A terminological interpretation of (Abductive) Logic Programming

Marc Denecker *

Department of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium.
e-mail : marcd@cs.kuleuven.ac.be

Abstract

The logic program formalism is commonly viewed as a modal or default logic. In this paper, we propose an alternative interpretation of the formalism as a terminological logic. A terminological logic is designed to represent two different forms of knowledge. A TBox represents definitions for a set of concepts. An ABox represents the *assertional knowledge* of the expert. In our interpretation, a logic program is a TBox providing definitions for all predicates; this interpretation is present already in Clark's completion semantics. We extend the logic program formalism such that some predicates can be left undefined and use classical logic as the language for the ABox. The resulting logic can be seen as an alternative interpretation of abductive logic program formalism. We study the expressivity of the formalism for representing uncertainty by proposing solutions for problems in temporal reasoning, with null values and open domain knowledge.

1 Introduction

The logic program formalism is commonly viewed as a modal, autoepistemic logic or, closely related, as a default logic. In these different interpretations the connective *not* is interpreted as a modal connective (*not p* means *it not believed that p*) or as a default connective (*not p* means *it can be assumed by default* $\neg p$). In this paper, we investigate a complementary view on logic programming which relates this formalism to another important class of logics in AI and knowledge representation, the terminological logics. At the origin of the terminological languages lies the observation that an expert's knowledge consists of a *terminological component* and an *assertional component* [1]. The terminological component, described in the TBox, consists of the definitions of the *technical vocabulary* of the expert. The role of the assertional component, described in the ABox is intimately tied to the representation of uncertainty: when a number of concepts cannot be defined, then other, less precise information may be available which can be represented as a set of *assertions*.

*supported by a post-doctoral mandate of the research fund of the K.U.Leuven

The view that we investigate in this paper is that a logic program is a TBox: it gives definitions to the predicates. The interpretation of a logic program as a formalism for representing definitions is already present in Clark's work on completion semantics [2]. Under Clark's interpretation, a program consists of a set of possibly empty definitions for all predicates; predicates are defined by enumerating exhaustively the cases in which they are true. Under this interpretation, a problem with the use of logic programs for knowledge representation, is that an expert needs to provide definitions for all predicates. In many applications, such complete information is not available. A natural solution is to extend the logic program formalism such that a subset of the predicates can be defined while other predicates can be left undefined. One straightforward way to represent such an *open logic program*, as called in the sequel, is as a tuple (P, D), or P^D, which consists of a set P of program clauses and a set D of predicates which are defined in P; D contains all predicates which occur in the head of a program clause of P and possibly some other predicates with an empty definition. In the sequel, predicates in D are called *defined predicates*; other predicates are called *open predicates*. Note that in this formalism, one can distinguish predicates p which have an empty definition ($p \in D$) and predicates which have no definition ($p \notin D$). Just as in terminological logics, there is need for an ABox to represent assertional knowledge on open predicates; one can use classical first-order logic (FOL) for this. We thus obtain a new type of theories, called OLP-FOL theories or open theories, which consists of two components: an open logic program P^D and a set of FOL axioms T.

The interpretation of a logic program under the modal and default view on one hand and the terminological view on the other hand is fundamentally different. This appears very clearly in the following example. Consider the following logic program:

$$P = \{ \quad dead(X) \text{ :- } not\ alive(X) \quad \}$$

Under the auto-epistemic interpretation, the program clause reads as: *if it is not believed that X is alive then X is dead.* P does not contain any information about the aliveness of, say John. Therefore, it is not believed that John is alive, and therefore, it may be derived that he is dead.

Under the terminological view, the set P can be embedded in two open logic programs $P^{\{dead,alive\}}$ and $P^{\{dead\}}$. Both open programs contain the same definition of *dead* which reads as: *X is dead iff X is not alive.* The first open logic program provides full information on alive: nobody is alive; hence, everybody is dead. The second open logic program contains no information on who is alive. Hence, this program does not allow to determine whether or not John is dead. Note that in $P^{\{dead\}}$ there is no evidence that John is alive; equivalently, it is not believed that John is alive. Yet the conclusion that he is dead cannot be drawn from $P^{\{dead\}}$. This shows that *not p* in OLP-FOL cannot be interpreted as *it is not believed that p* or *there is no evidence that p*. The *not* connective in OLP-FOL has the same *objective* modality as classical negation $\neg$ in FOL. This has important consequences for the negation as (finite) failure principle. Given an open logic program which defines all predicates, negation as (finite) failure acts as a solid inference rule to derive objective negative information. However, the inference rule cannot be applied for open logic programs. For example, the goal $\leftarrow alive(john)$ finitely fails in $P^{\{dead\}}$; yet, $\neg alive(john)$ may not be derived.

OLP-FOL does not contain default rules nor modal connectives. Yet, the formalism is nonmonotonic. Extending a definition by adding a new positive case is a nonmonotonic operation. E.g. a logic program $\{p(a) \text{ :- }\}$ entails $\neg p(b)$. After adding $p(b)$:- , it entails $p(b)$. OLP-FOL can be seen as an alternative interpretation of abductive logic programming [13]. The formal objects in OLP-FOL and the abductive logic program formalism (in the sequel: ALP) are the same: both contain tuples with a set of program clauses, an equivalent partition of the predicates in two subsets and a set of FOL formulas. Yet, OLP-FOL and ALP interpret these objects in different ways.

