A Declarative Agent Programming Language Based on Action Theories

Conrad Drescher, Stephan Schiffel, and Michael Thielscher

Department of Computer Science
Dresden University of Technology
{ conrad.drescher,stephan.schiffel,mit } @inf.tu-dresden.de

Abstract. We discuss a new concept of agent programs that combines logic programming with reasoning about actions. These *agent logic programs* are characterized by a clear separation between the specification of the agent's strategic behavior and the underlying theory about the agent's actions and their effects. This makes it a generic, declarative agent programming language, which can be combined with an action representation formalism of one's choice. We present a declarative semantics for agent logic programs along with (two versions of) a sound and complete operational semantics, which combines the standard inference mechanisms for (constraint) logic programs with reasoning about actions.

1 Introduction

Action theories, like the classical Situation Calculus [1], provide the foundations for the design of artificial, intelligent agents capable of reasoning about their own actions. Research in this area has led to mature action theories, including the modern Situation Calculus [2] or the Event Calculus [3], which allow to endow agents with knowledge of complex dynamic environments. Despite the existence of elaborate action theories today, surprisingly few attempts have been made to integrate these into actual programming languages for intelligent agents. Existing agent programming languages such as [4–6] use nothing but a very basic concept of reasoning about actions, where the belief base of an agent is updated in a STRIPS-like [7] fashion upon executing an action. Moreover, the agent programmer always has to provide this belief update as part of the behavioral strategy, rather than having the agent use a separate, behavior-independent action theory to reason about its actions and their effects.

Two exceptions to this notable gap between practical agent programming languages on the one hand and elaborate action theories on the other hand, are GOLOG [2] and FLUX [8], which have been built on top of two action theories, namely, the Situation Calculus and the Fluent Calculus. Both languages are procedural in nature: the semantics of FLUX is defined on the basis of Prolog computation trees, while GOLOG (= Algol in Logic) combines standard elements from imperative programming with reasoning about actions. Especially for Artificial Intelligence applications, however, declarative programming languages have

proved a useful alternative to the procedural programming paradigm (see, e.g., [9]). An open research issue, therefore, is the development of a declarative agent programming language based on action theories.

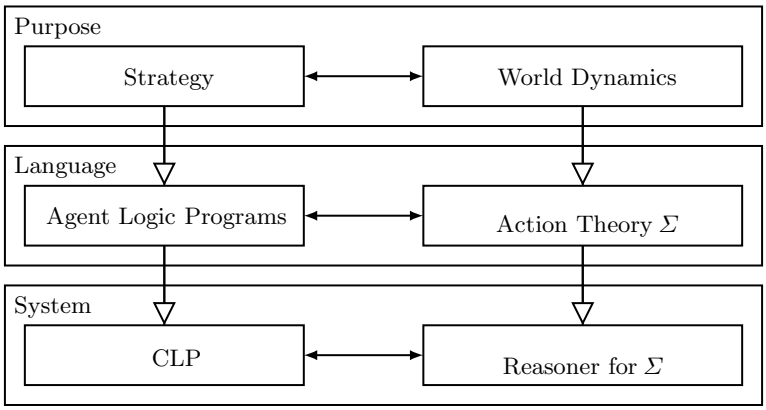

Fig. 1. Strategic Behavior in Dynamic Worlds with Agent Logic Programs

In this paper, we investigate *agent logic programs*, which are obtained by combining the logic programming paradigm with action theories. These programs are characterized by a clear separation between the specification of the agent's strategic behavior (given as a logic program) and the underlying theory about the agent's actions and their effects (see Figure 1). This makes it a fully generic agent programming language, which can be combined with a variety of action formalisms, including the Situation-, Event-, and Fluent Calculus. Specifically,

- we give a generic, declarative semantics for agent logic programs;
- we provide a sound operational semantics which combines the standard inference mechanisms for logic programs with reasoning about actions; and
- we provide completeness results for the operational semantics by investigating increasingly expressive classes of agent logic programs and by incorporating concepts derived from constraint logic programming.

Agent programs can be used in two distinct ways: If executed online they directly control the behavior of an intelligent agent. By executing them offline the agent can infer plans that achieve strategic goals. We will see that agent logic programs go beyond established agent programming languages by being capable of inferring conditional plans for open-world planning problems.

The rest of this paper is organized as follows. In the next section, we give the basic formal definitions of an agent logic program and illustrate these by means of an example that will be used throughout the paper. We then show how the usual declarative reading of a logic program can be combined with an action theory to provide a declarative semantics for agent logic programs. Thereafter we

present two operational semantics and prove their soundness and completeness. In the ensuing section we discuss the use of agent logic programs for planning. We conclude with a brief discussion of the results and future work.

2 Agent Logic Programs

The purpose of agent logic programs is to provide high-level control programs for agents using a combination of declarative programming with reasoning about actions. The syntax of these programs shall be kept very simple: standard (definite) logic programs are augmented with just two special predicates, one — written `do(`α`)` — to denote the execution of an action by the agent, and one — written `?(`φ`)` — to verify properties against (the agent's model of) the state of its environment. This model, and how it is affected by actions, is defined in a separate action theory. This allows for a clear separation between the agent's strategic behavior (given by the agent logic program itself) and the underlying theory about the agent's actions and their effects. Prior to giving the formal definition, let us illustrate the idea by an example agent logic program.

Example 1 Consider an agent whose task is to find gold in a maze. For the sake of simplicity, the states of the environment shall be described by a single *fluent* (i.e., state property): $At(u, x)$ to denote that $u \in \{Agent, Gold\}$ is at location x. The agent can perform the action $Go(y)$ of going to location y, which is possible if y is adjacent to, and accessible from, the current location of the agent. The fluent and action are used as basic elements in the following agent logic program. It describes a simple search strategy based on two parameters: a given list of locations (choice points) that the agent may visit, and an ordered collection of backtracking points.[1]

```
explore(Choicepoints,Backtrack) :-          % finished, if
   ?(at(agent,X)), ?(at(gold,X)).           % gold is found

explore(Choicepoints,Backtrack) :-
   ?(at(agent,X)),
   select(Y,Choicepoints,NewChoicepoints),  % choose available direction
   do(go(Y)),                               % go in this direction
   explore(NewChoicepoints,[X|Backtrack]).  % store it for backtracking

explore(Choicepoints,[X|Backtrack]) :-      % go back one step
   do(go(X)),
   explore(Choicepoints,Backtrack).

select(X,[X|Xs],Xs).
select(X,[Y|Xs],[Y|Ys]) :- select(X,Xs,Ys).
```

[1] Below, we follow the Prolog convention according to which variables are indicated by a leading uppercase letter.

