Integrating Action Calculi and AgentSpeak: Closing the Gap

Michael Thielscher[*]
School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW, 2052, Australia
mit@cse.unsw.edu.au

Abstract

Existing action calculi provide rich, declarative formalisms for reasoning about actions. BDI-based programming languages like AgentSpeak, on the other hand, are procedural and geared towards practical applications of cognitive agents. In this paper, we close the gap between these two lines of research by integrating action calculi and AgentSpeak programs. Specifically, we develop a new and purely declarative semantics for AgentSpeak, which paves the way for combining this language with any suitable action calculus in a strictly modular fashion. As the main technical result, we prove that the new declarative semantics is correct wrt. the standard operational semantics for AgentSpeak. This provides the basis for a modular integration of a BDI-based agent programming language with sophisticated methods for reasoning about actions.

Introduction

For the design and implementation of cognitive agents, two mostly independent directions have been taken in the past. On the one hand, BDI-based programming languages (for an overview see (Mascardi, Demergasso, and Ancona 2005)) are geared towards the practical deployment of agent technology. They allow programmers to directly specify the intended behaviour of an agent in form of procedures to be selected and executed under pre-defined circumstances. In BDI-based AgentSpeak (Rao 1996), for example, agents maintain an internal, symbolic world model (their *beliefs*) and update this model upon perceiving the environment and executing actions, but they do not employ a sophisticated action theory for this purpose. Rather, the designer of an AgentSpeak program specifies in a simple STRIPS-like fashion (Fikes and Nilsson 1971) how the agent should revise its beliefs at specific stages of the program.

On the other hand, there is a long tradition of research in Knowledge Representation on the automation of reasoning about actions and their effects. The classical Situation Calculus (McCarthy and Hayes 1969) and other, similarly expressive action formalisms have evolved into high-level programming languages, such as GOLOG (Levesque et al. 1997) or FLUX (Thielscher 2005a; 2005b). However, their focus is on complex, long-term strategies rather than collections of short-term behaviours, and they are much less used in practise compared to BDI-based programming languages.

Surprisingly little has been done to exploit the advantages of both lines of research in combination, where a BDI-style language is used to specify the behaviour of an agent and a separate, rich action theory is used to reason about the agent's actions and their effects. In this paper, we intend to close this gap by integrating action theories into AgentSpeak (Rao 1996), a language which has been highly influential in the development of BDI-based agent programming languages (Bordini, Hübner, and Wooldridge 2007). Our main goal is to achieve a clear and modular combination of the two main aspects of cognitive agency: the *procedural knowledge,* which concerns the behaviour of the agent and is specified by a BDI-based program, and the *action knowledge*, which is given by a domain theory in a rich action calculus and concerns the use and update of the internal world model according to the effects of actions and the agent's percepts.

Our combination of these two formalisms follows the scheme depicted in Figure 1. We will show how a given AgentSpeak program can be translated into a so-called *Agent Logic Program (AgentLP)* (Drescher, Schiffel, and Thielscher 2009). This will be combined with a generic AgentLP that constitutes a declarative account of the general execution strategy in AgentSpeak. The resulting AgentLP admits a purely logical reading, which can then be straightforwardly combined with a background axiomatisation composed of domain axioms and the foundational axioms for the action calculus of one's choice. The set of formulae thus obtained provides a purely declarative semantics for the integration of the two sides.

Our result closes the gap between the two branches of research on cognitive agents: on the one hand, it allows to underpin a practical BDI-based language with sophisticated methods for reasoning about actions and their effects, thus aiding the programmer by allowing a much richer than STRIPS-style specification. On the other hand, it provides a way to augment elaborate action formalisms like the Situation Calculus with a practical method for specifying agent behaviours using the BDI-model.

[*]The author acknowledges support by the Australian Research Council under grant FT0991348.

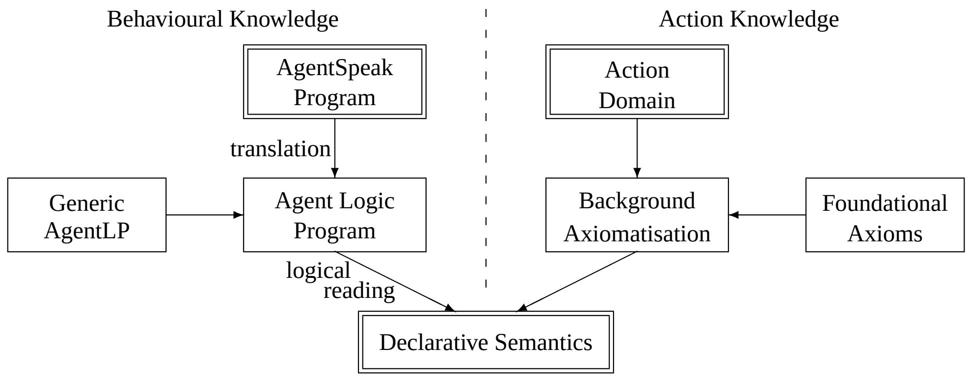


Figure 1: Combining AgentSpeak programs with rich action domain axiomatisations.

