Web Reasoning and Rule Systems

Third International Conference, RR 2009
Chantilly, VA, USA, October 25-26, 2009
Proceedings
Preface

The promise of the Semantic Web, at its most expansive, is to allow knowledge to be freely accessed and exchanged by software. It is now recognized that if the Semantic Web is to contain deep knowledge, the need for new representation and reasoning techniques is going to be critical. These techniques need to find the right trade-off between expressiveness, scalability and robustness to deal with the inherently incomplete, contradictory and uncertain nature of knowledge on the Web. The International Conference on Web Reasoning and Rule Systems (RR) was founded to address these needs and has grown into a major international forum in this area. The third RR conference was held during October 25–26, 2009 in Chantilly, Virginia, co-located with the International Semantic Web Conference (ISWC 2009).

This year 41 papers were submitted from authors in 21 countries. The Program Committee performed outstandingly to ensure that each paper submitted to RR 2009 was thoroughly reviewed by at least three referees in a short period of time. The resulting conference presented papers of high quality on many of the key issues for reasoning on the Semantic Web. RR 2009 was fortunate to have two distinguished invited speakers. Robert Kowalski, in his talk “Integrating Logic Programming and Production Systems with Abductive Logic Programming Agents” addressed some of the fundamental considerations behind reasoning about evolving systems. Benjamin Grossof’s talk “SILK: Higher Level Rules with Defaults and Semantic Scalability” described the design of a major next-generation rule system. The invited tutorial “Uncertainty Reasoning for the Semantic Web” by Thomas Lukasiewicz provided perspectives on a central issue in this area.

Regular papers addressed fundamental issues of reasoning with topics including deduction procedures for ontologies with defaults and for conceptual logic programs, evaluation procedures for path query languages, analysis of production systems using fixed-point logic, and general perspectives on control in rule engines. The importance of scalability was reflected by papers on distributed resolution for ontologies, parallel logic programming techniques for Abox querying, and the separation of terminological from assertional data. The topic of knowledge amalgamation was studied by papers on alignment, modularity and paraconsistency for ontologies. Uncertainty was explored by papers on semantics and inference procedures for fuzzy reasoning, and a paper on inference procedures for a logic of belief.

The results of Web Reasoning and Rule Systems are not confined only to foundational issues, but are being applied to Web standards and real-world applications. Papers considered paraconsistent and fuzzy extensions to the RDF standard, which just celebrated its 10th anniversary and is becoming widely used. Another paper explored scalability of the OWL-2 standard, which is soon to be
published by W3C. Finally, one paper described how Semantic Web techniques were applied in a risk-assessment system for elective surgery.

We thank the following sponsors for supporting RR 2009:

- BBN Technologies, USA
- Logic Programming Associates Ltd., UK
- Kno.e.sis Center, USA
- IOS Press
- The LarKC - Large Knowledge Collider Project
- Clark&Parsia LLC, USA
- Digital Enterprise Research Institute (DERI) at the National University of Ireland, Galway
- Vulcan Inc., USA
- The NeOn project

We are particularly thankful also to the authors, the invited speakers, and attendees for contributing and discussing the latest results in relevant areas to this conference, as well as to all members of the Program Committee, and the external reviewers for their critical reviews of submissions.

October 2009

Terrance Swift
Axel Polleres
Organization

General Chair

Michael Kifer
SUNY Stony Brook, NY, USA

Program Co-chairs

Terrance Swift
CENTRIA, Universidade Nova de Lisboa, Portugal

Axel Polleres
Digital Enterprise Research Institute National University of Ireland, Galway

Local Arrangements Chair

Guilin Qi
Universität Karlsruhe, Germany

Sponsorship Chair

Pascal Hitzler
Universität Karlsruhe, Germany

Program Committee

Grigoris Antoniou
Marcelo Arenas
Leopoldo Bertossi
Piero Bonatti
Carlos Damásio
Wlodek Drabent
Bernardo Cuenca Grau
Volker Haarslev
Giovambattista Ianni
Manolis Koubarakis
Domenico Lembo
Thomas Lukasiewicz
Francesca Alessandra Lisi

Wolfgang May
David Pearce
Enrico Pontelli
Guilin Qi
Marie-Christine Rousset
Sebastian Rudolph
Sebastian Schaffert
Michael Sintek
Heiner Stuckenschmidt
Péter Szeredi
Sergio Tessaris
Hans Tompits
Dirk Vermeir

