Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents

Robert Kowalski and Fariba Sadri
Imperial College London,

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

Confusions

Russell and Norvig 2003 view production rules as just logical conditionals used to reason forward (page 286).

Thagard, in *Mind: Introduction to Cognitive Science* 2005, argues that "Rules are if-then structures …very similar to the conditionals…, but they have different representational and computational properties." (page 43).

Simon, in the *MIT Encyclopedia of Cognitive Science*, includes Prolog "among the production systems widely used in cognitive simulation."

Rao, characterises AgentSpeak as "very similar to SLD-resolution of logic programming languages", but ignores declarative semantics.

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

Production Systems in Practice

Three kinds of rules:

reactive rules
missing in logic programming

forward chaining logic rules
possible with logic programs

goal-reduction rules
typical of logic programming

Production system cycle
missing in logic programming

Destructively changing database
missing in logic programming

Reactive rules implement stimulus-response associations

Typically have implicit or *emergent* goals:

if a car coming towards you
then get out of its way

has the implicit goal

to stay safe.

Forward chaining logic rules

Give the impression that production rules are just conditionals used to reason forward:

if X is human then X is mortal.

Use forward chaining to implement forward reasoning.

Goal-reduction rules

Thagard claims that

> "Unlike logic, rule-based systems can also easily represent strategic information about what to do. Rules often contain actions that represent goals, such as
>
> *IF you want to go home for the weekend and you have the bus fare, THEN you can catch a bus*." (page 45)."

In logic programming, strategic rules are obtained by backward reasoning:

> *you go home for the weekend*
> *if you have the bus fare and you catch a bus.*

The two most influential cognitive models, SOAR and ACT-R, mainly use production rules for goal-reduction.

Five reasons for integrating production rules (PRs) and logic programs (LPs):

1. Eliminate the overlap between forward logic rules in PRs and forward/declarative clauses in LP.

2. Eliminate the overlap for goal-reduction.

3. Provide a better separation between goals and facts.

4. Provide a declarative semantics for PRs and for the combination of LP and PRs.

5. Provide a cycle and destructive database of facts, missing in LP.

Other approaches

In the majority of other approaches, production rules (PR) are mapped into LP.

To our knowledge, there has been no work that combines both LP and PR *side-by-side*.

Zaniolo and Statelog use a situation calculus-like representation with frame axioms, and reduce PRs and ECA rules to LPs. Both suffer from the frame problem.