The rest of the paper is organised as follows. In the next section, we briefly recapitulate the basic syntax of AgentSpeak and Agent Logic Programs, respectively. Thereafter, we present a generic AgentLP for the general execution strategy of a BDI-based agent and show how AgentSpeak procedures can be translated into appropriate AgentLP clauses. We then illustrate how this allows to combine arbitrary action theories with AgentSpeak procedures. Finally, we assess the declarative semantics thus obtained against the standard operational semantics for AgentSpeak and prove our main result, namely, that the former provides a correct characterisation of the latter. The paper ends with a short discussion.

Background

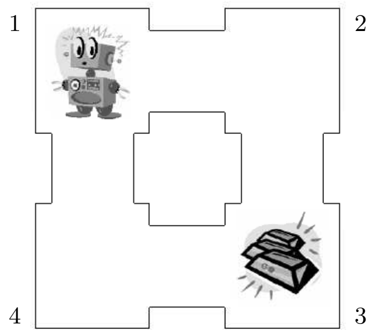
AgentSpeak

AgentSpeak (Rao 1996) is arguably the most influential programming language for behaviour-based agents that follows the BDI-model. In this paper, we focus on the basic version of this language, which is often referred to as AgentSpeak(L). This suffices for our purpose, and in fact highlights a crucial feature of our intended separation of the behaviour and the reasoning aspect in an agent: existing language extensions, e.g. speech-act based communication (Moreira, Vieira, and Bordini 2003), are a matter of how the agent reasons about its beliefs and goals, and hence a matter of the underlying action theory rather than the agent's procedural knowledge.

A world model in AgentSpeak is composed of *belief atoms* representing the relevant properties of the environment in which the agent lives. Beliefs can be held on both static as well as dynamic properties. Besides predicates for beliefs, a domain signature contains predicates representing the range of actions of the agent.

Definition 1. An *AgentSpeak domain signature* consists of a finite set of *belief predicates* and a finite set of *action predicates*. A *belief literal* is a belief predicate with arguments (i.e., an atom) or its negation. □

As a simple example, consider the following scenario.



The task of the agent is to grab the gold (whose location the agent does not know initially) and to take it to the home square. Consider the following belief predicates as means to describe the states of this environment: $\mathit{At}(x,y)$, where $x \in \{\mathit{Agent}, \mathit{Gold}\}$ and $y \in \{1,\ldots,4\}$, along with $\mathit{Has}(\mathit{Agent}, \mathit{Gold})$. The initial belief base of the agent may then be given by the single atom $\mathit{At}(\mathit{Agent}, 1)$. Let the possible actions of the agent be to move clockwise to the next cell and to grab and drop the gold, respectively represented by these action predicates: *Go*, $\mathit{Grab}(\mathit{Gold})$, $\mathit{Drop}(\mathit{Gold})$. Finally, we assume a simple sensor that signals the agent whenever it is in a cell with gold.

In order to keep the definition of individual behaviours conceptually simple, behaviours in AgentSpeak are mere sequences of elementary program statements. Such a statement can be an action of the agent like the aforementioned, or the addition of a new goal. There are two kinds of goals: *achieving* a specific property, written $!f$, where f is a belief atom; or *testing* a property, written $?f$. As an example, consider the sequence

$$\mathit{Grab}(\mathit{Gold}),\ !\mathit{At}(\mathit{Agent}, 1),\ \mathit{Drop}(\mathit{Gold}) \qquad (1)$$

This behaviour causes the agent to first grab the gold, then pursue the goal to be at the home location 1, and finally drop the gold.

Each behaviour in AgentSpeak needs to be accompanied by a condition under which it can be adopted. This includes a *triggering event* and a *context* defining preconditions for the applicability of the procedure. Triggering events can be the addition or removal of a belief or goal.