Suppose we are given a list of choice points `C`, then the query `:- explore(C,[])` lets the agent systematically search for gold from its current location: the first clause describes the base case where the agent is successful; the second clause lets the agent select a new location from the list of choice points and go to this location (the declarative semantics and proof theory for `do(`α`)` will require that the action is possible at the time of execution); and the third clause sends the agent back using the latest backtracking point.

The example illustrates two distinct features of agent logic programs. First, an agent strategy is defined by a logic program that may use arbitrary function and predicate symbols in addition to the signature of the underlying action theory. Second, and in contrast to usual BDI-style programming languages like, e.g. AgentSpeak [6], the update of the agent's belief according to the effects of its actions is not part of the strategy. The formal definition of agent logic programs is as follows.

Definition 1 Consider an action theory signature Σ, including the pre-defined sorts ACTION and FLUENT, and a logic program signature Π.

- *Terms* are from $\Sigma \cup \Pi$.
- If `p` is an n-ary relation symbol from Π and $\mathtt{t}_1, ..., \mathtt{t}_n$ are terms, then `p(`$\mathtt{t}_1, ..., \mathtt{t}_n$`)` is a *program atom*.
- `do(`α`)` is a *program atom* if α is an ACTION term in Σ.
- `?(`φ`)` is a *program atom* if φ is a *state property* in Σ, that is, a formula (represented as a term) based on the FLUENTs in Σ.
- Clauses, programs, and queries are then defined as usual for definite logic programs, with the restriction that the two special atoms cannot occur in the head of a clause. ■

It is easy to verify that our example program complies with this definition given the aforementioned action theory signature and the usual list notation.

3 Semantics: Program + Action Theory

The semantics of an agent logic program is given in two steps. First, the program needs to be "temporalized," by making explicit the state change that is implicit in the use of the two special predicates, `do(`α`)` and `?(`φ`)`. Second, the resulting program is combined with an action theory as the basis for evaluating these two special predicates. The overall declarative semantics is the classical logical semantics of the expanded program together with the action theory.

Time is incorporated into a program through macro-expansion: two arguments of sort TIME[2] are added to every regular program atom $p(\bar{x})$, and then

[2] Which specific concept of time is being used depends on how the sort TIME is defined in the underlying action theory, which may be branching (as, e.g., in the Situation Calculus) or linear (as, e.g., in the Event Calculus).

$p(\bar{x}, s_1, s_2)$ is understood as restricting the truth of the atom to the temporal interval between (and including) s_1 and s_2. The two special atoms receive special treatment: atom `?(`φ`)` is re-written to $\mathit{Holds}(\varphi, s)$, with the intended meaning that φ is true at s; and `do(`α`)` is mapped onto $\mathit{Poss}(\alpha, s_1, s_2)$, meaning that action α can be executed at s_1 and that its execution ends in s_2. The formal definition is as follows.

Definition 2 For a clause `H :- B`$_1$`,...,B`$_n$ ($n \geq 0$), let $s_1, \ldots, s_{n+1}$ be variables of sort TIME.

- For $i = 1, \ldots, n$, if `B`$_i$ is of the form
  - `p(t`$_1$`,...,t`$_m$`)`, expand to $P(t_1, \ldots, t_m, s_i, s_{i+1})$.
  - `do(`α`)`, expand to $\mathit{Poss}(\alpha, s_i, s_{i+1})$.
  - `?(`φ`)`, expand to $\mathit{Holds}(\varphi, s_i)$ and let $s_{i+1} = s_i$.[3]
- The head atom `H=p(t`$_1$`,...,t`$_m$`)` is expanded to $P(t_1, \ldots, t_m, s_1, s_{n+1})$.
- The resulting clauses are understood as universally quantified implications.

Queries are expanded exactly like clause bodies, except that

- a special constant S_0 — denoting the earliest time-point in the underlying action theory — takes the place of s_1 ;
- the resulting conjunction is existentially quantified. ■

Example 1 (continued) The example program of the preceding section is understood as the following axioms:

$$
\begin{aligned}
(\forall)\mathit{Explore}(c, b, s, s) &\subset \mathit{Holds}(\mathit{At}(\mathit{Agent}, x), s) \wedge \mathit{Holds}(\mathit{At}(\mathit{Gold}, x), s)\\
(\forall)\mathit{Explore}(c, b, s_1, s_4) &\subset \mathit{Holds}(\mathit{At}(\mathit{Agent}, x), s_1) \wedge \mathit{Select}(y, c, c', s_1, s_2) \wedge\\
&\quad\ \mathit{Poss}(\mathit{Go}(y), s_2, s_3) \wedge \mathit{Explore}(c', [x|b], s_3, s_4)\\
(\forall)\mathit{Explore}(c, [x|b], s_1, s_3) &\subset \mathit{Poss}(\mathit{Go}(x), s_1, s_2) \wedge \mathit{Explore}(c, b, s_2, s_3)\\
(\forall)\mathit{Select}(x, [x|x'], x', s, s) &\subset \mathit{true}\\
(\forall)\mathit{Select}(x, [y|x'], [y|y'], s_1, s_2) &\subset \mathit{Select}(x, x', y', s_1, s_2)
\end{aligned}
$$

The resulting theory constitutes a purely logical axiomatization of the agent's strategy, which provides the basis for logical entailment. For instance, macro-expanding the query `:- explore(C,[])` from the above example results in the temporalized formula $(\exists s)\, \mathit{Explore}(C, [\,], S_0, s)$. If this formula follows from the axioms above, then that means that the strategy can be successfully executed, starting at S_0 , for the given list of choice points C . Whether this is actually the case of course depends on the additional action theory that is needed to evaluate the special atoms *Holds* and *Poss* in a macro-expanded program. ■

Macro-expansion provides the first part of the declarative semantics of an agent logic program; the second part is given by an action theory in form of a logical axiomatization of actions and their effects. The overall declarative semantics

[3] Setting $s_{i+1} = s_i$ is a convenient way of saying that the expansion of `?(`φ`)` "consumes" only one TIME variable, in which case less than $n+1$ different variables for a clause with n body atoms are needed.

of agent logic programs is given by the axiomatization consisting of the action theory and the expanded program.