First, abductive logic programming is seen as the study of abduction in the context of logic programming. In contrast, we view OLP-FOL as a declarative logic in its own right. In [7], we investigated the use of other computational paradigms such as deduction and satisfiability checking on an equal basis as abduction. Second, the role of the formal model semantics is different in ALP and OLP. In ALP, a model semantics is meant as a specification of what abductive solutions must be computed [3] [14] [22]. In contrast, we look at model semantics as a formalisation of the declarative and descriptive meaning of an OLP-FOL theory. Third, the role of the FOL component in ALP and OLP seems different. In [13], a FOL formula is interpreted as an *integrity constraint*. The role of an integrity constraint is determined by its relation to abduction and can be classified either under the *theoremhood view*, the *consistency view* or *epistemic view*. On the other hand, OLP-FOL is an integration of two language components on equal basis. The FOL axioms in an OLP-FOL theory have the same role and meaning as in a FOL theory and are not bound to a specific role in any form of computation.

The content of this paper is the following. In section 2, we define the syntax of OLP-FOL and introduce the semantical primitives. Based on these, section 3 gives a semantics for the logic. Section 4 gives a number of well-chosen examples which clarify the role of OLP-FOL for knowledge representation. The paper closes with a discussion of related work and a conclusion. The proofs of the theorems are omitted and can be found in [4] and [6].

2 Preliminaries.

An alphabet Σ, terms, atoms, literals and formulae based on Σ are defined as usual. As usual, a free variable in a formula is not bound by a quantifier; a closed or ground formula does not contain free variables. Substitutions and variable assignments are defined as usual. A program clause here corresponds to a *general program clause* as defined in [19] except that we use the operator " :- " instead of "$\leftarrow$" in order to distinguish between program clauses and FOL implications.

An open logic program P^D is a pair of a set P of program clauses and a subset D of predicates of Σ^p, called *defined*, such that $= \notin D$ and all predicates in the head of program clauses in P are contained in D. The other non-equality predicates in $\Sigma^p \setminus D$ are called *open*. A closed logic program is an open logic program which defines all non-equality predicates. The *definition* of a defined predicate p in P is the set of clauses with p in the head. This set may be empty. An OLP-FOL theory $\mathcal{T}$, also called open theory, is a tuple (P^D, T) with an

open logic program P^D and a set of FOL axioms T.

In the next section we define a model semantics for OLP-FOL. The model semantics is based on the notion of a 3-valued Σ-interpretation (for a definition see e.g. [17]). It is essentially a domain, a correspondence between functors and functions on the domain and a correspondence between predicate symbols and 3-valued relations on the domain. With a Σ-interpretation I, a truth function $\mathcal{H}_I$ can be associated, which maps closed formulas to truth values, as usual.

3 The semantics of OLP-FOL.

The formal declarative semantics of an OLP-FOL theory is given by the class of its models. In the model theory of OLP-FOL, a model intuitively represents a possible state of the world. Both the terminological component and the assertional component of an OLP-FOL theory can be characterised by the way they impose restrictions on the possible states of the problem domain. Therefore, though OLP-FOL is an amalgamation of two different languages, it has a conceptually coherent semantics.

The semantics of the FOL-language is defined as usual except that 3-valued interpretations are allowed.

Definition 3.1 *A Σ-interpretation I satisfies a closed FOL formula F iff $\mathcal{H}_I(F) = \mathbf{t}$. A Σ-interpretation I satisfies a FOL theory T iff for all $F \in T : \mathcal{H}_I(F) = \mathbf{t}$.*

We denote this by $I \models F$ and $I \models T$.

For the semantics of the open logic program language, we follow more or less Clark's approach. Our approach extends both the 2-valued completion semantics for abduction in logic programming of [3] and the 3-valued completion semantics for closed logic programs of [17]. An open logic program P^D defines a set $comp_3(P^D)$, consisting of the *completed definitions* of the defined predicates of P^D [17]. As in [17], the classical equivalence operator $\leftrightarrow$ is replaced by a new operator $\Leftrightarrow$. The truth function $\mathcal{H}_I$ associated to some interpretation I is extended for this logical connective as given by the following table:

$\mathcal{H}_I(E_1 \Leftrightarrow E_2) = \mathbf{t}$ iff $\mathcal{H}_I(E_1) = \mathcal{H}_I(E_2)$
$\mathcal{H}_I(E_1 \Leftrightarrow E_2) = \mathbf{f}$ iff $\mathcal{H}_I(E_1) \neq \mathcal{H}_I(E_2)$

For two-valued interpretations, the meaning of the normal logical equivalence $\leftrightarrow$ and $\Leftrightarrow$ is the same. The use of $\Leftrightarrow$ is restricted to completed definitions and may not occur in the FOL component of an open theory.

An integration of a logic programming formalism and FOL raises a problem concerning the role of equality. Logic programming formalisms subsume virtually always the theory of Free Equality (denoted $FEQ(\Sigma^f)$), also called Clark's Equality theory [2] or the Unique Names Axioms [25]. In intuitive terms, two different terms represent different objects. These axioms lack in FOL; unless explicitly forbidden, two different terms in a FOL theory possibly represent the same object. Because computational techniques in logic programming such as

negation as failure rely on Free Equality, we define OLP-FOL as a free equality logic. For a definition of $FEQ(\Sigma^f)$, we refer to [2].

Using the above definitions, the declarative semantics of an OLP-FOL theory is defined as follows:

Definition 3.2 *A Σ-interpretation I is a Σ-model of an OLP-FOL theory (P^D, T) iff I is 2-valued on all open predicates and I satisfies $T \cup \mathrm{FEQ}(\Sigma^f) \cup comp_3(P^D)$.*

Given is an OLP-FOL theory $\mathcal{T}$ and a FOL formula F based on Σ.