Definition 2. If f is a belief atom, then $!f$ and $?f$ are *goals*. A *triggering event* is any of $+f$, $-f$, $+g$, $-g$, where f is a belief atom and g a goal. A *procedure* is an expression of the form

$$e : b_1, \ldots, b_m \leftarrow p_1, \ldots, p_n$$

where e is a triggering event, $b_1, \ldots, b_m$ (the *context*) are belief literals ($m \geq 0$), and $p_1, \ldots, p_n$ (the *body*) is a sequence of action atoms or goals ($n \geq 0$). An empty context or body is denoted by *True*. $\square$

Here is an example procedure for our gold mining agent:

$$\begin{aligned} +!\,\mathit{Has}(\mathit{Agent}, \mathit{Gold}) &: \mathit{True} \\ &\leftarrow \mathit{Go}, !\,\mathit{Has}(\mathit{Agent}, \mathit{Gold}) \end{aligned} \tag{2}$$

The agent may adopt this procedure whenever its goal is to get the gold (that is, to achieve $\mathit{Has}(\mathit{Agent}, \mathit{Gold})$). The procedure body tells the agent to move to the next cell and to establish the same goal from there. Another example is this procedure, which uses sequence (1) from above:

$$\begin{aligned} +!\,\mathit{Has}(\mathit{Agent}, \mathit{Gold}) &: \mathit{At}(\mathit{Agent}, x), \mathit{At}(\mathit{Gold}, x) \\ &\leftarrow \mathit{Grab}(\mathit{Gold}), !\,\mathit{At}(\mathit{Agent}, 1), \mathit{Drop}(\mathit{Gold}) \end{aligned} \tag{3}$$

This behaviour can be adopted if the goal is to get the gold in a context where the agent happens to be at some location x of which it knows that gold can be found there. Two final procedures may be used to tell the agent what to do to reach a specific location:

$$\begin{aligned} +!\,\mathit{At}(\mathit{Agent}, x) &: \mathit{At}(\mathit{Agent}, x) \leftarrow \mathit{True} \\ +!\,\mathit{At}(\mathit{Agent}, x) &: \neg \mathit{At}(\mathit{Agent}, x) \\ &\leftarrow \mathit{Go}, !\,\mathit{At}(\mathit{Agent}, x) \end{aligned} \tag{4}$$

The first procedure says that the goal to be at some location is achieved if the agent is already there, while the second one says that the same goal can be achieved by a behaviour that first takes a single step to the adjacent cell and then to pursue again the goal to reach x.

The execution principle for AgentSpeak programs follows the generic Procedural Reasoning System (Georgeff and Lansky 1987), where the state of the agent at any time is characterised by the following three sets.

Definition 3. *Beliefs*, *desires*, and *intentions* in AgentSpeak can be defined as follows.

- $\mathcal{B}$, the beliefs, is a set of variable-free belief atoms.
- $\mathcal{I}$ is a set of intentions, each of which is a stack

  $$[P_1; \ldots; P_k]$$

  where $k \geq 0$ and each P_i $(1 \leq i \leq k)$ is a sequence of actions or goals. The first procedure body, P_1, is the one that can be executed next.[1]
- $\mathcal{D}$ is a set of desires, each of which is of the form $\langle e; i \rangle$ where e is a triggering event and i an intention.[2] $\square$

In general, the second component, i, of a desire encodes the remaining steps of the procedure which has generated event e. For example, executing the second step in the body of procedure (3) may yield the desire

$$\langle +!\,\mathit{At}(\mathit{Agent}, 1); [\mathit{Drop}(\mathit{Gold})] \rangle$$

This indicates that the agent desires $\mathit{At}(\mathit{Agent}, 1)$ in order to be able to continue with the remaining sequence $[\mathit{Drop}(\mathit{Gold})]$. So-called *external desires* are desires of the form $\langle e; [\,] \rangle$. These are generated as a result of perceiving the environment. An example may be $\langle +!\,\mathit{Has}(\mathit{Agent}, \mathit{Gold}); [\,] \rangle$, representing the initial request to the agent to get the gold.

The three standard components of a BDI-agent are accompanied by three selection functions: one that selects an element from the current desires; one that selects an element from the current intentions; and one that selects an applicable procedure instance for a triggering event in accordance with the following definition.

Definition 4. Consider a set $\mathcal{B}$ of ground (i.e., variable-free) belief atoms. Let e be a triggering event and p a procedure $d : b_1, \ldots, b_m \leftarrow p_1, \ldots, p_n$. Then p is *relevant* for e if $d\theta = e\theta$ for some most general unifier θ. If, furthermore, a substitution η exists such that $(\forall)\,(b_1 \wedge \ldots \wedge b_m)\theta\eta$ is true in $\mathcal{B}$,[3] then the procedure instance $p\theta\eta$ is *applicable* to e (wrt. $\mathcal{B}$). $\square$

For example, suppose the belief base of our gold mining agent contains $\mathit{At}(\mathit{Agent}, 3)$, and consider the triggering event $+!\,\mathit{At}(\mathit{Agent}, 1)$. While both procedures in (4) are relevant in this case, only the second one is applicable, determining the substitution $\theta = \{x/1\}$ and $\eta = \{\}$. The resulting behaviour is then given by the instance of the procedure body, viz. $\mathit{Go}, !\,\mathit{At}(\mathit{Agent}, 1)$.

