
**Positive Interaction.** In the Mars Rover example, there are two main places for positive interaction. The first is when multiple goals all require the rover to perform tests/tasks at the same location, while the second is when the goals require transmitting their results back to the mission control team, after having performed the tests. When the goals have all obtained their test results, these can either be transmitted back to the base individually, or one goal can assume the responsibility of transmitting all the results back at the same time. This means only one plan has to be executed whereas without the reasoning an agent ends up executing one plan per goal.

The negative interference was removed from this setup to ensure all goals could be achieved without any reasoning. This meant the number of plans executed could be compared more fairly. The results are shown in Figure 7(b).

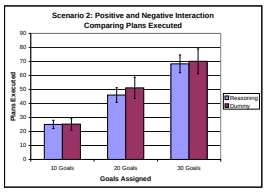
A clear reduction in the average number of plans executed can again be observed in these results, with higher levels of concurrency giving a 32% reduction in the number of plans executed to achieve the same goals. Even the lowest level of concurrency offers a 28% reduction that could be highly beneficial when there are many constraints imposed on an agent, such as time and resource availability.

**Combined Negative and Positive Interaction.** While both types of reasoning can be effectively used on their own, the combined effects of both types of reasoning gives the best results, particularly in highly constrained conditions. In the final set of results reported here, we show the results of the combined reasoning about negative interference and positive interaction in the Mars rover scenario.

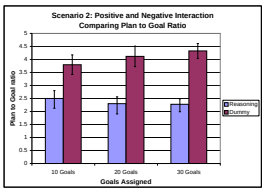
The results are shown in Figure 8. These are broken down into three groups: 8(a) goals achieved, 8(b) plans executed, and 8(c) the ratio between plans executed and goals achieved. The standard deviations are also included in each of these graphs.




(a) Comparing goals achieved.




(b) Comparing plans executed.




(c) Comparing the ratio between plans executed and goals achieved.

**Fig. 8.** Experimental Results for Reasoning about Negative and Positive Interaction in the Mars Rover Example.

While the results all show that there is only a slight improvement in the number of plans executed, the number of goals achieved by the reasoning agent is significantly more, and the *plan to goal ratio* is almost half that of the agent without any reasoning, increasing from a 34% reduction in the number of plans per goal to a 47% reduction as the amount of concurrency increases. The reasoning agent is again consistently achieving all the goals it has been given, while the proportion the dummy agent was able to achieve dropped from 67% to 54% as the amount of concurrency increased. The standard deviation also shows that the reasoning agent is more consistent in its results in this scenario, with a lower range of variation.

## 5 Conclusions and Future Work

In this paper we have presented an alternative approach to reasoning about negative and positive interactions between goals. The results clearly show a significant improvement in the number of goals being achieved, and the number of plans required to achieve them. To the best of our knowledge, this is the first time such types of reasoning have been presented combined together to show the joint effects of both positive and negative reasoning working in tandem for

an individual agent. As only a small extra computing cost is expected to result from the added reasoning, the benefits are very likely to outweigh any costs. However, in future work, we aim to analyse in detail the costs associated with the reasoning and compare this cost with alternative approaches such as a CSP representation and existing approaches such as the approach by Thangarajah *et al.* using a GPT [21–23]. In all experiments reported in this paper, such costs appeared to be negligible.

Preliminary work has been done in representing the same type of reasoning approached in this paper as a CSP, in order to provide further sources of comparison. A further type of reasoning that can be used to aid an agent is reasoning about resources, particularly when there is a limited supply of consumable resources available. We are currently in the process of including that type of reasoning in both our Petri-net and CSP-based techniques for reasoning about goals.

Currently, the Petri nets are being produced manually, but their modular design provides scope for automating this process, so that it can be incorporated into an agent architecture for on-the-fly reasoning about new goals to be potentially adopted. This will also be possible for the CSP-based approach, offering the agents a choice of reasoners if one proves to be better suited for particular situations (e.g., the structure/scale of the agent's GPT, or specific properties of the environment) than the others. Our long-term objective is to incorporate such reasoners into the interpreters of agent-oriented programming languages.

## Acknowledgements

We gratefully acknowledge the support of EPSRC's DTA scheme. Many thanks to Berndt Farwer for recommending the Renew tool and the help in using it.

## References

1. O. Bonnet-Torrès and C. Tessier. From team plan to individual plans: a petri net-based approach. In *proceedings of AAMAS'05, 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 797–804, New York, July 2005. ACM Press.
2. R. H. Bordini, A. L. C. Bazzan, R. de Oliveira Jannone, D. M. Basso, R. M. Viccari, and V. R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In C. Castelfranchi and W. Johnson, editors, *proceedings of First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002)*, pages 1294–1302, New York, USA, July 2002. NY: ACM Press.
3. R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*. Number 15 in Multi-agent Systems, Artificial Societies, and Simulated Organizations. Springer-Verlag, 2005.
4. P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK intelligent agents - components for intelligent agents in java. Technical report, Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia, 1998.