Additional Reviewers

Anne Schlicht
Johannes Oetsch
Luigi Sauro

Francesco Ricca
Gergely Lukácsy
Zsolt Zombori
VIII Organization

Arash Shaban-Nejad
Jörg Pührer
Héctor Pérez-Urbina
Christian Meilicke
Jun Fang
Antoine Zimmermann

Ratnesh Sahay
Jianfeng Du
Jeroen Janssen
Irini Fundulaki
Manuel Möller
# Table of Contents

## Invited Papers and Tutorial

Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents ........................................ 1  
*Robert Kowalski and Fariba Sadri*

SILK: Higher Level Rules with Defaults and Semantic Scalability  
(Abstract of Invited Talk) .............................................................. 24  
*Benjamin N. Grosof*

Uncertainty Reasoning for the Semantic Web  .................... 26  
*Thomas Lukasiewicz*

## Proof/Deduction Procedures

A Preferential Tableaux Calculus for Circumscriptive $\textit{ALCO}$ .... 40  
*Stephan Grimm and Pascal Hitzler*

A Reasoner for Simple Conceptual Logic Programs ................ 55  
*Stijn Heymans, Cristina Feier, and Thomas Eiter*

Search for More Declarativity: Backward Reasoning for Rule Languages Reconsidered ......................................................... 71  
*Simon Brodt, François Bry, and Norbert Eisinger*

## Scalability

Distributed Resolution for Expressive Ontology Networks ............ 87  
*Anne Schlicht and Heiner Stuckenschmidt*

Scalable Web Reasoning Using Logic Programming Techniques .... 102  
*Gergely Lukácsy and Péter Szeredi*

On the Ostensibly Silent ‘W’ in OWL 2 RL ...................... 118  
*Aidan Hogan and Stefan Decker*

## Uncertainty

Answer Sets in a Fuzzy Equilibrium Logic ......................... 135  
*Steven Schockaert, Jeroen Janssen, Dirk Vermeir, and Martine De Cock*
Belief Logic Programming with Cyclic Dependencies .............................. 150

Hui Wan

A Minimal Deductive System for General Fuzzy RDF ...................... 166

Umberto Straccia

Knowledge Amalgamation and Querying

An Efficient Method for Computing Alignment Diagnoses ................. 182

Christian Meilicke and Heiner Stuckenschmidt

Paraconsistent Reasoning for OWL 2 ........................................... 197

Yue Ma and Pascal Hitzler

A Formal Theory for Modular ERDF Ontologies ............................. 212

Anastasia Analyti, Grigoris Antoniou, and Carlos Viegas Damásio

The Perfect Match: RPL and RDF Rule Languages ....................... 227

François Bry, Tim Furche, and Benedikt Linse

Rules for Decision Support and Production Systems

A Hybrid Architecture for a Preoperative Decision Support System
Using a Rule Engine and a Reasoner on a Clinical Ontology ............ 242

Matt-Mouley Bouamrane, Alan Rector, and Martin Hurrell

A Logic Based Approach to the Static Analysis of Production
Systems ....................................................................................... 254

Jos de Bruijn and Martín Rezk

Author Index ............................................................................... 269
Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents

Robert Kowalski and Fariba Sadri

Department of Computing, Imperial College London, 180 Queens Gate, London SW7 2AZ
{rak,fs}@doc.ic.ac.uk

Abstract. In this paper we argue the case for integrating the distinctive functionalities of logic programs and production systems within an abductive logic programming agent framework. In this framework, logic programs function as an agent’s beliefs and production rules function as the agent’s goals. The semantics and proof procedures are based on abductive logic programming, in which logic programs are integrated with integrity constraints that behave like production rules.

Similarly to production systems, the proof procedure is an operational semantics, which manipulates the current state of a database, which is modified by actions implemented by destructive assignment. The semantics can be viewed as generating a model, based on the sequence of database states and logic program, which makes the production rules true.

Keywords: Abductive logic programming, Production systems, Integrity constraints, agents.

1 Introduction

Rules are the basic form of knowledge representation in many areas of Artificial Intelligence, including both production systems and logic programming, and more recently in BDI (Belief Desire Intentions) agent languages. Despite their wide-spread use, there is a great deal of confusion between the different kinds of rules, and little agreement about the relationship between them.