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

ALP as a first step in combining logic programs and production rules

Integrity constraint: *If there is an emergency then I get help.*

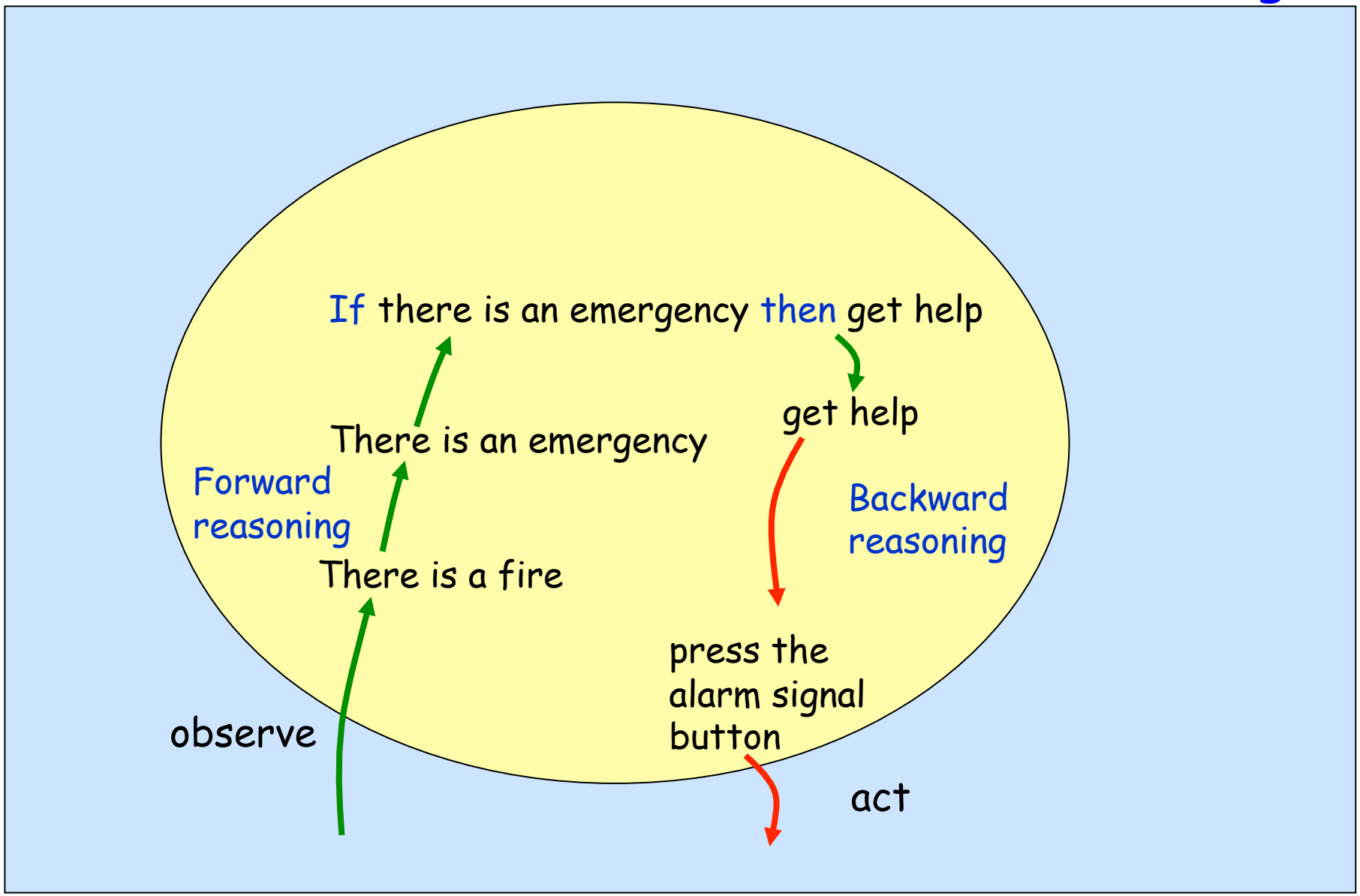
Logic program:

A person gets help if the person presses the alarm signal button.
There is an emergency if there is a fire.
There is an emergency if one person attacks another.
etc.

Abducible predicates: *there is a fire,*
a person presses the alarm signal button

Observation: *There is a fire.*
Forward reasoning: *There is an emergency.*
Forward reasoning, Goal: *I get help.*
Backward reasoning, Solution: *I press the alarm signal button.*

ALP combines forward and backward reasoning



ALP gives a logical semantics to production rules.

Reactive rules represented by integrity constraints.

Forward chaining logic rules represented by LP clauses.

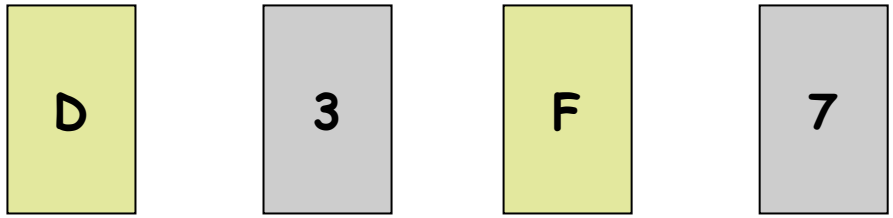
Goal-reduction rules represented by LP clauses.

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

Wason Selection Task

Four cards, letters on one side, numbers on the other.