In principle, agent logic programs can be combined with any action formalism that features the predicates $Poss(a, s_1, s_2)$ and $Holds(\phi, s)$. However, in the formal definitions we make some assumptions that existing action formalisms either already satisfy or can be made to satisfy: We stipulate that action theories are based on many-sorted first order logic with equality and the four sorts TIME, FLUENT, OBJECT, and ACTION.[4] Fluents are reified, and the standard predicates $Holds(f, s)$ and $Poss(a, s_1, s_2)$ are included. For actions, objects, and fluents unique name axioms have to be included. We abstract from a particular time structure: an axiomatization of the natural or the positive real numbers (providing the linear time structure of e.g. the Event Calculus), or of situations (the branching time structure of Situation and Fluent Calculus) can be included.

Definition 3 [Action Theory Formula Types] We stipulate that the following formula types are used by action theories:

- State formulas express what is true at particular times: A *state formula* $\Phi[\bar{s}]$ *in* $\bar{s}$ is a first-order formula with free variables $\bar{s}$ where
  - for each occurrence of $Holds(f, s)$ we have $s \in \bar{s}$;
  - predicate $Poss$ does not occur.
- A *state property* ϕ is an expression built from the standard logical connectives and terms $F(\bar{x})$ of sort FLUENT. With a slight abuse of notation, by $Holds(\phi, s)$ we denote the state formula obtained from state property ϕ by replacing every occurrence of a fluent f by $Holds(f, s)$.[5] State properties are used by agent logic programs in `?(Phi)` atoms.
- An *action precondition axiom* is of the form $(\forall)Poss(A(\bar{x}), s_1, s_2) \equiv \pi_A[s_1]$, where $\pi_A[s_1]$ is a state formula in s_1 with free variables among $s_1, s_2, \bar{x}$.
- *Effect axioms* have to be of the form $Poss(A(\bar{x}), s_1, s_2) \supset \phi[\bar{x}, s_1, s_2]$. This assumption is implicit in the macro-expansion of `do(A)` to $Poss(a, s_1, s_2)$.

For illustration, the following is a background axiomatization for our example scenario. Essentially it is a basic Fluent Calculus theory in the sense of [10], with a simple syntactic modification to meet the requirements just given.

Example 1 (continued)

- Initial state axiom

$$Holds(At(Agent, 1), S_0) \wedge Holds(At(Gold, 4), S_0)$$

- Precondition axiom

$$Poss(Go(y), s_1, s_2) \equiv (\exists x)(Holds(At(Agent, x), s_1) \wedge (y = x + 1 \vee y = x - 1)) \\ \wedge s_2 = Do(Go(y), s_1)$$

[4] By convention variable symbols s, f, x, and a are used for terms of sort TIME, FLUENT, OBJECT, and ACTION, respectively.

[5] In an expanded program Π we always treat $Holds(\phi, s)$ as atomic.

- Effect axiom

$$\begin{aligned}&\mathit{Poss}(\mathit{Go}(y), s_1, s_2) \supset \\ &\quad (\exists x)(\mathit{Holds}(\mathit{At}(\mathit{Agent}, x), s_1) \wedge \\ &\quad [(\forall f)\mathit{Holds}(f, s_2) \equiv (\mathit{Holds}(f, s_1) \vee f = \mathit{At}(\mathit{Agent}, y)) \wedge f \neq \mathit{At}(\mathit{Agent}, x)]).\end{aligned}$$

Given this (admittedly very simple, for the sake of illustration) specification of the background action theory, the axiomatization of the agent's strategy from above entails, for example, $(\exists s)\, \mathit{Explore}([2,3,4,5], [\,], S_0, s)$, because the background theory allows to conclude that

$$\mathit{Holds}(\mathit{At}(\mathit{Agent}, 4), S) \wedge \mathit{Holds}(\mathit{At}(\mathit{Gold}, 4), S),$$

where S denotes the situation term $\mathit{Do}(\mathit{Go}(4), \mathit{Do}(\mathit{Go}(3), \mathit{Do}(\mathit{Go}(2), S_0)))$. It follows that $\mathit{Explore}([5], [3,2,1], S, S)$ according to the first clause of our example ALP. Consider, now, the situation $S' = \mathit{Do}(\mathit{Go}(3), \mathit{Do}(\mathit{Go}(2), S_0))$, then action theory and strategy together imply

$$\mathit{Holds}(\mathit{At}(\mathit{Agent}, 3), S') \wedge \mathit{Select}(4, [4,5], [5], S', S') \wedge \mathit{Poss}(\mathit{Go}(4), S', S)$$

By using this in turn, along with $\mathit{Explore}([5], [3,2,1], S, S)$ from above, we obtain $\mathit{Explore}([4,5], [2,1], S', S)$, according to the second program clause. Continuing this line of reasoning, it can be shown that

$$\begin{aligned}&\mathit{Explore}([3,4,5], [1], \mathit{Do}(\mathit{Go}(2), S_0), S) \\ \text{and hence, } &\mathit{Explore}([2,3,4,5], [\,], S_0, S)\end{aligned}$$

This proves the claim that $(\exists s)\, \mathit{Explore}([2,3,4,5], [\,], S_0, s)$. On the other hand e.g. the query $(\exists s)\, \mathit{Explore}([2,4], [\,], S_0, s)$ is *not* entailed under the given background theory: Without location 3 among the choice points, the strategy does not allow the agent to reach the only location that is known to house gold. ■

The example illustrates how an agent logic program is interpreted logically by first adding an explicit notion of time and then combining the result with a suitable action theory. As indicated above, most action formalisms can be used directly or can be extended to serve as a background theory. Languages like the Planning Language PDDL [11] or the Game Description Language GDL [12] require the addition of a time structure before they can be employed.