In this paper we argue that production rules and logic programming rules have complementary characters and that one cannot usefully be reduced to the other. We show how abductive logic programming (ALP) combines the two kinds of rules in a single unified framework. The ALP framework gives a model-theoretic semantics to both kinds of rules and provides them with powerful proof procedures, combining both backward and forward reasoning. We present evidence from BDI agents, deductive databases and psychological experiments to support the distinct nature of the two kinds of rules.

We discuss the impact of including the two kinds of rules in a production system or agent cycle, which embeds the rules in a destructively changing environment, which is like a production system working memory. In this embedding, the environment can be viewed as the semantic structure that gives meaning to the two kinds of rules. Although the rules themselves need to be understood as explicitly or implicitly representing...
change of state, the fact that the environment changes destructively and exists at any
given time only in its current state gives an efficient solution to the frame problem.
We assume the reader is familiar with the basic concepts of logic programming
and production systems, although not necessarily with all of their technicalities.

1.1 Confusions

The most popular textbook on Artificial Intelligence Russell and Norvig [45] views
production rules as just logical conditionals used to reason forward (page 286). In
contrast, in one of the main textbooks on Cognitive Science, Thagard [50] argues that
“Rules are if-then structures …very similar to the conditionals…, but they have dif-
erent representational and computational properties.” (page 43). “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 bus fare, THEN you can catch a bus.” (page 45).

Thagard [50] characterizes Prolog as “a programming language that uses logic rep-
resentations and deductive techniques”. Simon [48], on the other hand, includes
Prolog “among the production systems widely used in cognitive simulation.”

There is a similar confusion in the field of agents. Rao [41], for example, charac-
terises AgentSpeak as similar to logic programming. But in his comparison, he con-
siders only the similarities between the operational semantics of plans in AgentSpeak
and the execution of clauses in logic programming. He ignores the declarative seman-
tics of logic programs (LPs).

1.2 Production Systems and Logic Programs in Practice

There have been many theoretical studies of the relationship between production rules
and logic programs, which we discuss below in Section 2. Most of this work has been
focussed on giving a declarative semantics to production systems by translating them
into logic programs. However, there seems to have been little attention paid to the
way in which logic programs and production rules are used in practice, and conse-
quently little attempt to use this practice to guide the theoretical analysis. We argue
that in practice, the two kinds of rules have both distinct and overlapping functional-
ities, and that the distinct functionalities are lost by translating one kind of rule into
the other. We will show that abductive logic programming (ALP) both capitalises on
the distinct functionalities and eliminates the overlap.

We argue that, in addition to the production rule (PR) cycle and destructively
changing database, which are absent in LP, PRs offer three distinct functionalities:
reactive rules that implement stimulus-response associations; forward chaining logic
rules; and goal-reduction rules.

Reactive rules are, arguably, the most distinctive type of production rules, which
are responsible for their general characterisation as condition-action rules. This kind
of rule typically has implicit or emergent goals. For example, the rule if a car coming
towards you then get out of its way has the implicit goal to stay safe. Reactive rules
provide a functionality that is not directly available in logic programming.

The second kind of rule, for example if X is a cat then X is an animal uses forward
chaining to implement forward reasoning with a logical conditional. It is probably this
kind of rule that gives the impression that production rules are just conditionals used to reason forward.

It is the third kind of rule, exemplified by Thagard’s example of the goal-reduction rule “IF you want to go home for the weekend and you have bus fare, THEN you can catch a bus.”, that overlaps the most with logic programming. In logic programming, such strategic rules would be obtained by reasoning backward with the clause you go home for the weekend if you have bus fare and you catch a bus. The two best known cognitive models of human thinking, SOAR [34] and ACT-R [3], are based on production systems and focus on the use of production rules for goal-reduction.

Logic programming has its own confusions, mostly about whether clauses are to be understood declaratively or procedurally. The purely declarative interpretation of clauses, which is neutral about reasoning method, is probably the one that is most attractive to its admirers. It is well-suited for high-level program specifications and for certain applications where efficiency is not a major concern.

However, it is probably the procedural interpretation, in which clauses are used to obtain goal-reduction by backward reasoning, that is the main way in which logic programs are used in practice. This is also where there is the greatest overlap with production rules. Arguably, logic programs with backward reasoning are more suitable for this purpose than production rules with forward reasoning, because logic programs can also be interpreted declaratively. The declarative interpretation of logic programs makes it possible to give goal-reduction procedures the declarative semantics that is missing with production rules.