Select those and only those cards that need to be turned over, to determine whether the following conditional is true:

If there is a D on one side,
then there is a 3 on the other side.

Only 5-10% of all people select the right cards.

Wason selection task

Four people.
Determine whether the following rule holds:

> *If a person is drinking beer in a bar,*
> *then the person is over eighteen.*

Most people get the right answer.

Explanation?

People don't use logic, but have evolved a cheater detection scheme :

> *If you receive a benefit,*
> *then you must meet its requirement.*

Wason Selection Task
Alternative Explanation

Cheng and Holyoak: people reason in classical logic when conditionals involve *deontic* notions of obligation and prohibition.

Stenning and van Lambalgen: to solve the selection task 1) interpret the conditional and 2) reason with the interpretation.

> Subjects interpret the conditional in card versions of the task as logic programs. This entails the converse of conditionals and inhibits contrapositives.
>
> Subjects interpret the conditional in bar versions of the task in deontic logic.

Arguably, the deontic interpretation can be obtained more simply by interpreting conditionals as integrity constraints.

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

BDI agents

Originally specified in multi-modal logics,
with modal operators for goals, beliefs and intentions.

AgentSpeak and its successors abandoned modal logic.
Programs (called plans) generalise production rules:

Event E: conditions C ⇐ goals G and actions A.

Plans manipulate a database, like the working memory in production systems. The database contains both *belief literals* and *goal atoms*.
(Some recent BDI agents include logic programs as beliefs.)

Belief literals and goal atoms are added and deleted destructively.

Goals in BDI agents

Some BDI agent systems store goals in a goal stack.
(Also SOAR and ACT-R.)

Goal atoms represent achievement goals.
Maintenance goals are represented implicitly as "plans".

Maintenance goals have the form:

If certain conditions then achieve certain conclusions.

or equivalently:

Make the following conditional true:

If certain conditions then certain conclusions.

Maintenance goals are important in:

Management Science
Software Engineering.

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

Deductive databases

A deductive database can be thought of as a set of beliefs.

Integrity constraints have the same semantics as queries.

> *Ad hoc* queries are concerned with properties that hold in a given state of the database.
>
> Integrity constraints are persistent queries that are intended to hold in all states of the database.

Deductive databases

Deduction rule in a database:

If X is a bank manager
then X is a bank employee.

Integrity Constraint:

If X is a bank employee
then for some integer Y
X has social security number Y.

Semantics of integrity constraints

In relational databases, an integrity constraint IC is satisfied
if the IC is *true* in the database regarded as a Herband model.

In deductive databases, an IC is satisfied

in the *consistency view*,
if IC is consistent with the completion of the database.

in the *theorem-hood view*,
if IC is a theorem, entailed by the completion.

in the *epistemic view* ,
if the database knows the ICs are true.

Semantics of integrity constraints

For example, in the epistemic view:

If X is a bank employee
then for some integer Y
X has social security number Y

is interpreted as:

If the database knows that X is a bank employee
then for some integer Y
the database knows that X has social security number Y

Reiter showed all three views are equivalent in many cases for databases augmented with the *closed world assumption*.

In Horn clause databases DB, the three views are equivalent to the view that an IC is satisfied if it is *true* in the unique minimal model of DB.

Integrity checking

The obvious way to check an integrity constraint IC is to evaluate IC as a query.

But evaluating IC in a new state duplicates much of the work of evaluating IC in the previous state.

To improve efficiency, checking IC in a new state can assume that IC is satisfied in the previous state, using forward reasoning to focus on the update.

The combination of a sequence of database updates and forward reasoning resembles the production system cycle.

Note. This kind of integrity checking applies only to static Ics.

Integrity checking in the SK procedure

In the Sadri-Kowalski (SK) procedure, ICs are formalised as denials, e.g.:

> *If X is a bank employee and not X has a social security number Y then false .*

If an update matches one of the conditions of a clause or integrity constraint, forward reasoning (via resolution) generates the resolvent.