4 Generic Proof Calculus

In this section, we provide an operational counterpart to the declarative semantics given in the preceding section, beginning with the simple integration of reasoning about actions with standard SLD-resolution.

4.1 Elementary Case: LP($\mathcal{D}$)

The basic proof calculus for agent logic programs is obtained as an adaptation of SLD-resolution: for action domain $\mathcal{D}$, expanded agent logic program Π and query $(\exists)\Gamma$, we prove that $\mathcal{D} \cup \Pi \models (\exists)\Gamma$ by proving unsatisfiability of $\mathcal{D} \cup \Pi \cup \{(\forall)\neg\Gamma\}$. As a fully generic proof calculus, for the two special atoms in queries and clause bodies the inference steps $\mathcal{D} \vDash \mathit{Holds}(\varphi, s)$ and $\mathcal{D} \vDash \mathit{Poss}(\alpha, s_1, s_2)$ are treated as atomic, which allows to integrate any reasoner for $\mathcal{D}$.

Definition 4 The proof calculus is given by two inference rules on *states*, which are of the form $\langle\ \neg\Gamma, \theta\ \rangle$:

- *Normal Goals* (with G_i different from *Holds* and *Poss*):

$$\frac{\langle\ (\neg G_1 \vee \ldots \vee \neg G_{i-1} \vee \neg G_i \vee \neg G_{i+1} \vee \ldots \vee \neg G_n),\ \theta_1\ \rangle}{\langle\ (\neg G_1 \vee \ldots \vee \neg G_{i-1} \vee \bigvee_{j=1..m} \neg B_j \vee \neg G_{i+1} \vee \ldots \vee \neg G_n)\theta_2,\ \theta_1\theta_2\ \rangle}$$

  where $H \subset B_1 \wedge \ldots \wedge B_m$ is a fresh variant of a clause in Π such that G_i and H unify with most general unifier θ_2.

- *Special Goals* ($G_i = \mathit{Holds}(\varphi, s)$ or $\mathit{Poss}(\alpha, s_1, s_2)$):

$$\frac{\langle\ (\neg G_1 \vee \ldots \vee \neg G_{i-1} \vee \neg G_i \vee \neg G_{i+1} \vee \ldots \vee \neg G_n),\ \theta_1\ \rangle}{\langle\ (\neg G_1 \vee \neg G_{i-1} \vee \neg G_{i+1} \vee \neg G_n)\theta_2,\ \theta_1\theta_2\ \rangle}$$

  such that $\mathcal{D} \vDash (\forall)G_i\theta_2$ with substitution θ_2 on the variables in G_i. ■

For illustration, the reader may verify that the agent logic program in Example 1, together with the background theory given in the preceding section, admits a derivation starting from $\langle\ \neg\mathit{Explore}([2,3,4,5],[\,],S_0,s),\varepsilon\ \rangle$ and ending with $\langle\ \Box, \theta\ \rangle$ [6] such that the replacement $s/\mathit{Do}(\mathit{Go}(4), \mathit{Do}(\mathit{Go}(3), \mathit{Do}(\mathit{Go}(2), S_0)))$ is part of the resulting substitution θ.

Under the assumption that the underlying reasoning about actions is sound, soundness of the basic proof calculus follows easily from the corresponding result in standard logic programming (see, e.g., [9]).

Proposition 1 (Soundness). *Let Π be an expanded agent logic program, $\mathcal{D}$ a background domain axiomatization, and $(\exists)\Gamma$ an expanded query. If there exists a derivation starting from $\langle\ (\forall)\neg\Gamma, \varepsilon\ \rangle$ and ending in $\langle\ \Box, \theta\ \rangle$, then $\Pi \cup \mathcal{D} \vDash (\forall)\Gamma\theta$.*

While being sound, the use of standard SLD-resolution in combination with reasoning about actions is incomplete in general. This can be illustrated with a simple example, which highlights the influence of the background action theory on the existence of computed answers to queries for an agent logic program.

Example 2 Suppose we are given the following disjunctive knowledge of the initial state:

$$\mathit{Holds}(\mathit{At}(\mathit{Gold}, 4), S_0) \vee \mathit{Holds}(\mathit{At}(\mathit{Gold}, 5), S_0) \tag{1}$$

[6] The symbols $\Box$ and ε stand for the empty query and the empty substitution.

Consider the query `:- ?(at(gold,X))`, corresponding to the question whether $(\exists x)\, \mathit{Holds}(\mathit{At}(\mathit{Gold}, x), S_0)$ is entailed. This is obviously the case given (1), but there is no *answer substitution* θ such that $\mathit{Holds}(\mathit{At}(\mathit{Gold}, x), S_0)\theta$ is implied. The same phenomenon can be observed in the presence of state knowledge that is merely "de dicto," as in $(\exists y)\, \mathit{Holds}(\mathit{At}(\mathit{Gold}, y), S_0)$ in place of (1).

In case of classical logic programming, the computational completeness of SLD-resolution hinges on the fact that whenever a program Π entails $(\exists)Q$ then there also exists a substitution θ such that $\Pi \models (\forall)Q\theta$. For this reason, we introduce the following restricted class of background action theories.

Definition 5 An action domain axiomatization $\mathcal{D}$ is *query-complete* if and only if $\mathcal{D} \models (\exists)Q$ implies that there exists a substitution θ such that $\mathcal{D} \models (\forall)Q\theta$, for any Q of the form $\mathit{Holds}(\varphi, s)$ or $\mathit{Poss}(\alpha, s_1, s_2)$.

Under the assumption that the underlying reasoning about actions is complete *and* that the background action theory is query-complete, the basic proof calculus for agent logic programs can be shown to be complete.