The confusion between the declarative and procedural uses of logic programs and how best to combine them is well-known even though it is not very well solved. However, there is another use of logic programs that has received less attention, and is perhaps even more confusing. It is the use of logic programs for forward reasoning. This use is not very common in practice, but is prevalent in theoretical investigations of logic programming. We will see that in ALP, clauses can be used to reason both backward and forward.

1.3 Combining Production Systems and Logic Programs

Broadly speaking, there are four motivations for combining PRs and LPs:

1. To eliminate the overlap between forward logic rules in PRs and forward/declarative clauses in LP. For example, one very simple combination of PRs and LPs is to use LPs to define ramifications of the working memory/database. Then existing PRs could simply query a deductive database rather than a relational database. This would hand over the forward reasoning logic rules from the PR to the LP component. Moreover, it would allow the decision about how to execute the ramifications to be taken by the implementation. The declarative semantics of ramifications would be compatible with executing them forward, backward, or any combination of the two.

2. To eliminate the overlap for goal-reduction, by using LPs for this purpose. Using LP for goal-reduction provides system support for managing goals as and-or trees, which is missing in production systems. Whereas most production systems just treat goals as ordinary facts in the working memory, SOAR and ACT-R manipulate them in goal stacks. Using LP for goal-reduction allows the declarative nature of LP
clauses to be exploited, so they can be used either for goal-reduction or for forward reasoning, as the context requires.

3. To provide a declarative semantics for PRs and for the combination of LP and PRs. Declarative semantics provides an independent specification for implementations, as illustrated by the discussion above about the implementation of ramifications. Declarative semantics also clarifies the nature of PRs as a representation language. Without a declarative semantics, which establishes a relationship between syntactic expressions and semantic structures, of the kind provided by the model theory of logic, the term “representation” has no meaning.

Our proposal is to combine LP and PR in the same way that ALP combines LP and integrity constraints (ICs), and to use the model-theoretic semantics of ALP to give a model-theoretic semantics to the combination of LP and PR. We will show that integrity constraints in ALP, especially when embedded in ALP agents, generalise production rules to include condition-goal rules, where the goal is like the body of a plan in BDI agents.

4. To provide a cycle and destructive database of facts, missing in LP. Without a cycle, LP is both closed and passive – closed because logic programs cannot be updated by the environment, and passive because they cannot perform updates on the environment. Without a destructive database of facts, LP suffers from the inefficiencies of the frame problem.

2 Other Approaches

Typically PRs, as well as event-condition-action (ECA) rules and active integrity constraints are defined by means of an operational semantics based on state transitions. However several authors have studied the relationship between these various kinds of rules and LP, with the aim of providing the rules with a declarative LP-based semantics. In the majority of these 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 with an integrated semantics or proof procedure that would exploit the strengths of both paradigms.

Raschid [52] 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.


Caroprese et al. [5] transform active integrity constraints into LPs. They characterise the set of “founded” repairs for the database as the stable model of the database augmented by the LP representation of the active integrity constraints. Fraternali and Tanca [17] also consider active databases but provide a logic-based core syntax for representing low-level, procedural features of active database rules. They provide procedural semantics for core rules and show how this can capture the procedural semantics of known active database systems.
Most other work regards the cycle and actions in condition-action rules and ECA rules declaratively as performing a change of state. Zaniolo [54], for example, uses a situation calculus-like representation with frame axioms, and reduces PRs and ECA rules to LPs. Statelog [53] 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. [16] 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. The ECA language also allows the recognition of complex events from an event history.

ERA (Evolving Reactive Algebraic Programs), developed by Alferes et al. [1], extends the dynamic logic programming system EVOLP [2] 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 the operational semantics the ECA and LP rules maintain their distinct characteristics, but in the declarative semantics the ECA rules are translated to LP. The declarative semantics is based on a variant of stable models developed for EVOLP.

3 The Selection Task

Psychological evidence from the selection task suggests that people reason differently with two kinds of conditionals. One school of thought is that the difference depends, at least in part, on whether conditionals are interpreted descriptively or deontically. We will argue that descriptive conditionals are like logic programs, and deontic conditionals are like integrity constraints in abductive logic programming.