Definition 3.3 *$\mathcal{T}$ is consistent iff there exists a Σ-model of $\mathcal{T}$. Otherwise $\mathcal{T}$ is inconsistent. $\mathcal{T}$ entails F iff for each Σ-model M of $\mathcal{T}$, $M \models F$. F is satisfiable with $\mathcal{T}$ iff there is a Σ-model of $\mathcal{T}$ which satisfies F.*

The OLP-FOL logic is an integration of logic programming and classical logic. The integration of FOL in OLP-FOL is subject to one important restriction: the embedding of a FOL theory T will be equivalence preserving only if T entails the theory of Free Equality.

Theorem 3.1 *A closed logic program P based on Σ under Kunen's completion semantics [17] is logically equivalent with the OLP-FOL theory (P^D, ϕ) with $D = \Sigma^p \setminus \{=\}$.*

A FOL theory T based on Σ which entails $\mathrm{FEQ}(\Sigma^f)$ is logically equivalent with the OLP-FOL theory (ϕ^ϕ, T).

In principle, the free equality in OLP-FOL does not restrict the expressivity in OLP-FOL compared to FOL. N-ary functors for which the axioms in FEQ do not hold can always be replaced by n+1-ary predicates.

The above semantics raises a number of questions. Why 3-valued logic and how to interpret **u**? Why are the open predicates 2-valued?

The OLP component is to be used for definitions. However, due to the generality of the OLP-formalism, one can easily construct senseless definitions. Consider the following definition for p:

$\{ \; p \text{ :- } \neg p \; \}$

Under a 2-valued completion semantics, an open logic program containing such definition would be inconsistent. In 3-valued completion semantics, such a program remains consistent; yet the badly defined facts will have truth value **u** in all or a subclass of the models. This use of **u** seems a better, more permissive solution to deal with badly constructed definitions. At the same time, this shows that **u** is an error condition and not a truth value. **u** should be interpreted as *badly defined*. $\mathcal{H}_M(p(x_1, \ldots, x_n)) = \mathbf{u}$ means that the definition of p fails to describe the truth value of $p(x_1, \ldots, x_n)$ in M. A 3-valued interpretation for an open

predicate would not make sense under this interpretation of **u**: open predicates have no definition and cannot be badly defined[1].

For the purposes of this paper, one interesting advantage of using the 3-valued completion semantics is that it is the weakest semantics known for the (O)LP-formalism. In [4], the following theorem is proven:

Theorem 3.2 *If M is a model of P^D wrt (2-valued completion semantics [2] [3]) (generalised stable semantics [14]) (generalised well-founded semantics [22]) (justification semantics [5]) then M is a model of P^D wrt 3-valued completion semantics.*

As a consequence, the 3-valued completion semantics induces the weakest entailment relation $\models$: if an open theory entails F according to the 3-valued completion semantics then also wrt to the other semantics. In intuitive terms: the declarative meaning of an open logic program under the 3-valued completion semantics is a safe approximation of its meaning under these other semantics. As a consequence, many of the applications of OLP-FOL under 3-valued completion semantics described in this paper are preserved under the stronger semantics.

4 Knowledge Representation in OLP-FOL.

Crucial for knowledge representation in OLP-FOL is to distinguish assertional knowledge and *terminological* or *definitional* knowledge. Below, we explore this distinction in a number of problems. The problems include different forms of uncertainty: uncertainty on predicates, null values and uncertainty on the domain of discourse. First, we further explore the difference between the modal and the terminological interpretation.

4.1 A modal knowledge example.

Recall the example in section 1. The following definition for *dead*:

$dead(X)$:- $\neg alive(X)$

in an open program is interpreted as the phrase *X is dead iff X is not alive*. In contrast, the program clause in Extended Logic Programming (ELP):

$dead(X)$:- *not* $alive(X)$

under the modal interpretation is read as *X is dead if it is not believed that X is alive*. The program clause with strong negation:

[1] In addition, the introduction of 3-valued open predicates would cause disastrous technical problems. One easily verifies that in such a relaxed form of 3-valued completion semantics, an interpretation I such that $\mathcal{H}_I(A) = \mathbf{u}$ for all atoms A, would be a model of P^D. However, no FOL formula is satisfied in I. As a consequence, the tautologies of classical logic would not longer be satisfied in OLP-FOL and it would be impossible to embed classical logic in OLP-FOL. The assertional component of OLP-FOL would loose its expressive power. None of the applications that are described further on in this paper would be maintained.

$dead(X) \text{ :- } \neg alive(X)$

under the modal interpretation is read as *X is dead if it is believed*[2] *that X is not alive.*

None of the representations in ELP have the same meaning as the representation in OLP-FOL. The *not* and $\neg$ of ELP have both a different modality than $\neg$ in FOL and in OLP-FOL. In the above example, only the OLP-FOL representation correctly represents our knowledge on the relationship between dead and alive. In other examples, the modal interpretation is the correct one. We recall an example from [10].

In ELP, one can represent the behaviour of a dare-devil who crosses a railroad if he has no evidence that there is a train coming by the following program clause:

$cross \text{ :- } not\ train$

Under the modal interpretation, the logic program P_1 consisting of the above program clause contains no information whether there is a train or not, so *cross* should be true. Under answer set semantics this is correctly modelled: the unique answer set of P_1 is $\{cross\}$.