5. C. Castelfranchi and R. Falcone. Conflicts within and for collaboration. In C. Tessier, L. Chaudron, and H.-J. Müller, editors, *Conflicting Agents: Conflict Management in Multiagent Systems*, Multiagent systems, Artificial societies, and Simulated organizations, chapter 2, pages 33–62. Kluwer Academic Publishers, 2001.
6. B. J. Clement and E. H. Durfee. Identifying and resolving conflicts among agents with hierarchical plans. In *proceedings of AAAI Workshop on Negotiation: Settling Conflicts and Identifying Opportunities, Technical Report WS-99-12*, pages 6–11. AAAI Press, 1999.
7. B. J. Clement and E. H. Durfee. Theory for coordinating concurrent hierarchical planning agents using summary information. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 495–502, Menlo Park, CA, USA, 1999. AAAI Press.
8. B. J. Clement and E. H. Durfee. Performance of coordinating concurrent hierarchical planning agents using summary information. In *proceedings of 4th International Conference on Multi-Agent Systems (ICMAS)*, pages 373–374, Boston, Massachusetts, USA, July 2000. IEEE Computer Society.
9. M. Dastani, F. de Boer, F. Dignum, and J.-J. Meyer. Programming agent deliberation: an approach illustrated using the 3apl language. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 97–104, New York, NY, USA, 2003. ACM Press.
10. M. Hannebauer. Their problems are my problems - the transition between internal and external conflict. In C. Tessier, L. Chaudron, and H.-J. Müller, editors, *Conflicting Agents: Conflict Management in Multiagent Systems*, Multiagent systems, Artificial societies, and Simulated organizations, chapter 3, pages 63–110. Kluwer Academic Publishers, 2001.
11. J. F. Horty and M. E. Pollack. Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127(2):199–220, 2004.
12. O. Kummer, F. Wienberg, and M. Duvigneau. Renew – the Reference Net Workshop. Available at: http://www.renew.de/, May 2006. Release 2.1.
13. H. Mazouzi, A. El Fallah Seghrouchni, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 517–526, New York, NY, USA, 2002. ACM Press.
14. J. L. Peterson. *Petri Net Theory and the modeling of Systems*. Prentice-Hall, 1981.
15. A. Pokahr, L. Braubach, and W. Lamersdorf. A goal deliberation strategy for bdi agent systems. In T. Eymann, F. Klügl, W. Lamersdorf, and M. H. M. Klusch, editors, *Third German conference on Multi-Agent System TEchnologieS (MATES-2005); Springer-Verlag, Berlin Heidelberg New York, pp. 82-94*. Springer-Verlag, Berlin Heidelberg New York, 9 2005.
16. A. Raja and V. Lesser. Reasoning about coordination costs in resource-bounded multi-agent systems. *proceedings of AAAI 2004 Spring Symposium on Bridging the multiagent and multi robotic research gap*, pages 25–40, March 2004.
17. P. Surynek. On state management in plan-space planning from CP perspective. In *In proceedings of Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems, International Conference on Automated Planning and Scheduling ICAPS, Cumbria, UK*. AAAI Press, June 2006.
18. P. Surynek and R. Barták. Encoding HTN planning as a dynamic CSP. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Con-*

*ference, Sitges, Spain*, volume 3709 of *Lecture Notes in Computer Science*, page 868. Springer, October 2005.

19. J. Thangarajah. *Managing the Concurrent Execution of Goals in Intelligent Agents*. PhD thesis, School of Computer Science and Informaiton Technology, RMIT University, Melbourne, Victoria, Australia, December 2004.
20. J. Thangarajah and L. Padgham. An empirical evaluation of reasoning about resource conflicts in intelligent agents. In *proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 04)*, pages 1298–1299, 2004.
21. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *proceedings of 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 721–726, Acapulco, Mexico, August 2003. Morgan Kaufmann.
22. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and exploiting positive goal interaction in intelligent agents. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 401–408, New York, NY, USA, 2003. ACM Press.
23. J. Thangarajah, M. Winikoff, and L. Padgham. Avoiding resource conflicts in intelligent agents. In F. van Harmelen, editor, *proceedings of 15th European Conference on Artifical Intelligence 2002 (ECAI 2002)*, Amsterdam, 2002. IOS Press.
24. M. B. van Riemsdijk, M. Dastani, and J.-J. C. Meyer. Semantics of declarative goals in agent programming. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 133–140, New York, NY, USA, 2005. ACM Press.
25. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), 22–25 April, Toulouse, France*, pages 470–481, 2002.

# Author Index