Definition 6 A *computation rule* is a function selecting an atom from a non-empty negated query to continue the derivation with.

Proposition 2 (Completeness). *Let Π be an expanded agent logic program, $\mathcal{D}$ a background domain axiomatization, and $(\exists)\Gamma$ an expanded query. If $\Pi \cup \mathcal{D} \models (\forall)\Gamma\theta_1$ for some θ_1, then there exists a successful derivation via any computation rule starting with $\langle\ \neg(\exists)\Gamma, \varepsilon\ \rangle$ and ending in $\langle\ \Box, \theta_2\ \rangle$. Furthermore, there is a substitution θ_3 such that $\Gamma\theta_1 = \Gamma\theta_2\theta_3$.*

Proof. The claim can be proved by a straightforward adaptation of Stärk's proof of the completeness of plain SLD-resolution [13]: his concept of an implication tree is extended by allowing instances of the two special atoms, *Holds* and *Poss*, to occur as leaves just in case they are entailed by the background theory. The base case in the completeness proofs then follows for these two atoms from the assumption of a query-complete theory.

It is worth pointing out that the restriction to query-completeness is not the same as the following, more common notion: A first-order theory is called *complete* iff for every sentence ϕ either ϕ or $\neg\phi$ is in the theory. Referring to Example 2, say, if the initial state axiom does not contain any information at all concerning the location of gold, then this theory *is* query-complete (while it is not complete in the above sense). Thus the completeness of the basic proof calculus does extend to action domains with incomplete information.

On the other hand, query-complete action domains cannot express disjunctive or mere existential information. Because this is exactly one of the strong-points of general action calculi, we next present a proof theory for agent logic programs that is suitable for the general case.

4.2 General Case: CLP($\mathcal{D}$)

We address the problem of incompleteness of the basic proof calculus by moving to the richer framework of constraint logic programming [14]. Denoted by CLP(X), constraint logic programming constitutes a family of languages where, in addition to the syntax of standard logic programs, special *constraints* are used and evaluated against the background constraint theory X. We instantiate this general framework to CLP($\mathcal{D}$)—constraint logic programming over action domains $\mathcal{D}$—where the two special atoms of agent logic programs, $Poss(\alpha, s_1, s_2)$ and $Holds(\varphi, s)$, are taken as constraints.

As illustrated by Example 2, it is the lack of a most general answer substitution that causes the incompleteness of the basic proof calculus in case of domains that are not query-complete. This motivates the use of the following, more expressive notion of answer substitutions (see, e.g., [15]).

Definition 7 A *disjunctive substitution* is a set $\Theta = \{\theta_1, \ldots, \theta_n\}$ of substitutions. The *application* of a disjunctive substitution Θ to a clause c results in the disjunction $\bigvee_{i=1,\ldots,n} c\theta_i$. The *composition* $\Theta_1\Theta_2$ of two disjunctive substitutions is defined as $\{\theta_i\theta_j \mid \theta_i \in \Theta_1 \text{ and } \theta_j \in \Theta_2\}$. A substitution $\Theta_1 = \{\theta_1, \ldots, \theta_k\}$ is *more general* than a substitution $\Theta_2 = \{\theta_{k+1}, \ldots, \theta_l\}$ if for every $\theta_i \in \Theta_1$ there exist $\theta_j \in \Theta_2$ and θ such that $\theta_i\theta = \theta_j$.

Every disjunctive substitution Θ determines a formula in disjunctive normal form consisting of equality atoms. With a little abuse of notation we will denote this formula by Θ, too; e.g. we treat $\Theta = \{\{x/3\}, \{x/4\}\}$ and $x = 3 \vee x = 4$ interchangeably. These equational formulas, together with *Holds* and *Poss* atoms, constitute the elements of the *constraint store*, which replaces the simple substitutions in the derivation states used in LP($\mathcal{D}$).

In CLP(X), the derivation rule that handles constraint atoms G is based on the logical equivalence $G \wedge \mathcal{S} \equiv \mathcal{S}'$, where $\mathcal{S}$ and $\mathcal{S}'$ denote the constraint store prior to and after the rule application, respectively. In our setting CLP($\mathcal{D}$), where G is either *Holds* or *Poss*, the resulting constraint store $\mathcal{S}'$ is obtained based on the following equivalences:

- if there is a substitution Θ such that $\mathcal{D} \models (\forall) \bigvee_{\theta \in \Theta} G\theta$ then we can exploit that
$$\mathcal{D} \models (G \wedge \mathcal{S}) \equiv ((G \wedge \Theta) \wedge \mathcal{S}); \text{and} \tag{2}$$
- if $\mathcal{D}$ does not entail that the constraint is unsatisfiable (i.e. $\mathcal{D} \not\models \neg\mathcal{S} \wedge G$) we can use the following trivial equivalence
$$\mathcal{D} \models G \wedge \mathcal{S} \equiv G \wedge \mathcal{S}. \tag{3}$$

Prior to defining the CLP-based proof calculus for agent logic programs, let us discuss how disjunctive substitutions are applied. Most CLP(X)-languages come with an additional *Solve* transition, which maps derivation states into simpler, equivalent ones. A typical example is the application of substitutions, which, for instance, allows to transform the state $\langle\ \neg P(x), x = 1\ \rangle$ into the simpler $\langle\ \neg P(1), \mathit{true}\ \rangle$.

The application of a disjunctive substitution $\Theta = \{\theta_1, \ldots, \theta_k\}$ is a bit more involved: In the definition of our CLP($\mathcal{D}$) calculus, if we have obtained a disjunctive substitution $\Theta = \{\theta_1, \ldots, \theta_n\}$ for special atom G, we employ reasoning by cases: we split the current substate of the derivation into a disjunction of substates, one for each θ_i. We extend the substates by an additional argument $\mathcal{C}$ (the case store), for recording the case $G \wedge \theta_i$. In a substate $<$ Negative Clause$, \mathcal{S}, \mathcal{C} >$ special atoms are evaluated against the action theory augmented by the respective case — $\mathcal{D} \cup \{\mathcal{C}\}$. It is crucial to observe that the disjunction of the action theories $\mathcal{D}_i$ augmented by the respective cases $\mathcal{C}_i$ is equivalent to the original $\mathcal{D}$.