SLDNF is used to try to verify the remaining conditions of the resolvent.
If the conditions are verified,
then the instantiated conclusion is added as a new update.

If the new update is *false*,
then the procedure terminates, and integrity has been violated.

Otherwise, the procedure is repeated with the new update.

If the procedure does not generate false
then the transaction satisfies the integrity constraints.

Note. SK might not terminate.

Integrity checking in the SK procedure

ICs: *If P and not R then false*
If S and Q then false

Database: *S if M*
Q if T
R if T

Updates: *{P, T}*

Forward reasoning:

If not R then false
Q
If S then false.

The SLDNF evaluation of the two conditions *not R* and *S* fails finitely, and therefore SK terminates successfully.

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

Abductive Logic Programs <P, A, IC>

P a normal logic program.

A a set of abducible predicates.

IC a set of integrity constraints.

(equivalent to first-order sentences)

Typically, ICs are expressed as denials:

not (A1 &...& An & not B)

or as conditionals:

If A1 &...& An then B.

Normally, predicates in A are not allowed to occur in conclusions of clauses in P (without loss of generality).

ALP can be used to explain observations

P: *Grass is wet if it rained.*
Grass is wet if the sprinkler was on.
The sun was shining.

A: *it rained,* *the sprinkler was on*

IC: *If it rained and the sun was shining then false.*

Observation: *Grass is wet*

Candidate explanations: *it rained,* *the sprinkler was on*

The only explanation that satisfies the integrity constraint is

the sprinkler was on.

ALP Semantics and Proof Procedures

Semantics:

Given $< P, A, IC >$, an abductive explanation for a goal G (or observation) is a set Δ of ground atoms in predicates A such that:

G holds in $P \cup \Delta$
IC holds in $P \cup \Delta$.

Different notions of "holds" are compatible with these characterisations, e.g. truth in the minimal model of $P \cup \Delta$, etc.

Proof procedures:

Backward reasoning to show G holds in $P \cup \Delta$.
Forward reasoning to show IC holds in $P \cup \Delta$.

In the simplest case, showing G generates Δ and checking IC tests Δ.

IFF proof procedure for ALP (Fung and Kowalski 1997)

Definitions of non-abducible predicates in completion form:

atom iff disjunction of (conjunctions of (atomic formulae or conditionals)).

Negative literals *not p* are represented as conditionals *if p then false*.

Integrity constraints are represented as conditionals of the form:

if conditions then disjunction of conjunctions

Conclusions of integrity constraints have the same form as the bodies of definitions:

If X is a bank employee then X has social security number Y.

Disjuncts in the bodies of definitions have the same form as integrity constraints:

The banking department gets a 5 % bonus starting tomorrow
iff if X is an employee in the banking department today
and X has salary S today
then X has salary S + .05S tomorrow.

IFF proof procedure for ALP

Given an initial problem *G* and integrity constraints *IC*,
IFF conjoins *G* and *IC*.

The search space is represented as a *disjunction of conjunctions*.
Equivalence-preserving inferences transform one state of the search space into another, e.g.:

> Backward reasoning replaces an atom by its definition.
>
> Forward reasoning inside a disjunct adds the resolvent of an atom with a condition of a conditional to the same disjunct.

IFF is sound and, with certain modest restrictions, complete with respect to the Kunen three-valued semantics.

Completeness of IFF

ICs can be satisfied by making their conditions false:

if I commit a mortal sin
and don't go to confession
then I will go to hell

can be satisfied either by:

not committing a mortal sin,
committing a mortal sin and going to confession, or
committing a mortal sin, not going to confession and going to hell.

Completeness of IFF

Observation: *mortal sin*
IC: *If mortal sin and not confess then go to hell*
LP: *confess iff see priest*

A: *see priest* and *go to hell*

Forward reasoning derives *If not confess then go to hell*.