The behaviour of a more careful person who crosses if he believes that no train is coming can be represented using strong negation:

$cross \text{ :- } \neg train$

The logic program P_2 consisting only of the above program clause contains no information whether there is a train or not, so *cross* should not be true. Under answer set semantics this is correctly modelled: the unique answer set of P_2 is $\{\}$[3].

Compare this with the OLP approach. Consider the set $P = \{cross \text{ :- } \neg train\}$. There are two open logic programs which include it: one in which only *cross* is defined ($D_1 = \{cross\}$) and another in which both *cross* and *train* are defined ($D_2 = \{cross, train\}$).

P^{D_1} contains no information on whether the train comes or not. P^{D_2} contains full information on *train* because *train* has an empty definition. Both P^{D_1} and P^{D_1} share the definition of *cross*. This definition asserts that whether a person knows that a train is coming or not, he crosses iff there is no train coming. Clearly, this definition does not correspond to the behaviour of the dare-devil nor of the more normal person.

In our opinion, this example shows clearly the different modalities of *not* and $\neg$ in Extended Logic Programming (ELP) and of $\neg$ in classical logic and OLP-FOL. It shows also that both interpretations are of value. We believe that OLP-FOL is not suited to represent modal knowledge such as in the train example, while the modal interpretation is not suited to represent definitional information.

[2]Though [10] interprets the explicit negation $\neg$ in ELP as classical negation, various mappings of extended logic programs to autoepistemic logic have shown that it is more natural to interpret $\neg p$ as *it is known that p is not true* [18]. An answer set represents a set of beliefs. A literal $\neg p$ in an answer set is interpreted as *it is believed that p is not true.* It seems natural to carry over this interpretation to the literals in the program.

[3]Note that P_2 does not contain information about what to do if it is unknown whether a train comes or not. The program clause $\neg cross \text{ :- } not \neg train$ represents the behaviour that no crossing happens if it is not believed that the train does not come. P_2 augmented with this program clause has the unique answer set $\{\neg cross\}$.

4.2 Temporal uncertainty.

The representation of uncertainty has been investigated most intensively in the context of temporal domains. Here we present solutions for a number of benchmark problems using an OLP-FOL version of event calculus [16]. The event calculus provides a more natural representation for reasoning on the *actual time-line*, compared to situation calculus which needs to be extended to reason on an actual time-line [23] [15]. The solutions of the benchmark problems below are similar to the abductive logic programs in [8]. However,[8] ignores the declarative aspects of the representation and focusses on the abductive reasoning.

A fundamental assumption in event calculus is that the state of the world at any point in time is determined completely by the initial state and the effects of actions in the past. Intuitively, a property P is true at a certain moment T if it is initiated by some earlier event and is not terminated since then. This knowledge can be interpreted as a definition for the predicate $holds\_at(P,T)$.

$holds\_at(P,T)$:- $happens(E), E < T, initiates(E,P), \neg clipped(E,P,T)$
$holds\_at(P,T)$:- $initially(P), \neg clipped(P,T)$
$clipped(E,P,T)$:- $happens(C), E < C < T, terminates(C,P)$
$clipped(P,T)$:- $happens(C), C < T, terminates(C,P)$

$clipped(E,P,T)$ is an auxiliary predicate which intuitively means that the property P is terminated during the interval $]E,T[$. $clipped(P,T)$ means that the property P is terminated before T. Interesting is that this definition for *holds_at* shows that *definitional knowledge* is not necessarily *terminological knowledge*: the definition for *holds_at* can hardly be seen as the definition of *technical vocabulary* of an expert. Here, this definition is used to represent an empirical observation relating the state of fluents with the occurrence of past actions.

In [8], we defend the view of event calculus as a linear time theory, in contrast to situation calculus which has branching time. That $<$ is a linear order is formally assured by adding the theory of linear order as FOL assertions:

$X < Y \wedge Y < Z \rightarrow X < Z$	transitivity
$\leftarrow X < Y, Y < X$	asymmetry, irreflexivity
$happens(X) \wedge happens(Y) \rightarrow X < Y \vee X = Y \vee Y > X$	linearity

For simplicity, we assume here that two events cannot happen simultaneously.

The definitions of *holds_at* and *clipped* rely on the predicates *initially*, *happens*, $<$, *initiates* and *terminates*, which describe the initial situation, the events, their order and their effects. Depending on the given information, different subsets of these predicates can be defined. We illustrate this in a number of well-known variants of the notorious Turkey shooting problem. In all these problems the effects of load, shoot and wait actions can be represented as definitions for *initiates* and *terminates*:

$initiates(E, loaded)$:- $act(E, loading)$
$terminates(E, loaded)$:- $act(E, shooting)$
$terminates(E, alive)$:- $act(E, shooting), holds\_at(loaded, E)$

In the Turkey shooting problem, the other predicates $initially/1$, $happens/1$, $</2$, $act/2$ can be defined by exhaustive enumeration:

> $initially(alive), happens(e1), happens(e2), happens(e3), happens(e4), act(e1, load)$, $act(e2, wait), act(e3, shoot)$ and the transitive closure of $e1 < e2 < e3 < e4$.

The resulting OLP-FOL theory $\mathcal{T}_{TS}$ defines all predicates. The FOL axioms in the theory of linear order in $\mathcal{T}_{TS}$ are entailed by the definitions of *happens* and $<$ and can be dropped from $\mathcal{T}_{TS}$. The resulting closed logic program is a correct representation of the problem specification and entails $\neg holds\_at(alive, e4)$.