The operational semantics of AgentSpeak can be summarised as follows (Rao 1996). At any time, the state of an agent is given by its current beliefs $\mathcal{B}$, desires $\mathcal{D}$, and intentions $\mathcal{I}$. The agent starts with a given set of beliefs while both the set of desires and the set of intentions are initially empty. The agent then runs through a sense-select-act cycle: *Sensing* means to update $\mathcal{B}$ according to the agent's percepts and to augment $\mathcal{D}$ by all external desires thus sensed. *Selecting* means to select one of the desires along with an applicable procedure instance, whose body is then added to the intentions. *Acting* means to choose an intention and to execute its first step.

Agent Logic Programs

Agent Logic Programs, or short: AgentLPs, combine reasoning about actions with the declarative programming paradigm (Drescher, Schiffel, and Thielscher 2009).

[1] In the original definition of AgentSpeak, an intention is a sequence of (partially instantiated) *procedures*, including triggering event and context (Rao 1996). Just taking (partially executed) procedure bodies leads to a conceptual simplification. We also remark that we write the stack in reversed order, where the leftmost element, P_1, denotes the top of the stack.

[2] In the literature on AgentSpeak, what we call desires is often called an event; we decided not to follow this convention to make clear that a desire is something complex, namely, a triggering event plus an intention.

[3] By definition, a negated belief atom $\neg b$ is true in $\mathcal{B}$ iff no ground instance of b occurs in $\mathcal{B}$.

AgentLPs are logic programs (see, e.g., (Lloyd 1987)) that can be combined with a rich background action theory and that use two special predicates: one, written $\underline{\texttt{do}}(\alpha)$, represents the execution of an action α by the agent, and the other, written $\underline{\texttt{knows}}(\varphi)$, denotes a condition φ on the state of the environment in which the agent lives.[4]

Definition 5. Consider an action theory signature Σ, including the pre-defined sorts ACTION and FLUENT,[5] and a logic program signature Π.

- If $\texttt{p}$ is an n-ary relation from Π and $\texttt{t}_1, \ldots, \texttt{t}_n$ are terms from $\Sigma \cup \Pi$, then $\texttt{p}(\texttt{t}_1, \ldots, \texttt{t}_n)$ is a *program atom*.
- $\underline{\texttt{do}}(\alpha)$ is a *program atom* if α is an ACTION in Σ.
- $\underline{\texttt{knows}}(\varphi)$ and $\neg\underline{\texttt{knows}}(\varphi)$ are *program atoms* if φ is a FLUENT in Σ.[6]
- Clauses, programs, and queries are then defined as usual for definite logic programs, with the restriction that the two special program atoms cannot occur in the head of a clause. □

The interested reader may take a peek at Figure 2 at this point to see an example AgentLP.

Agent Logic Programs are understood and executed with the help of a background action theory, which provides the agent with knowledge of its actions and their effects. Any sufficiently expressive representation formalism for actions can be used in combination with AgentLPs. The only prerequisite is that it provides a special sort TIME (which can be linear, or branching as in the Situation Calculus) and the atomic predicates $\mathit{Knows}(f, s)$ and $\mathit{Poss}(a, s, t)$, respectively indicating that the agent knows FLUENT f to be true at TIME s and that the execution of ACTION a is possible starting at TIME s and ending at TIME t. These two predicates provide the connection of an AgentLP to the underlying background theory (see also Figure 1): the special atoms $(\neg)\underline{\texttt{knows}}(\varphi)$ are evaluated on the basis of *Knows*, and $\underline{\texttt{do}}(\alpha)$ is understood on the basis of *Poss*. With this, the semantics of an AgentLP is basically that of standard logic programming augmented by the background theory on the actions and the knowledge of the agent. However, since logic programs are static in nature while reasoning about actions describes changes over time, an AgentLP needs to be expanded into a set of axioms in which the temporal aspect is made explicit; the formal details will be given later in the context of the generic AgentLP for AgentSpeak, which will be introduced next.

[4]For the latter predicate, the symbol "?" is used in (Drescher, Schiffel, and Thielscher 2009). We opted to replace this in order to avoid confusion with the syntax in AgentSpeak and to indicate that conditions are evaluated against the knowledge of the agent.