A *derivation state* in CLP($\mathcal{D}$) is a disjunction, denoted by $\dot{\vee}$, of sub-states $\langle\ \neg G_1 \vee \ldots \vee \neg G_n, \mathcal{S}, \mathcal{C}\ \rangle$. The derivation rules of the proof calculus are depicted in Figure 2. Due to lack of space, we omit the straightforward rule for regular program atoms. A *derivation* of a query Γ starts with $\langle\ \neg\Gamma, \mathit{true}, \mathit{true}\ \rangle$ and, if successful, ends in a state $\langle\ \Box, \mathcal{S}_1, \mathcal{C}_1\ \rangle \dot{\vee} \ldots \dot{\vee} \langle\ \Box, \mathcal{S}_m, \mathcal{C}_m\ \rangle$, ($m \geq 1$). The formula $\mathcal{S}_1 \vee \ldots \vee \mathcal{S}_m$ is the *computed answer*. A *failed* derivation ends with a sub-state to which none of the rules can be applied.

Substitution Rule:

$$\frac{< (\neg G_1 \vee \ldots \vee \neg G_{j-1} \vee \neg G_j \vee \neg G_{j+1} \vee \ldots \vee \neg G_n), \mathcal{S}, \mathcal{C} >}{< \dot{\bigvee}_i < (\neg G_1 \vee \ldots \vee \neg G_{j-1} \vee \neg G_{j+1} \vee \ldots \vee \neg G_n), \mathcal{S} \wedge G_j \wedge \theta_i, \mathcal{C} \wedge G_j \wedge \theta_i >}$$

where $\mathcal{D} \cup \{\mathcal{C}\} \models (\forall) \bigvee_{\theta_i \in \Theta} G_j\theta_i$ with most general disjunctive substitution Θ

Constraint Rule:

$$\frac{< (\neg G_1 \vee \ldots \vee \neg G_{j-1} \vee \neg G_j \vee \neg G_{j+1} \vee \ldots \vee \neg G_n), \mathcal{S}, \mathcal{C} >}{< (\neg G_1 \vee \ldots \vee \neg G_{j-1} \vee \neg G_{j+1} \vee \ldots \vee \neg G_n), \mathcal{S} \wedge G_j, \mathcal{C} >}$$

if $\mathcal{D} \cup \{\mathcal{C}\} \not\models \neg(\exists)\ \mathcal{S} \wedge G_j$.

Fig. 2. Inference rules for CLP($\mathcal{D}$) (the given rules operate on single sub-states).

Example 2 (continued) Recall that this action domain contains the initial state axiom $\mathit{Holds}(\mathit{At}(\mathit{Gold}, 4), S_0) \vee \mathit{Holds}(\mathit{At}(\mathit{Gold}, 5), S_0)$. Further, consider the simple program clause

```
p(Y) :- ?(at(gold,Y)).
```
(4)

along with the query `:- ?(at(gold,X)), p(X).` Obviously there exists x such that $\mathit{Holds}(\mathit{At}(\mathit{Gold}, x), S_0)$ and $P(x, S_0, S_0)$. This is a successful derivation:[7]

$$\begin{array}{r} \langle\ \neg\mathit{Holds}(\mathit{At}(\mathit{Gold}, x), S_0) \vee \neg P(x, S_0, s), \mathit{true}\ \rangle \\ \mapsto \\ \langle\ \neg P(4, S_0, s), \mathit{Holds}(\mathit{At}(\mathit{Gold}, x), S_0) \wedge x = 4\ \rangle \\ \dot\vee\ \langle\ \neg P(5, S_0, s), \mathit{Holds}(\mathit{At}(\mathit{Gold}, x), S_0) \wedge x = 5\ \rangle \\ \mapsto \\ \langle\ \neg\mathit{Holds}(\mathit{At}(\mathit{Gold}, 4), S_0), \mathit{Holds}(\mathit{At}(\mathit{Gold}, 4), S_0) \wedge \theta_1\ \rangle \\ \dot\vee\ \langle\ \neg\mathit{Holds}(\mathit{At}(\mathit{Gold}, 5), S_0), \mathit{Holds}(\mathit{At}(\mathit{Gold}, 5), S_0) \wedge \theta_2\ \rangle \\ \mapsto \\ \langle\ \Box, \mathit{Holds}(\mathit{At}(\mathit{Gold}, 4), S_0) \wedge \theta_1\ \rangle \\ \dot\vee\ \langle\ \Box, \mathit{Holds}(\mathit{At}(\mathit{Gold}, 5), S_0) \wedge \theta_2\ \rangle \end{array}$$

where θ_1 is $x = 4 \wedge s = S_0$, and θ_2 is $x = 5 \wedge s = S_0$. This example illustrates the necessity of reasoning by cases: in both sub-states, the last step would not be possible without adding the case store to the domain axioms prior to verifying the respective *Holds* instance. ■

Piggybacking on the general proofs for CLP(X), and based on the equivalences in (2) and (3), the CLP($\mathcal{D}$) approach is sound and complete in the following sense, provided that the underlying reasoning about actions is sound and complete. In the following, let Π be an expanded agent logic program, $\mathcal{D}$ a background domain axiomatization, and Γ an expanded query.

Theorem 1 (Soundness of CLP($\mathcal{D}$)). *If Γ has a successful derivation with computed answer $\bigvee_{i=1..k} \mathcal{S}_i$, then $\Pi \cup \mathcal{D} \models (\forall) \bigvee_{i=1..k} \mathcal{S}_i \supset \Gamma$.*

Theorem 2 (Completeness of CLP($\mathcal{D}$)). *If $\Pi \cup \mathcal{D} \models (\forall)\mathcal{S} \supset \Gamma$ and $\mathcal{S}$ is satisfiable wrt. $\mathcal{D}$, then there are successful derivations for Γ with computed answers $\mathcal{S}_1, \ldots, \mathcal{S}_n$ such that $\mathcal{D} \models (\forall)(\mathcal{S} \supset (\mathcal{S}_1 \vee \ldots \vee \mathcal{S}_n))$.*

5 Planning with ALPs

Agent logic programs can be used to solve two complementary tasks: They can be used to control an intelligent agent online; or the agent may use them to infer a plan that helps it achieve its goals. For online agent control it is a sound strategy to non-deterministically pick one path in the proof tree of agent logic programs, restricted to non-disjunctive substitutions. Essentially, this is the same as what is being done in GOLOG and FLUX. In this section we consider how agent logic programs can be utilized by agents for offline deliberation on planning problems.