A number of variants show various forms of uncertainty. An interesting variant is the Russian Turkey shooting problem, in which an indeterminate event of spinning the gun's chamber takes place instead of waiting. The effect of this event is that it potentially unloads the gun. A theory representing this problem should be satisfiable with both $holds\_at(alive, e4)$ and $\neg holds\_at(alive, e4)$.

The indeterminacy of the spinning action complicates the formulation of a definition for *terminates*. One possibility is to leave the predicate open and to represent all knowledge on it as FOL assertions. But it is significantly simpler, clearer and more concise to give a definition of *terminates* in which a new open predicate $good\_luck/1$ is used which captures the indeterminacy. The resulting open logic program $\mathcal{T}_{RTS}$ is obtained from $\mathcal{T}_{TS}$ by replacing the program clause $act(e2, wait)$ by $act(e2, spin)$ and adding the following clause to the definition of terminates/2:

$$terminates(E, loaded) \text{ :- } act(E, spinning), holds\_at(loaded, E), good\_luck(E)$$

Both $holds\_at(alive, e4)$ and $\neg holds\_at(alive, e4)$ are satisfiable with $\mathcal{T}_{RTS}$.

Another variant is the Murder Mystery, in which a shooting event $e1$ followed by a waiting event $e2$ occur and the turkey is found dead at a next event $e3$. The initial state is unknown. A desired conclusion is that if the turkey is initially alive then the gun is initially loaded.

For this problem, all predicates except $initially/1$ can be defined. The definitions of *holds_at*, *initiates*, *terminates* are as in $\mathcal{T}_{TS}$. The predicates *happens*, $<$ and *act* can be defined by exhaustively describing the scenario:

> $happens(e1)$, $happens(e2)$, $happens(e3)$, $act(e1, shoot)$, $act(e2, wait)$, and the transitive closure of $e1 < e2 < e3$.

The knowledge that the turkey is dead at $e3$ is formulated as a simple FOL assertion $\neg holds\_at(alive, e3)$. The resulting theory $\mathcal{T}_{MM}$ provably entails[4]:

$$initially(loaded) \vee \neg initially(alive)$$

A variant of the murder mystery illustrates nicely the distinction between *definitional knowledge* and *assertional knowledge* and how these forms of knowledge are properly represented in OLP-FOL. Assume that the same events happen as in the murder mystery but that

[4]Note that again, the definition of $<$ entails all axioms of the theory of linear order. Hence, these axioms can be dropped from $\mathcal{T}_{MM}$.

the turkey is found alive instead of dead at $e3$. Desired conclusions here are that initially the turkey is alive and that the gun is unloaded. The question now is: should we add the fact $holds\_at(alive, e3)$ to the open logic program or as a FOL axiom?

Adding the atom to the open logic program boils down to extending the definition of $holds\_at$. It is clear that event calculus' basic assumption -the state is determined by initial state and effects of past actions- is not falsified in this example, so there seems no a priori reason why the definition should be changed. Stronger even, adding the atom as a program clause would alter the definition in a way which really contradicts our intuition: according to the extended definition, it would be possible that the fluent *alive* can originate at $e3$ as by deus ex machina, without being caused by past actions[5]. This is evidenced by the behaviour of the corresponding program: the open logic program consisting of the program clauses of $\mathcal{T}_{MM}$ augmented with the program clause $holds\_at(alive, e3)$:- is satisfiable with $\neg initially(alive)$.

The correct representation is the theory $\mathcal{T}_{MM'}$ obtained from $\mathcal{T}_{MM}$ by substituting $holds\_at(alive, e3)$ for $\neg holds\_at(alive, e3)$ as a FOL assertion. $\mathcal{T}_{MM'}$ entails $initially(alive) \wedge \neg initially(loaded)$.

In all previous examples, the theory of linear order is entailed by the program and can be dropped safely. The theory of linear order comes into play when there is uncertainty about the events and/or their order. Take a scenario in which the gun is initially loaded (with one bullet) and the turkey is alive; at $e3$ the gun is fired. Before $e3$, two different events $e1$ and $e2$ take place at which the gun's chamber is turned for 180 degrees. The order of $e1$ and $e2$ is unknown. Note that no matter which is the order of $e1$ and $e2$, the gun's chamber is turned for 360 degrees, so it is loaded when the shooting event takes place. Therefore, a desired conclusion in this problem is that after the shooting the turkey is dead.

To represent this problem, a new action *turn* is introduced. When the gun is loaded, then *turn* removes the bullet from the barrel. The resulting state is called *opposite*. Again turning moves the bullet in the barrel and loads the gun. The definitions of *initiate* and *terminates* in $\mathcal{T}_{TS}$ are extended with:

$terminates(E, loaded)$:- $act(E, turn), holds\_at(loaded, E)$
$terminates(E, opposite)$:- $act(E, turn), holds\_at(opposite, E)$
$initiates(E, loaded)$:- $act(E, turn), holds\_at(opposite, E)$
$initiates(E, opposite)$:- $act(E, turn), holds\_at(loaded, E)$

The predicates *initially*, *happens*, *act* are defined by enumeration:

$initially(alive), initially(loaded), happens(e1), happens(e2), happens(e3), happens(e4),$
$act(e1, turn), act(e2, turn), act(e3, shoot).$

The remaining problem is the representation of the knowledge on $<$. It is tempting to add the partial order of known atoms of $<$ as a definition for $<$ to the open logic program. This approach for the representation of this form of uncertainty is proposed in [16]. The

[5]Note here the non-monotonicity of the concept of a definition.

resulting logic program P_{LTTS}[6] contains definitions for all predicates. Note that the OLP-FOL theory consisting of P_{LTTS} augmented with the theory of linear order is inconsistent: P_{LTTS} violates the linearity constraint since it entails $\neg e1 < e2 \wedge \neg e2 < e1 \wedge \neg e1 = e2$. In [21], examples were given where this approach failed. Also in this case the approach fails: P_{LTTS} entails $holds\_at(alive, e4)$ and Prolog can be used to prove it. In more detail, one easily verifies that P_{LTTS} entails $holds\_at(loaded, e1)$ and $holds\_at(loaded, e2)$. Therefore, both $e1$ and $e2$ unload the gun, so that $\neg holds\_at(loaded, e3)$ is entailed. Since the gun is unloaded at $e3$, the shooting has no effect and the turkey remains alive.