In Wason’s original selection task, there are four cards, with letters on one side and numbers on the other. The cards are lying on a table, and only one side of each card is visible, showing the letters D and F, and the numbers 3 and 7.

The task is to 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.

Variations of this experiment have been performed numerous times. The surprising result is that only about 10% of the subjects give the correct answer according to the norms of classical logic.

Almost everyone recognizes, correctly, that the card showing D needs to be turned over, to make sure there is a 3 on the other side. Most people also recognize, correctly, that the card showing F does not need to be turned over. But many subjects also think, incorrectly, that it is necessary to turn over the card showing 3, to make sure there is a D on the other side. This is logically incorrect, because the implication does not claim that conversely:
If there is a 3 on one side, 
then there is a D on the other side.

Only a few subjects realise that it is necessary to turn over the card showing 7, to make sure that D is not on the other side. It is necessary to turn over the 7, because the original implication is logically equivalent to its contrapositive:

If the number on one side is not 3 (e.g. 7),
then the letter on the other side is not D.

It has been shown that people perform far better, according to the norms of classical logic, when the selection task experiment is conducted with certain other formulations of the problem that are formally equivalent to the card version of the task. The classic experiment of this kind considers the situation in which people are drinking in a bar, and the subject is asked to check whether the following conditional holds:

If a person is drinking alcohol in a bar, 
then the person is at least eighteen years old.

Again the subject is presented with four cases to consider, but this time instead of four cards there are four people. We can see what two people are drinking, but cannot see how old they are; and we can see how old two people are, but not what they are drinking. In contrast with the card version of the selection task, most people solve the bar version correctly. They realise that it is necessary to check the person drinking alcohol to make sure that he is at least eighteen years old, and to check the person under eighteen to make sure that she is not drinking alcohol. They also realise that it is not necessary to check the person who is eighteen years old or older, nor the person who is drinking a non-alcoholic beverage.

Cognitive psychologists have proposed a bewildering number of theories to explain why people are so much better at solving such versions of the selection task compared with the original card version. One of the most influential of these is the theory put forward by Cheng and Holyoak [8] that people tend to reason in accordance with classical logic when conditionals involve deontic notions concerned with permission, obligation and prohibition. However, except for [49] and [25], there has been little attempt to explain why people reason as they do with descriptive variants of the selection task, such as the card version.

Stenning and van Lambalgen [49] propose that understanding and solving the selection task is a two stage process: interpreting the conditional and then reasoning with the interpretation. They argue that people interpret conditionals of the kind involved in the card version of the task as logic programs. Interpreted as a logic programming clause, a conditional is understood, according to the completion semantics, as the if-half of a definition in if-and-only-if form [9]. The completion semantics entails the converse of conditionals in the selection task and inhibits the application of reasoning with contrapositives. This is exactly the kind of reasoning most people display in the card version of the selection task.

Stenning and van Lambalgen also argue that it is natural to interpret conditionals of the kind involved in the bar version of the task in deontic logic. However, Kowalski [25] has argued that the deontic interpretation can be obtained more simply by interpreting conditionals as integrity constraints.
It is curious that, although production systems, such as SOAR and ACT-R, are widely used in cognitive psychology as a model of human thinking, it seems that conditionals in the form of condition-action rules have not been studied in relation to the selection task.

4 Intelligent Agents

The psychological evidence that people reason differently with descriptive and deontic conditionals is mirrored by the notion that the mental state of an intelligent agent is best understood as having separate goals and beliefs. An agent’s beliefs represent the way things are, and its goals represent the way the agent would like them to be. Thus beliefs have a descriptive character, whereas goals have a prescriptive or deontic character. The BDI (Belief, Desire, Intention) [4] model of agents adds to beliefs and desires (or goals) the notion of intention, which is an agent’s plan of actions for achieving its goals. Intentions are derived from goals using beliefs to reduce goals to subgoals.

Arguably, the most influential of the BDI agent models is that of Rao and Georgeff [42] and its successors dMARS [13] and AgentSpeak [41]. The abstract agent intermediate language AIL of Dennis et al. [12] is an abstraction of these languages, based mainly on AgentSpeak and its successors.

The earliest BDI agent systems were specified in multi-modal logics, with separate modal operators for goals, beliefs and intentions. However, their procedural implementations bore little resemblance to their logical specifications. AgentSpeak abandoned the attempt to relate the modal logic specifications with their procedural implementations, observing instead that “…one can view agent programs as multi-threaded interruptible logic programming clauses”. This abandonment of modal logic specifications is inherited by AgentSpeak’s successors and their abstraction AIL.