5.1 Inferring Plans

It is surprisingly easy to formulate a generic agent logic program for arbitrary planning problems:

[7] We only show one of the constraint and case store (always identical in this example).

Example 3 [Generic ALP for Planning] The following is the generic ALP for planning problems, where `Phi` shall denote the respective goal description:

```
plan :- ?(Phi).
plan :- do(A), plan.
```

Put in words, we execute arbitrary actions until the goal ϕ is reached.

The soundness and completeness results from the previous section assure us that the query `?-plan.` can be proved if and only if it is entailed by this ALP together with the action theory. Somewhat surprisingly, there remains the question on how the inferred plan — if it exists — can be communicated to the programmer. The usual notion of a computed answer in logic programming does not provide any information concerning the plan inferred by deriving this query. In situation-based action theories $\mathcal{D}$ we could simply print out the final situations. If the natural or the real numbers serve as time structures this does not work, though.

For solving planning problems the sequence of the evaluated $Poss(a, s_1, s_2)$ atoms provides the programmer with the desired information (or, in the case of disjunctive plans, the corresponding tree does). The construction of such a planning tree is easily included into our proof calculi: In the case of CLP($\mathcal{D}$) the constraint stores already contain this information, and in the case of query complete domains the calculus is easily extended to construct this sequence.

5.2 Planning Completeness

The soundness and completeness results from the previous section assure us that a query can be proved if and only if it is entailed by the ALP together with the action theory. In this section we consider the following question: Assume that the action theory entails that some goal is achievable. Does the ALP from example 3 allow to infer a plan achieving this goal? It turns out that, unfortunately, in general the query `?-plan.` need not have a successful derivation, even if the action theory entails that the goal is achievable, i.e. $\mathcal{D} \models (\exists s) Holds(\phi, s)$. We next identify a number of natural conditions on action theories that will preclude this kind of situation.

Definition 8 [Properties of Action Theories] An action theory with precondition axioms $\mathcal{D}_{\text{Poss}}$ and time structure axiomatization $\mathcal{D}_{\text{Time}}$ is

- *progressing* if $\mathcal{D}_{\text{Poss}} \cup \mathcal{D}_{\text{Time}} \models Poss(a, s_1, s_2) \supset s_1 < s_2$.
- *sequential* if it is progressing and no two actions overlap; that is

$$\mathcal{D}_{\text{Poss}} \cup \mathcal{D}_{\text{Time}} \models Poss(a, s_1, s_2) \wedge Poss(a', s_1', s_2') \supset (s_2 < s_2' \supset s_2 \leq s_1') \wedge (s_2 = s_2' \supset a = a \wedge s_1 = s_1').$$

- *temporally determined* if all precondition axioms are of the form

$$Poss(A(\bar{x}), s_1, s_2) \equiv \pi_A[s_1] \wedge \bigvee_{\text{i}} \varphi_i, \tag{5}$$

where $\pi_A[s_1]$ does not mention s_2 , and each φ_i is an equality atom equating the time variable s_2 to a function with arguments among s_1 and $\bar{x}$.

– *anytime* if it is sequential and action applicability is not tied to a specific time-point; that is $\mathcal{D}_{\text{Poss}} \cup \mathcal{D}_{\text{Time}}$ entail

$$(\forall)(\exists s_2)(Poss(a, s_1, s_2) \wedge [Holds(f, s_1) \equiv Holds(f, s_1')]) \supset (\exists s_2')Poss(a, s_1', s_2').$$

Let us illustrate the last two of these notions by the following example:

Example 4 [Properties of Action Theories] A precondition axiom is not temporally determined if it is e.g. of the form $(\forall)Poss(A, s_1, s_2) \equiv s_2 > s_1$. Next, consider the precondition axioms $Poss(A, s_1, s_2) \equiv s_1 = 0 \wedge s_2 = 1$ and $Poss(B, s_1, s_2) \equiv s_1 = 2 \wedge s_2 = 3$, violating the conditions for a anytime action theories. The query `?- do(a), do(b)` is macro-expanded to $(\exists)Poss(A, 0, s_1') \wedge Poss(B, s_1', s_2')$, which does not follow under the given precondition axioms.

Define an action theory to be admissible if it satisfies all of the above properties. Now, if an admissible action theory entails that there exists a time where a goal holds, by an agent logic program a plan achieving that goal can be inferred:

Theorem 3 (Planning Completeness of ALPs). *Let $\mathcal{D}$ be an admissible action theory. Further, let Π be the generic planning ALP from example 3, and let the query Γ be* `?- plan.`*. Assume that $\mathcal{D} \models (\exists s)\text{Holds}(\phi, s)$ for planning goal ϕ, and $\Pi \cup \mathcal{D} \models (\forall)\mathcal{S} \supset \Gamma$ and $\mathcal{S}$ is satisfiable wrt. $\mathcal{D}$. We then know that there exist successful derivations of the query $Plan(S_0, s)$ in CLP($\mathcal{D}$) with computed answers $\mathcal{S}_1, \ldots, \mathcal{S}_n$ such that $\mathcal{D} \models (\forall)(\mathcal{S} \supset (\mathcal{S}_1 \vee \ldots \vee \mathcal{S}_n))$. The plans computed by these derivations can be combined into a plan achieving the goal ϕ. Note that this plan can be disjunctive, and conditional on the constraint store. For LP($\mathcal{D}$) and query-complete action theories $\mathcal{D}$ a similar result holds.*

Admissible action theories are not overly restrictive: while some of the expressivity of the Event Calculus and concurrent planning languages like full PDDL is lost, most standard agent and planning languages, as well as all of the basic Fluent and Situation Calculus are preserved.[8]

The correspondence between goal achievability and the existence of disjunctive plans is a well-known property of the Situation Calculus [16, 17]. Interestingly, GOLOG, which is based on Situation Calculus action theories, cannot be used to infer disjunctive plans. Agent logic programs, on the other hand, instantiated with any admissible action theory can handle this task.