A correct solution can be represented in OLP-FOL. Since there is uncertainty about $<$, this predicate cannot be defined. The correct theory $\mathcal{T}_{LTTS}$ is obtained from P_{LTTS} by dropping the definition of $<$ and adding the theory of linear order and the following atoms as FOL assertions:

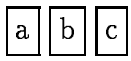
$$e1 < e3,\ e2 < e3,\ e3 < e4^{7}$$

$\mathcal{T}_{LTTS}$ correctly entails $\neg holds\_at(alive, e4)$.

Interesting applications of uncertainty on the events and their order in event calculus are planning problems. The goal of planning is to find an ordered set of events which produces a given desired final state Q; the events are a priori unknown: in an OLP-FOL theory $\mathcal{T}$ representing a planning domain, $happens/1$, $< /2$ and $act/2$ have no definition. An abductive procedure can be used to find a set Δ of definitions for these predicates such that the theory $\mathcal{T} + \Delta \models Q$ [9].

4.3 Moore's example: a null value.

Moore gives the following example [**?**]. Three blocks, a, b and c, arearranged as shown:



The colour of a is green, c is blue and the colour of b is unknown. A desired conclusion is that there is a green block adjacent to a non-green block[8].

For simplicity, we assume that a, b and c are the only blocks and green and blue are the only colours. Due to this assumption, *block* and *colour* can be defined by exhaustive enumeration. The predicates *next* and *adjacent* can be defined as well.

$block(a)$:-
$block(b)$:-
$block(c)$:-
$colour(blue)$:-
$colour(green)$:-

[6] LTTS stands for "Load Turn Turn Shoot".

[7] Note that this set of facts is not transitive. Due to the presence of the of linear order, additional atoms like $e1 < e4$ are entailed.

[8] Moore's aim was to defend logic by showing that the desired conclusion can be obtained only by a reasoning by 2 cases (either b is green or not), a form of reasoning formalised only in logic.

$next(a, b)$:-
$next(b, c)$:-
$adjacent(X, Y)$:- $next(X, Y)$
$adjacent(X, Y)$:- $next(Y, X)$

The fact that the colour of b is unknown poses a problem to define *has_colour*. An elegant solution would be to represent the colour of b by a *null value cb* of type *colour*. [27] formalises null values in classical logic. This approach exploits the fact that in classical logic, a constant represents some unique object of unknown identity. Unfortunately, this approach does not work in OLP-FOL due to the fact that OLP-FOL is a free equality logic. The reason is clear: the FOL assertion $colour(cb)$ and the definition of *colour* entail that *cb* is either *green* or *blue* which is excluded by the free equality. However, there is an easy way to simulate null values in a free equality logic. We introduce a new open predicate $colour\_of\_b/1$ which -intuitively- represents the singleton $\{cb\}$. Using this predicate, *has_colour* can be defined as follows:

$has\_colour(a, green)$:-
$has\_colour(c, blue)$:-
$has\_colour(b, C)$:- $colour\_of\_b(C)$

The assertional knowledge that $colour\_of\_b/1$ contains a unique *null value* of type *colour* is represented by the FOL assertions[9]:

$\exists!C.colour\_of\_b(C)$
$\forall C.colour\_of\_b(C) \rightarrow colour(C)$

The result is an OLP-FOL theory $\mathcal{T}_{Moore}$ with one open predicate $colour\_of\_b/1$. The theory provably entails that there is a block which is not green adjacent to a green block.

4.4 The third man: an open domain problem.

The scenario of this simple problem is as follows. Bob is found murdered in his cottage while being visited by John. Suicide is excluded and John seems to have an alibi but which is hard to verify. The conclusions that can be derived in this domain clearly depend on whether it is known or not that Bob and John were the only persons in the cottage. Only if this knowledge is given, it can be concluded that John has a false alibi and is the murderer.

The following open theory $\mathcal{T}_{3M}$ represents this scenario:

$D = \{killed, alibi\}$
$P = \{ \quad alibi(john)$:-
$\qquad killed(X, bob)$:- $murderer(X) \quad \}$
$T = \{ \quad \exists X.\neg X = bob \wedge killed(X, bob)$
$\qquad \forall X.murderer(X) \wedge alibi(X) \rightarrow false\_alibi(X) \quad \}$

[9] $\exists!X.F[X]$ is the standard notation for $\exists X.F[X] \wedge \forall X_1, X_2.F[X_1] \wedge F[X_2] \rightarrow X_1 = X_2$.

Open predicates are *murderer* and $false\_alibi$.