However, this view of AgentSpeak in logic programming terms applies only to the procedural interpretation of clauses. In fact, programs in AgentSpeak are better viewed as a generalisation of production rules than as variant of logic programming. AgentSpeak programs, also called plans, have the form:

\[ \text{Event } E: \text{ conditions } C \text{ goals } G \text{ and actions } A. \]

AgentSpeak plans manipulate a “declarative” database, like the working memory in production systems. The database contains both belief literals (atoms and negations of atoms) and goal atoms. The belief literals represent the current state of the environment and are added and deleted destructively, simulating the execution of atomic actions. Goal atoms are added when they are generated as sub-goals, and deleted when they are solved.

The event \( E \) in the head of a plan can be the addition or deletion of a belief or of a goal. Plans are embedded in a cycle similar to the production system cycle, and are executed in the direction in which they are written. With the arrow written backwards, the execution of plans can be viewed as backward chaining. However, if the arrow is reversed, their execution can be viewed as forward chaining. No matter how their execution is viewed, plans have only an operational semantics.
The following are examples of possible AgentSpeak plans:

+ there is a fire: true    + there is an emergency.
+? there is an emergency: true   ? there is a fire.
+! there is an emergency: true   ! there is a fire.

Observations and actions do not have associated times, and the database provides only a snapshot of the current state of the world. To compensate for this lack of a temporal representation, the prefixes +,-,!, and ? are used to stand for add, delete, achieve, and test, respectively.

Notice that the first plan behaves like a logical conditional used to reason forwards, but in the opposite direction of the arrow, to conclude there is an emergency if it is observed that there is a fire. The other two plans are goal-reduction rules, one for testing whether there is an emergency and the other for creating an emergency.

In general, in the case where a plan has the form:

\[ \text{Goal } E: \text{ conditions } C \text{ goals } G \text{ and actions } A \]

and the triggering event \( E \) is the addition of a goal, the plan can be reformulated as a logic programming clause:

\[ E' \text{ if } C' \text{ and } G' \text{ and } A' \text{ and temporal constraints} \]

where the prefixed predicates of AgentSpeak are replaced by predicates with explicit associated times. The corresponding clause subsumes the behaviour of the plan, but also has a declarative reading.

In the simple example of the three plans above, the corresponding clause is:

\[ \text{there is an emergency at time } T \text{ if there is a fire at time } T. \]

Represented in this way, the clause can be viewed as defining a ramification, which views fires more abstractly as emergencies.

Thus, although BDI agent models were inspired by the modal logic representation of goals and beliefs, this inspiration has largely been lost in recent years. Most agent systems today represent goals as facts, mixed with belief facts in database or represented in a separate stack, as in ACT-R and SOAR. Belief facts and goal facts are manipulated uniformly by procedures, often called plans, which generalise production rules.

The only kind of goal that can easily be represented as a fact in a database or in a goal stack, in this way, is an achievement goal, which is a one-off problem to be solved, including the problem of achieving some desired future state of the world. The higher-level notion of maintenance goal, which persists over all states of the world, is lost in the process.

In ALP agents, as we will see below, maintenance goals are integrity constraints, which have the form of universally quantified conditionals with existentially quantified conclusions. Thus maintenance goals are higher-level than achievement goals in ALP, because an achievement goal is derived as an instance of the conclusion of a maintenance goal, whenever an instance of the conditions of the maintenance goal are satisfied. For example, given the maintenance goal:
For all times $T_1$

If there is an emergency at time $T_1$ then there exists a time $T_2$ such that
I get help at time $T_2$ and $T_1 < T_2$

and an emergency at some specific time $t_1$, forward reasoning in ALP would derive
the achievement goal of getting help at some later time. The later time could be
bounded by an additional conjunct in the conclusion, or it could be left to the decision-making component of the agent cycle to take into account how urgently the
achievement goal needs to be accomplished.

In AgentSpeak and its successors, maintenance goals and goal-reduction rules are
just different kinds of plans. Our aim is to restore the high level distinction between
goals and beliefs, to recognise the importance of maintenance goals in particular, to
combine the distinctive forms of reasoning appropriate to the distinction between
goals and beliefs, and to give their combination a logical, model-theoretic semantics. For this purpose we interpret beliefs as logic programs, goals as integrity constraints, and combine goals and beliefs in the way that logic programs and integrity constraints are combined in abductive logic programming.