Example 5 [Disjunctive Plan] Consider a further simplified version of the action theory from example 1, where the agent can move instantaneously to any location: The precondition of moving is axiomatized as

$$Poss(Go(y), s, t) \equiv (\exists x)(Holds(At(Agent, x), s) \wedge t = Do(Go(y), s),$$

and the effect axiom is as before. Let the initial state be given by

$$Holds(At(Agent, 1), S_0) \wedge (Holds(At(Gold, 4), S_0) \vee Holds(At(Gold, 5), S_0)).$$

On top of this action theory, consider the following agent logic program:

[8] Discussing in detail which (parts of) existing action formalisms are admissible is beyond the scope of this paper.

```
goToGold :- ?(at(gold,X)), do(go(X)).
```

It is not hard to see that there exists a successful derivation of the query `?-goToGold.`, from which the plan $Poss(Go(4), S_0) \vee Poss(Go(5), S_0)$ can be extracted, informing us that the agent should go to either location 4 or 5 to achieve its goal.

6 Conclusion

We have developed a declarative programming paradigm which can be used to specify agent strategies on top of underlying background action theories. A declarative semantics in pure classical logic has been given and complemented with a sound and complete operational semantics for two different settings: one, where we admit only query-complete background theories and which, as a consequence, harmonizes with basic logic programming; and one that is fully general, appealing to constraint logic programming. Agent logic programs are generic in that they can be used in combination with different (sufficiently expressive) action calculi. This paves the way for implementations of agent logic programs which combine SLD-resolution (or CLP-implementation techniques) with existing reasoners for actions. Specifically, we are currently developing an implementation of agent logic programs on the basis of recent results on how to build a decidable action calculus on top of Description Logics[18–20].

It is worth pointing out that the declarative semantics we have used in this paper is not the only meaningful interpretation of an agent logic program. In fact, macro-expansion via Definition 2 implicitly requires an agent to execute actions in a strictly sequential order. For future work, we intend to investigate alternative interpretations, which support overlapping actions and temporal gaps between two actions. This will allow us to broaden the class of action theories on which agent logic programs are planning complete.

With regard to related work, GOLOG [2] and FLUX [8] are two major exponents of agent programming languages based on general action calculi. In contrast to agent logic programs, GOLOG is a procedural language, whose elements derive from the classical programming language Algol. GOLOG has been implemented in Prolog on the basis of a Situation Calculus-style axiomatization of its programming elements. We believe that this can form the basis of a generic agent logic program for GOLOG, which would then provide a nice reconciliation of the procedural and the declarative programming paradigms. FLUX programs, on the other hand, are full Prolog programs together with a sound implementation of a fragment of the Fluent Calculus. As a consequence, they only admit an operational semantics based on the notion of Prolog computation trees [8]. However, FLUX restricted to pure definite logic programs is an example of a sound implementation of a fragment of agent logic programs.

A prominent feature of agent logic programs that is absent from existing agent programming languages are the disjunctive substitutions in $\mathrm{CLP}(\mathcal{D})$, which can be viewed as conditional plans. Agent logic programs also stand apart by their clear separation between world dynamics and employed strategy.

References

1. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence **4** (1969) 463–502
2. Reiter, R.: Knowledge in Action. MIT Press (2001)
3. Mueller, E.: Commonsense Reasoning. Morgan Kaufmann (2006)
4. Dastani, M., de Boer, F., Dignum, F., Meyer, J.J.: Programming agent deliberation: An approach illustrated using the 3APL language. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2003) 97–104
5. Morley, D., Meyers, K.: The SPARK agent framework. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2004) 714–721
6. Bordini, R., Hübner, J., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley (2007)
7. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence **2** (1971) 189–208
8. Thielscher, M.: FLUX: A logic programming method for reasoning agents. Theory and Practice of Logic Programming **5** (2005) 533–565
9. Lloyd, J.: Foundations of Logic Programming. Springer (1987)
10. Thielscher, M.: Reasoning Robots: The Art and Science of Programming Robotic Agents. Volume 33 of Applied Logic Series. Kluwer (2005)
11. McDermott, D.: The 1998 AI planning systems competition. AI Magazine **21** (2000) 35–55
12. Genesereth, M., Love, N., Pell, B.: General game playing: Overview of the AAAI competition. AI Magazine **26** (2005) 62–72
13. Stärk, R.: A direct proof for the completeness of SLD-resolution. In: Third Workshop on Computer Science Logic. (1990)
14. Jaffar, J., Lassez, J.L.: Constraint logic programming. In: Proceedings of the 14th ACM Principles of Programming Languages Conference, Munich (1987)
15. Green, C.: Theorem proving by resolution as a basis for question-answering systems. Machine Intelligence **4** (1969) 183–205
16. Reiter, R.: The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed.: Artificial Intelligence and Mathematical Theory of Computation. Academic Press (1991) 359–380
17. Savelli, F.: Existential assertions and quantum levels on the tree of the situation calculus. Artificial Intelligence **170** (2006) 643–652
18. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Updating description logic ABoxes. In: Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 06), Lake District of the UK (2006)
19. Baader, F., Lutz, C., Milicic, M., Sattler, U., Wolter, F.: Integrating description logics and action formalisms: First results. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Pittsburgh (2005) 572–577
20. Drescher, C., Liu, H., Baader, F., Guhlemann, S., Petersohn, U., Steinke, P., Thielscher, M.: Putting abox updates into action. In: Proceedings of the 7th International Symposion on Frontiers of Combining Systems (FroCoS 2009), Trento, Italy (2009)