Just as classical logic, OLP-FOL is an *open domain logic*: the domain closure assumption (DCA) is not a priori imposed. The theory $\mathcal{T}_{3M}$ has models in which *bob* and $john$ are the only domain elements. In these models, $false\_alibi(john)$ holds. There are other models in which other domain elements appear and where both facts are false. In these models, $murderer(x)$ is true for some domain element x not represented by $john$.

The example clarifies the role of general interpretations versus Herbrand interpretations. OLP-FOL (just like FOL) does not impose the DCA, due to the fact that general non-Herbrand interpretations are allowed. A logic with a model semantics based on Herbrand interpretations entails the DCA automatically. In many applications, these axioms are naturally satisfied in the problem domain; in other applications, they are not. From the knowledge representation point of view, basing OLP-FOL on Herbrand interpretations would restrict the expressivity in an undesirable way. If in the example, it is known that Bob and John are the only persons, then this must be represented explicitly by adding the DCA[10]:

$$\forall X.X = bob \vee X = john$$

The resulting theory entails $murderer(john)$.

An interesting class of domains where the domain closure assumption cannot be made are planning domains in the context of event calculus. In such domains, the events are a priori unknown and hence, the DCA cannot be added. Such domains are less easily described in a logic with a Herbrand-based semantics.

4.5 A default knowledge example.

OLP-FOL does not contain default rules nor default connectives and hence, is not suitable for representing defaults. Nevertheless, *default reasoning* on an OLP-FOL theory is not excluded. We illustrate this with the well-known quaker-republican problem: *Nixon is a quaker and a republican. Normally, quakers are doves, republicans hawks. It is impossible to be both hawk and dove.* To represent defaults, it is essential that the number of applications of defaults is maximised, or equivalently that the number of exceptions to defaults is minimised. This view is found in default logic [26] and in circumscription [20]. To maintain consistency in the quaker-republican problem, one of the two defaults must be violated since Nixon cannot be both hawk and dove. Minimising the number of exceptions to default rules, only one default is allowed to be violated and one concludes that Nixon is either a hawk or a dove.

With respect to the representation in OLP-FOL, note that this example contains no definitional information: the concepts quaker, republican, dove, hawk cannot be defined using the above specification (unless we would assume that Nixon is the only republican and quaker). (OLP-)FOL does not provide default rules; therefore, we believe that the most precise formalisation in OLP-FOL would be the following FOL theory:

[10]It is well-known that when Σ contains functors with arity > 0, the DCA can not be represented correctly in FOL. As was shown in [5], the DCA can be represented correctly in OLP-FOL under stronger semantics than the 3-valued completion semantics.

$$T_1 = \{\ republican(nixon)$$
$$\quad quaker(nixon)$$
$$\quad \forall X. \neg hawk(X) \vee \neg dove(X) \quad \}$$

in which the two defaults are not represented.

However, [24] argued that though FOL is unsuited to represent *default knowledge*, *default reasoning* on FOL theories is possible. Following this approach, abnormality predicates ab_rep and ab_quak are introduced and two FOL implications are added to "represent" the defaults.

$$T_2 = T_1 \cup \{\ hawk(X) \leftarrow republican(X), \neg ab\_rep(X)$$
$$\quad dove(X) \leftarrow quaker(X), \neg ab\_quak(X) \quad \}$$

As a representation of the defaults, T_2 is a failure: there is no way to express in FOL that lesser or minimal interpretations of ab_rep and ab_quak are preferred. Therefore, the information content of T_1 and T_2 -restricted to the symbols of T_1- is the same. It is easy to prove that T_2 is a conservative extension [29] of T_1 and hence the formulas that do not contain ab_rep and ab_quak and that can be proven for T_2 are precisely those that can be proven from T_1.

However, [24] puts the burden of dealing with the defaults not on the information content of the theory but on the reasoning - not on the level of meaning but on the level of reasoning. More precisely, [24] argues that default reasoning is inherent to hypothetical or abductive reasoning; the default aspect of hypothetical reasoning lies in the fact that the set of explaining hypotheses is minimised. Such a procedure can be applied even to a monotonic logic.

Under this -weaker- view, it does not matter that T_2 does not represent the defaults in the strong sense. The default reasoning becomes alive when for example an abductive procedure is called to explain why Nixon does not violate the constraint. A suitable procedure will compute minimal explanations; it will raise the hypothesis that Nixon is an abnormal quaker or that he is an abnormal republican, but not both.

5 Conclusions and future work.

The above examples show the expressivity of OLP-FOL for knowledge representation. The elegance of the above presented solutions may easily mislead. Many of the temporal reasoning problems in section 4.2 are considered as difficult knowledge representation problems and have been used to show problems with several temporal reasoning proposals. The representation of null values and open domain knowledge is still an open question in logic programming and, in the case of ELP, seems to require substantial changes to the semantics [30] [12]. Other experiments further confirm the expressivity of OLP-FOL. [7] presents a sound and complete transformation of the temporal reasoning language $\mathcal{A}$ (which allows to represent uncertainty on the initial state) to OLP-FOL. In comparison, the transformation from $\mathcal{A}$ to ELP proposed in [11] is not sound in general and not complete. In ongoing work, we mapped Reiter's situation calculus [28] to OLP-FOL (under a stronger semantics than the

one used in this paper), despite the fact that this theory contains a second order induction axiom. In another experiment, a terminological language was mapped to OLP-FOL.