Rewriting gives *confess or go to hell*.

Backward reasoning derives *see priest or go to hell* .

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

ALP agents embed ALP in an observation-thought-decision-action cycle

- Record current observations.
- Use *forward and backward reasoning* to derive consequences of the observations, triggering any integrity constraints and adding any new goals.
- Use *backward reasoning* to reduce goals to sub-goals.
- Decide between sub-goals that are atomic actions.
- Execute the chosen actions.

Conflict Resolution

ICs: *If X attacks me then I attack X.*
If X attacks me then I run away.

Observation: *mary attacks me.*
Forward reasoning: *I attack mary and I run away.*

To avoid performing both actions:

ICs: *If X attacks me then I protect myself against X.*
LP: *I protect myself against X if I attack X.*
I protect myself against X if I run away.

Forward and backward reasoning:

I attack mary or I run away.

Semantics of ALP agents

Include time or state parameters in the semantics:

> *If X attacks me at time T1*
> *then I protect myself against X at time T2 and T1 ≤ T2.*

Include an action theory, such as the situation or event calculus, in the agent's beliefs *B*.

The semantics of the ALP agent cycle is a special case of the ALP semantics:

> Given beliefs *B*, goals *G*, initial database state S_0 and possibly infinite set $O = \{O_1, O_2, \ldots, O_n, \ldots\}$ of observations, an *ALP agent solution* is a possibly infinite set $\Delta = \{A_1, A_2, \ldots, A_m, \ldots\}$ of actions such that *G* is satisfied by the logic program $S_0 \cup B \cup O \cup \Delta$.
>
> E.g. *G* is true in the minimal (or perfect) model of $S_0 \cup B \cup O \cup \Delta$.

The ALP agent cycle is sound with respect to this semantics. Complete?

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

The Frame Problem

Action theories, such as the situation or event calculus, all include some form of frame axiom, e.g.:

> *fact F holds in state S+1 if fact F holds in state S*
> *and S+1 is obtained from S by action/event A*
> *and A does not terminate F.*

Forward reasoning copies every unterminated fact from state *S* to state *S+1*, duplicating them in both states *S* and *S+1*.

Backward reasoning requires a expensive calculation potentially going back to the initial state S_0.

The frame problem of efficiently reasoning about change of state has no solution within a purely logical representation.

Destructive change of state

Production systems and BDI agents
store only the current state
use destructive assignment to generate new states
do not employ an explicit representation of states or time.

ALP agents suffer from the frame problem.
However, they can avoid the frame problem in part by directly observing whether facts hold in the environment.

LPS (Logic-based Production System) (Kowalski and Sadri) is a variant of ALP agents without frame axioms,
but with a destructively changing database.

Outline of this talk

- Confusions
- Distinct functionalities
- Abductive logic programming (ALP)
- Evidence from
  - Psychology
  - Intelligent agents
  - Deductive databases
  - Abductive logic programming
- ALP agents
- The frame problem
- LPS, a logic based production system language with a destructively changing database

LPS – operational semantics

LPS maintains only the current state of a deductive database.
Facts (atoms) define the extensional predicates.
Logic programs (or deduction rules) define intensional predicates.

Actions change only the extensional predicates,
represented without explicit state parameters.

Actions are executed by deleting and adding facts.

Intentional predicates change implicitly as ramifications.

LPS - model-theoretic semantics

The sequence of database states with explicit state parameters can be viewed as a model-theoretic structure, which makes the production rules true:

Given beliefs B, goals G, an action theory, such as the event calculus EC and an initial database state S_0, a possibly infinite set $\Delta = \{A_1, A_2, \ldots, A_m, \ldots\}$ of actions is an *LPS solution* if and only if G is satisfied by the logic program $B \cup S_0 \cup \Delta \cup EC$.

E.g. G is true in the minimal (or perfect) model of $S_0 \cup B \cup \Delta \cup EC$.