[5]In action theories it is customary to use the name *fluent* for individual state properties like $\mathit{At}(x, y)$ or $\mathit{Has}(\mathit{Agent}, \mathit{Gold})$.

[6]The standard definition of AgentLPs is more general in that φ may be an arbitrary formula (Drescher, Schiffel, and Thielscher 2009). The restricted definition used here suffices for the design of an AgentLP as declarative semantics for AgentSpeak. Note also that $\neg\underline{\texttt{knows}}(\varphi)$ is formally treated as a program *atom* and, hence, that AgentLPs are definite (i.e., negation-free) logic programs.

An AgentLP for AgentSpeak

Because AgentSpeak borrows a number of notions from logic programming, such as unification and sequences of goals, Agent Logic Programs provide an ideal link between this BDI-based language and action logics. In this section, we present an AgentLP that, together with an underlying background theory, provides an axiomatic, declarative description of the AgentSpeak execution process and that can be readily used to combine a BDI-based control mechanism with a rich theory for actions.

The basic elements of a domain axiomatisation in any action calculus are the fluents and the actions. The former replace the belief predicates used in an AgentSpeak domain signature. For the purpose of combining an action theory with a BDI-based strategy, we assume the existence of an extra fluent $\mathit{Goals}(e)$, which in every situation defines the externally given addition (or removal) of goals. To this end, argument e is assumed to be a (possibly empty) list of triggering events with exactly the same syntax as in Definition 2.

The terms for the actions in an underlying domain correspond to the action predicates of an AgentSpeak program. We assume that these include an extra action *SenseAct*, which can always be executed and provides the agent with sensing information, including information about new external desires given by the special fluent $\mathit{Goals}(e)$. A precise example axiomatisation of $\mathit{Goals}(e)$ and *SenseAct* will be given in a later section.

The Generic AgentLP

We are now ready to present a generic Agent Logic Program that encodes the execution principle of AgentSpeak; see Figure 2. The program defines the three predicates `sense`, `select`, and `act`, which axiomatise the three steps in the sense-select-act cycle that describes the operation of an agent in AgentSpeak. Each of these predicates has as arguments a list of the current desires and a list of the current intentions.[7] Each intention is itself a list of the form $[P_1, \ldots, P_k]$ as in Definition 3. The desires are expressions of the form (e, i), where e is a triggering event and i an intention, again as in Definition 3. Lists are defined as usual in logic programming, that is, using a binary function, written $[h|t]$, concatenating a head h with a tail list t. The empty list is denoted by the constant $[\,]$. The clauses of the main AgentLP for AgentSpeak can thus be interpreted as follows.

The first step in the cycle, $\texttt{sense}(d, i)$, requires to do the special action *SenseAct*. The external triggering events thus sensed are added to the current desires. This is achieved with the help of `append`, which is defined so as to have the side-effect that each individual triggering event g is transformed into an external desire of the form $(g, [\,])$.

In the second step, $\texttt{select}(d, i)$, if there are no desires or intentions left, the cycle ends. If there are no desires but an incompletely executed intention, the agent continues with acting. Otherwise, a desire (x, j) is selected together with

[7]The current beliefs are missing because they will be supplied by the background action theory that will accompany the generic AgentLP.

```
agent_speak ⇐ sense([],[])

sense(D,I)  ⇐ do(sense_act), knows(goals(E)), append(D,E,F), select(F,I)

select([],[])    ⇐
select([],[I|J]) ⇐ act([],[I|J])
select(D,I)      ⇐ member((X,J),D,E), procedure(X,P), act(E,[[P|J]|I])

act(D,I) ⇐ member([],    I,J), sense(D,J)
act(D,I) ⇐ member([[]|P],I,J), sense(D,[P|J])
act(D,I) ⇐ member([[   A|P]|Q],I,J), do(A), sense(D,[[P|Q]|J])
act(D,I) ⇐ member([[?(F)|P]|Q],I,J), knows(F), sense(D,[[P|Q]|J])
act(D,I) ⇐ member([[!(F)|P]|Q],I,J), sense([(+(!(F)),[P|Q])|D],J)

append(D,[],D) ⇐
append(D,[G|H],[(G,[])|E]) ⇐ append(D,H,E)
member(X,[X|Xs],Xs) ⇐
member(X,[Y|Xs],[Y|Ys]) ⇐ member(X,Xs,Ys)
```

Figure 2: The generic Agent Logic Program for AgentSpeak.

an applicable procedure instance p, resulting in the new intention $[p|j]$. To this end, the given AgentSpeak procedures are assumed to be encoded by clauses defining the predicate `procedure` as detailed below.