An interesting issue which falls outside the scope of this paper is the representation of *inductive definitions* in the OLP-FOL formalism. OLP-FOL (under completion semantics) nor FOL are suited to represent inductive definitions; in ongoing work we showed that under stronger semantics such as justifications semantics [5], inductive definitions can be correctly represented in an open logic program. As a consequence, OLP-FOL under justification semantics significantly extends the FOL expressivity. This extra expressivity allows to represent the Domain Closure Axiom or the second order induction axiom in situation calculus. Another interesting issue is how theorem provers and problem solvers for OLP-FOL can be developed. In [6], we show how existing abductive extensions of SLDNF resolution can be used for reasoning on OLP-FOL theories, not only to solve abductive problems but also for deductive and satisfiability checking problems.

To conclude, the OLP-FOL logic provides an alternative interpretation of (abductive) logic programming. The auto-epistemic and default interpretation on one hand and the terminological interpretation on the other hand are different interpretations which assign a different modality to negation in logic programs. In OLP-FOL, negation has the objective modality of negation in classical logic. As a consequence, it makes no sense to add *classical negation* to OLP-FOL. On the contrary, it would be interesting to add modal forms of negation, as found in the auto-epistemic interpretation. Such an extension would allow to represent also modal problems (e.g. the train example in section 4.1) and would combine the representational power of ELP and OLP-FOL. The problems of such an integration don't seem insurmountable. Last but not least, OLP-FOL is a declarative integration of logic programming and classical logic and opens the possibility of cross fertilisation between both fields.

Acknowledgements

I thank Danny De Schreye and Kristof Van Belleghem for many valuable discussions. Danny, Kristof and Marion Mircheva gave justified comments on earlier drafts. Hector Levesque pointed me to the analogy in the motivations for OLP-FOL and terminological languages.

References

[1] R. J. Brachman and H.J. Levesque. Competence in Knowledge Representation. In *Proc. of the National Conference on Artificial Intelligence*, pages 189–192, 1982.

[2] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.

[3] L. Console, D. Theseider Dupre, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[4] M. Denecker. *Knowledge Representation and Reasoning in Incomplete Logic Programming*. PhD thesis, Department of Computer Science, K.U.Leuven, 1993.

[5] M. Denecker and D. De Schreye. Justification semantics: a unifying framework for the semantics of logic programs. In *Proc. of the Logic Programming and Nonmonotonic Reasoning Workshop*, pages 365–379, 1993.

[6] M. Denecker and D. De Schreye. A terminological interpretation of (Abductive) Logic Programming. Draft, K.U.Leuven, 1994.

[7] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. *Journal of Logic and Computation, to appear*, 1994.

[8] M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In *Proc. of the European Conference on Artificial Intelligence*, 1992.

[9] K. Eshghi. Abductive planning with Event Calculus. In R.A. Kowalski and K.A. Bowen, editors, *Proc. of the International Conference on Logic Programming*, 1988.

[10] M. Gelfond and V. Lifschitz. Logic Programs with Classical Negation. In D.H.D. Warren and P. Szeredi, editors, *Proc. of the 7th International Conference on Logic Programming 90*, page 579. MIT press, 1990.

[11] M. Gelfond and V. Lifschitz. Representing Action and Change by Logic Programs. *Journal of Logic Programming*, 17(2,3,4):301–322, 1993.

[12] M. Gelfond and H. Przymusinska. Reasoning in Open Domains . In *Proc. of Logic Programming and nonmonotonic reasoning workshop*, page 397, 1993.

[13] A. C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

[14] A.C. Kakas and P. Mancarella. Generalised stable models: a semantics for abduction. In *Proc. of the European Conference on Artificial Intelligence*, 1990.

[15] R. Kowalski and F. Sadri. The Situation Calculus and Event Calculus Compared. In *Proc. of International Logic Programming Symposium*. MIT Press, 1994.

[16] R.A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(4):319–340, 1986.

[17] K. Kunen. Negation in Logic Programming. *Journal of Logic Programming*, 4:231–245, 1989.

[18] V. Lifschitz and G. Schwarz. Extended Logic Programs as Autoepistemic Theories. In L. Pereira and A. Nerode, editors, *Proc. of the Logic Programming and Non-monotonic Reasoning Workshop*, 1993.

[19] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[20] J. McCarthy. Circumscription - a form of nonmonotonic reasoning. *Artifical Intelligence*, 13:89–116, 1980.

[21] L.R. Missiaen. *Localized abductive planning with the event calculus.* PhD thesis, Department of Computer Science, K.U.Leuven, 1991.

[22] L.M. Pereira, J.N. Aparicio, and J.J. Alferes. Hypothetical Reasoning with Well Founded Semantics. In B. Mayoh, editor, *Proc. of the 3th Scandinavian Conference on AI.* IOS Press, 1991.

[23] J. Pinto and R.Reiter. Temporal Reasoning in Logic Programming: A Case for the Situation Calculus. In *Proc. of the International Conference on Logic Programming*, pages 203–221, 1993.

[24] D. Poole. A Logical Framework for Default Reasoning. *Artifical Intelligence*, 36:27–47, 1988.

[25] R. Reiter. On Closed World Data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, 1978.

[26] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[27] R. Reiter. On integrity constraints. In M. Vardi, editor, *Proc. of Conf. on Theoretical Aspects o Reasoning about Knowledge*, pages 97–111, 1988.

[28] R. Reiter. Formalizing Database Evolution in the Situation Calculus. In *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 600–609, 1992.

[29] J. Shoenfield. *Mathematical Logic.* Addison-Wesley, Reading, Mass., 1967.

[30] B. Traylor and M. Gelfond. Representing Null Values in Logic Programming. In *Proc. of the ILPS'93 workshop on Logic Programming with Incomlete Information*, pages 35–47, 1993.