5 Deductive Databases

The combination of logic programs and integrity constraints in ALP evolved from
their relationship in deductive databases. The semantics of integrity constraints and
the development of proof procedures for constraint satisfaction were active research
areas in deductive databases in the 1980s.

The distinction between a database and its integrity constraints is intuitively clear
in database systems, where integrity constraints have the same semantics as database
queries. But, whereas ad hoc queries are concerned with properties that hold in a
given state of the database, integrity constraints are persistent queries that are in-
tended to hold in all states of the database. Ad hoc queries can be viewed as achieve-
ment goals, and integrity constraints can be viewed as maintenance goals and include
prohibitions as a special case. The database itself can be thought of as a set of beliefs.
Thus conventional database systems can be viewed as passive agents, which are open
to updates from the environment, but are unable to perform actions themselves.

In relational databases, there is a clear distinction between the syntax of beliefs in
the database and the syntax of goals. Beliefs are simple, ground, atomic sentences.
Goals, both ad hoc queries and persistent integrity constraints, are sentences of first-
order logic. However, the syntactic distinction is less clear in deductive databases,
where the database consists of both ground facts and more general logic programs
(also called deduction rules). The distinction is complicated by the fact that it is often
natural to express both deduction rules and integrity constraints in a similar condi-
tional form. Informal criteria for distinguishing between deduction rules and integrity
constraints were proposed by Nicolas and Gallaire [37]. Consider, for example, the
two conditionals:

The bus leaves at time $X:00$ if $X$ is an integer and $9 \leq X \leq 18$.

If the bus leaves at time $X:00$, then for some integer $Y$,
the bus arrives at its destination at time $X:Y$ and $20 \leq Y \leq 30$
The first conditional defines bus departure times constructively and therefore can function as a general rule in a deductive database. However, the second conditional has an existential quantifier in the conclusion, which means that it cannot be used to define data, but can only be used to constrain data, as an integrity constraint. In a passive database, the integrity constraint can be used to check updates to the database. But in an agent, the integrity constraint can be used as a maintenance goal, to generate achievement goals, which can then be reduced to plans of action. Thus an agent can be thought of as an active database, and its maintenance goals can be regarded as active database rules.

Several competing views of the semantics of integrity constraints were intensively investigated in the 1980s. The two main views, to begin with, were the consistency view and the theorem-hood view, both of which were defined relative to the completion of the database. In the consistency view, an integrity constraint is satisfied if it is consistent with the completion of the database. In the theorem-hood view, it is satisfied if it is a theorem, logically entailed by the completion.

Reiter [44] proposed an epistemic view of integrity constraints, according to which integrity constraints are statements about what the database knows. For example, the integrity constraint:

\[
\text{If } X \text{ is an employee then for some integer } Y \\
X \text{ has social security number } Y
\]

would be interpreted as:

\[
\text{If the database knows that } X \text{ is an employee then for some integer } Y \\
\text{the database knows that } X \text{ has social security number } Y
\]

However, Reiter [44] also showed that all three views are equivalent in many cases for databases augmented with the closed world assumption [43] which is the set of all the negations of atomic sentences that are not entailed by the database. For relational databases, the three views are also equivalent to the standard view in relational databases that a database satisfies an integrity constraint if it is true in the database regarded as a Herbrand model.

More generally, the four views of integrity satisfaction (consistency, theorem-hood, epistemic and truth-theoretic) coincide for any database whose closure has a single model, In the case of Horn clause databases, the four views are equivalent to the view that an integrity constraint is satisfied if (and only if) it is true in the unique minimal model of the set of Horn clauses.

However, whether or not the different views of integrity satisfaction are equivalent for a given database, it is generally accepted that queries and integrity constraints have the same semantics. Therefore, the most obvious way to check integrity satisfaction is to treat each integrity constraint as a query, using the same procedure for integrity checking as for query evaluation. The problem with this approach, is that in a dynamic setting, where the current database state is largely identical to the previous state, much of the work involved in processing the constraints in a new state duplicates the work performed in the previous state.

To alleviate this problem, the vast majority of integrity checking procedures developed in the 1980s incrementally checked the integrity of a new state of the database, assuming that the integrity constraints already hold in the previous state. As a