For the final step, the first and second clause for `act`(d,i) are used to resolve, respectively, an empty intention or an empty leading procedure body. The remaining clauses define the execution of the first element of the first procedure body in a selected intention:

- If it is an action a, it is performed using the special AgentLP predicate `do`(a).
- If it is a test goal $?f$, then the special AgentLP predicate `knows`(f) is used to test if the agent knows an instance of f that currently holds.
- If it is an achievement goal $!f$, then a new desire gets stipulated: Let the selected intention be $[[!f, p_2, \ldots, p_n]; P_2, \ldots; P_k]$, then the new desire is of the form $\langle +!f; i\rangle$ with the new intention i being $[[p_2, \ldots, p_n]; P_2; \ldots; P_k]$ (this is `[P|Q]` in the bottommost clause defining `act`(d,i)).

The selection of both a desire and an intention with which to continue is encoded by the auxiliary predicate `member`. For the sake of generality, its definition allows just any element to be selected in any step. This means, for example, that any intention can be suspended indefinitely as long as other intentions are present. This basic definition can easily be replaced by a more restrictive selection strategy of one's choice.

AgentSpeak Procedures as AgentLPs

The main AgentLP for AgentSpeak needs to be accompanied by specific clauses encoding the various procedures that compose an actual AgentSpeak program. The mapping is straightforward and modular.

Definition 6. A procedure $e : b_1, \ldots, b_m \leftarrow p_1, \ldots, p_n$ is *interpreted* as the AgentLP clause

$$\texttt{procedure}(e,[p_1,\ldots,p_n]) \Leftarrow \kappa(b_1),\ldots,\kappa(b_n)$$

where $\kappa(b_i)$ stands for `knows`(b_i) if b_i is positive and for $\neg$`knows`(b) if $b_i = \neg b$. □

In this way, a procedure is applicable to a given desire if the latter matches the triggering event e and if the agent knows that the context $b_1, \ldots, b_m$ holds. The body of a procedure is then encoded as a list of actions and goals, as required by the main AgentLP in Figure 2. As an example, consider the following AgentLP clauses, which correspond to the AgentSpeak procedures (2)–(4):

```
procedure(+(!(has(agent,gold))),
          [go,!(has(agent,gold))]) ⇐

procedure(+(!(has(agent,gold))),
          [grab(gold),
           !(at(agent,1)),drop(gold)])
  ⇐ knows(at(agent,X)),
    knows(at(gold,X))                      (5)

procedure(+(!(at(agent,X))),[])
  ⇐ knows(at(agent,X))

procedure(+(!(at(agent,X))),
          [go,!(at(agent,X))])
  ⇐ ¬knows(at(agent,X))
```

This encoding illustrates the elegance of using Agent Logic Programs to encode the execution process of AgentSpeak: the various notions related to the application of a procedure, that is, relevance, applicability, and instance (cf. Definition 4), coincide with the usual declarative and procedural semantics for logic programs.

To summarise, any given AgentSpeak program can be mapped onto a set of AgentLP clauses together with the generic AgentLP. The standard query to this program would be ⇐ `agent_speak`, which asks for a successful execution of the AgentSpeak program with initially empty desires and intentions.

Declarative Semantics for AgentSpeak: AgentLP + Action Theory

Agent Logic Programs are interpreted and executed with the help of a background theory, which provides the agent with knowledge of its actions and their effects (Drescher, Schiffel, and Thielscher 2009). In principle, AgentLPs can be combined with any underlying action representation formalism in which a time structure exists and which allows to give meaning to the two predicates $\mathit{Knows}(f, s)$ and $\mathit{Poss}(a, s, s')$. On this basis, the AgentLP for an AgentSpeak program can be understood as a purely logical axiomatisation with an underlying action theory.

Part 1: Logical Reading of the AgentLP

Because logic programs are static in nature while reasoning about actions describes changes over time, the declarative reading of an AgentLP is obtained by expanding it into a set of axioms in which the temporal aspect is made explicit. Two arguments of sort TIME are added to every "ordinary" program atom $p(\vec{x})$, and then $p(\vec{x}, s, t)$ means the restriction of the truth of the atom to the temporal interval between (and including) s and t. The two "special" program atoms receive special treatment: atom $(\neg)$`knows(`φ`)` is re-written to $(\neg)\mathit{Knows}(\varphi, s)$, with the intended meaning that φ is known (respectively, not known) to be 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.