LPS is sound, but is not complete for the same reason that condition-action rules are not complete.

Conclusions

Production systems and logic programs can be combined:
in an observation-thought-decision-action cycle,
along the lines of ALP agents,
retaining their individual contributions and
eliminating their overlap.

LPS is a variant of ALP agents with:
a declarative, logic-based semantics
an operational semantics with a destructively changing database.

Also: *How to be Artificially Intelligent- Computational Logic and Human Thinking,* to be published by C.U.P. 2010. Comments appreciated.

Two kinds of rules with distinct functionalities

In psychology
descriptive and deontic conditionals.

In intelligent agents
beliefs and goals.

In deductive databases
deduction rules and integrity constraints.

In abductive logic programming
logic programs and integrity constraints.

Logic programming has its own confusions

Are clauses to be understood declaratively or procedurally?

Backward reasoning is probably the main way in which logic programs are used in practice.

Forward reasoning is suitable for some applications, e.g. databases.

In abductive logic programming (ALP), clauses can be used to reason both backward and forward.

The ALP agent cycle

Conflict-resolution can be performed by using forward reasoning to derive consequences of candidate actions.

Decision theory can be used to choose actions whose consequences have maximal expected utility.

Backward reasoning can also be used to explain observations, before using forward reasoning to derive consequences.

Other approaches

In the majority of other approaches PRs, ECA rules or active integrity constraints are mapped into LP to provide them with LP-based semantics.
To our knowledge, there has been no proposal that would accommodate both LP and PR (or ECA rules or active integrity constraints) side-by-side.
Raschid combines LPs and ICs, but focuses on only two functionalities of PRs, namely on their use as reactive rules and as forward logic rules. She represents rules that add facts as LPs, and rules that delete facts as ICs. She then transforms their combination into LP, and uses the fixed point semantics of LP to chain forward and thereby simulate the production system cycle.

Other approaches

Caroprese *et al.* transform active integrity constraints into LPs.
Fraternali and Tanca also consider active databases but provide a logic-based *core* syntax for representing low-level, procedural features of active database rules.

Zaniolo uses a situation calculus-like representation with frame axioms, and reduces PRs and ECA rules to LPs.
Statelog also uses a situation-calculus-like representation for the succession of database states. Like Zaniolo, Statelog represents PRs and ECAs as LPs, and gives them LP-based semantics. Neither is concerned with the role of ICs or with the use of LPs and PRs for goal-reduction.
Fernandes *et al.* also view ECAs in terms of change of state, but use the event calculus as the basis for an ECA language coupled with a deductive database. The event calculus is used to evaluate the condition part of the ECA rules and to provide a specification for the effects of executing the action part.
Alferes *et al.* extend the dynamic logic programming system EVOLP by adding complex events and actions as well as external actions. ERA combines ECA and LP rules, and the firing of the ECA rules can generate actions that add or delete ECA or LP rules, as well as external actions. In in the declarative semantics the ECA rules are translated to LP.

ALP as a first step in combining logic programs and production rules

Integrity constraint: *If there is an emergency then I get help.*

Logic program: *A person gets help if the person presses the alarm signal button.*
There is an emergency if there is a fire.
There is an emergency if one person attacks another.
etc.

Abducible predicates: *there is a fire, a person presses the alarm signal button*

Observation: *There is a fire.*
Forward reasoning: *There is an emergency.*
Forward reasoning, Goal: *I get help*
Backward reasoning, Solution: *I press the alarm signal button.*

Alternatively, integrity constraints (with emergent goals).

If there is a fire then I press the alarm signal button.
If one person attacks another then I press the alarm signal button.

ALP agents embed ALP in an observation-thought-decision-action cycle

Beliefs, represented by logic programs,

Goals represented by integrity constraints, include

- condition-action rules
- obligations & prohibitions
- atomic and non-atomic actions

Thinking performed by IFF augmented by SK.

Deciding instead of conflict resolution.