Formally, an AgentLP is expanded by expanding each of its clauses `H` $\Leftarrow$ `B`$_1$`,...,B`$_n$ ($n \geq 0$) as follows. 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 $\mathit{Poss}(\alpha, s_i, s_{i+1})$;
  - `knows(`φ`)`, expand to $\mathit{Knows}(\varphi, s_i) \wedge s_i = s_{i+1}$.
  - $\neg$`knows(`φ`)`, expand to $\neg\mathit{Knows}(\varphi, s_i) \wedge s_i = s_{i+1}$.
- 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 taken as universally quantified implications as usual.

Applying this expansion to the AgentLP in Figure 2, the second clause, for example, is understood as

$$\begin{aligned}(\forall)\ & \mathit{Poss}(\mathit{SenseAct}, s_1, s_2) \wedge \\ & \mathit{Knows}(\mathit{Goals}(e), s_2) \wedge s_2 = s_3 \wedge \\ & \mathit{Append}(d, e, f, s_3, s_4) \wedge \mathit{Select}(f, i, s_4, s_5) \\ & \quad \supset \mathit{Sense}(d, i, s_1, s_5)\end{aligned} \tag{6}$$

Thus, $\mathit{Sense}(d, i)$ holds between s_1 and s_5 if *SenseAct* is possible from s_1 to s_2, if then fluent $\mathit{Goals}(e)$ holds in s_2, and if finally $\mathit{Append}(d, e, f)$ holds from s_3 $(= s_2)$ to s_4 and $\mathit{Select}(f, i)$ from s_4 to s_5. It is important to realise that the order in which atoms occur in the body of a clause needs to be respected when adding the time arguments to the individual atoms. However, once time has been incorporated the implication is a standard first-order formula and as such purely declarative. In a similar fashion, all other clauses of the main AgentLP are extended by TIME arguments to reflect their dynamic nature. The resulting set of formulae provides a purely declarative semantics for the AgentLP. Due to space restrictions we omit the details and just give the declarative reading of the AgentLP clauses for the individual AgentSpeak procedures in (5):

$$\begin{aligned}
&(\forall)\ \mathit{Procedure}(+!\,\mathit{Has}(\mathit{Agent}, \mathit{Gold}), \\
&\qquad\qquad [\mathit{Go}, !\,\mathit{Has}(\mathit{Agent}, \mathit{Gold})], s, s) \\[1ex]
&(\forall)\ \mathit{Knows}(\mathit{At}(\mathit{Agent}, x), s) \wedge \mathit{Knows}(\mathit{At}(\mathit{Gold}, x), s) \\
&\quad \supset \mathit{Procedure}(+!\,\mathit{Has}(\mathit{Agent}, \mathit{Gold}), \\
&\qquad\quad [\mathit{Grab}(\mathit{Gold}), !\,\mathit{At}(\mathit{Agent}, 1), \mathit{Drop}(\mathit{Gold})], s, s) \\[1ex]
&(\forall)\ \mathit{Knows}(\mathit{At}(\mathit{Agent}, x), s) \\
&\quad \supset \mathit{Procedure}(+!\,\mathit{At}(\mathit{Agent}, x), [\,], s, s) \\[1ex]
&(\forall)\ \neg\mathit{Knows}(\mathit{At}(\mathit{Agent}, x), s) \\
&\quad \supset \mathit{Procedure}(+!\,\mathit{At}(\mathit{Agent}, x), \\
&\qquad\qquad [\mathit{Go}, !\,\mathit{At}(\mathit{Agent}, x)], s, s)
\end{aligned} \tag{7}$$

A query $\Leftarrow$ `B`$_1$`,...,B`$_n$ is expanded just like a clause but without the instruction for the missing head; furthermore, all variables are existentially quantified and the first temporal variable, s_1, is instantiated by the least element, written S_0, of sort TIME (denoting the earliest time point in the underlying action theory). In this way, the query $\Leftarrow$ `agent_speak`, say, is understood as $(\exists s)\, \mathit{Agent\_Speak}(S_0, s)$ and thus denotes the question whether there exists a time point s such that this formula is entailed by the AgentLP plus the underlying action theory.

With the TIME arguments incorporated, the logical representation of procedures—together with the generic AgentLP—provides a purely declarative reformulation of the operational semantics of AgentSpeak. This completes the branch on the left-hand side in Figure 1: for any AgentSpeak program P, let $\mathit{Ax}(P)$ denote the set of first-order formulas obtained by mapping the procedures in P into AgentLP clauses according to Definition 6, conjoining these with the generic program of Figure 2, and taking the logical reading of the resulting clause set. As an example, recall our AgentSpeak program for gold mining. It can now be understood as the formulae (7) augmented by (the temporal expansion of) the AgentLP in Figure 2. As axiomatisation with the two special predicates *Knows* and *Poss*, it can be conjoined with a rich action theory that provides background knowledge (using the same two predicates) about the dynamic environment in which the agent lives.

Part 2: Background Action Theory

The purpose of the background action theory is to allow the agent to reason about what it knows and what actions are possible. Pure AgentSpeak has but a very simplistic action model (Rao 1996): the beliefs of an agent at any time are characterised by a set of ground atoms, and action specifications are confined to STRIPS-style addition and removal of atoms. This does not allow to express simple forms of incomplete knowledge, like $\mathit{At}(\mathit{Gold}, 3) \vee \mathit{At}(\mathit{Gold}, 4)$ or $(\exists x)\, \mathit{At}(\mathit{Gold}, x)$, nor is it suitable for domains where actions can have conditional or nondeterministic effects, ramifications, etc. Our new, declarative interpretation of AgentSpeak allows to combine it with just any expressive action representation formalism—provided it is based on logic so that it can simply be added to the logical reading of an AgentLP. In the following we will take, as example, the classical Situation Calculus (McCarthy and Hayes 1969) augmented by an axiomatisation of the agent's knowledge according to (Scherl and Levesque 2003).

The Situation Calculus is based on a branching time structure in which the elements of the basic sort TIME are called *situations*. These are formed using the constant S_0, denoting the initial situation, and the expression $\mathit{Do}(a, s)$, denoting the situation resulting from performing action a in situation s. Two standard predicates are $\mathit{Holds}(f, s)$, denoting that fluent f holds in situation s, and $\mathit{Poss}(a, s)$ (carrying just two arguments in the Situation Calculus), which means that action a is possible in situation s.

The axiomatisation of a dynamic domain with the help of the Situation Calculus is based on precondition and effect axioms. The former define the conditions on a situation under which an action is possible. Taking our gold mining domain as example again, these are suitable precondition axioms for the three actions of the agent:[8]

$$\begin{array}{l}
\mathit{Poss}(\mathit{Go}, s) \;\equiv\; \mathit{True} \\[1em]
\mathit{Poss}(\mathit{Grab}(x), s) \;\equiv \\
\quad x = \mathit{Gold} \wedge (\exists y)\,(\mathit{Holds}(\mathit{At}(\mathit{Agent}, y), s) \wedge \\
\qquad\qquad\qquad\qquad \mathit{Holds}(\mathit{At}(x, y), s)) \\[1em]
\mathit{Poss}(\mathit{Drop}(x), s) \;\equiv \\
\quad x = \mathit{Gold} \wedge \mathit{Holds}(\mathit{Has}(\mathit{Agent}, x), s)
\end{array} \tag{8}$$

Following (Reiter 1991), the effects of actions can be described in the Situation Calculus by *successor state axioms*, one for each domain fluent F. Their general form is $\mathit{Holds}(F, \mathit{Do}(a, s)) \equiv \gamma_F^+(a, s) \vee (\mathit{Holds}(F, s) \wedge \neg\gamma_F^-(a, s))$, where $\gamma_F^+(a, s)$ are the conditions (on action a and situation s) under which fluent F becomes true, and $\gamma_F^-(a, s)$ are the conditions under which fluent F becomes false. This axiomatisation technique provides a simple solution to the frame problem (McCarthy and Hayes 1969) because $\mathit{Holds}(F, \mathit{Do}(a, s))$ is equivalent to $\mathit{Holds}(F, s)$ whenever neither γ_F^+ nor γ_F^- applies (that is, whenever F is not affected by the action). Coming back to our running example, the effects of the actions on our two fluents are suitably specified by the following two successor state axioms:[9]

$$\begin{array}{l}
\mathit{Holds}(\mathit{At}(x, y), \mathit{Do}(a, s)) \;\equiv \\
\quad x = \mathit{Agent} \wedge a = \mathit{Go} \wedge (\exists z)(\mathit{Holds}(\mathit{At}(x, z), s) \wedge \\
\qquad\qquad\qquad\qquad\qquad y = (z \bmod 4) + 1) \\
\quad \vee \\
\quad x = \mathit{Gold} \wedge a = \mathit{Drop}(x) \wedge \mathit{Holds}(\mathit{At}(\mathit{Agent}, y), s) \\
\quad \vee \\
\quad (\,\mathit{Holds}(\mathit{At}(x, y), s) \wedge \neg\,[\, x = \mathit{Agent} \wedge a = \mathit{Go} \vee \\
\qquad\qquad\qquad\qquad x = \mathit{Gold} \wedge a = \mathit{Grab}(x)\,]\,)
\end{array}$$

[8]Here and in the following, all unbound variables are implicitly assumed to be universally quantified in axioms.

[9]For the following we tacitly assume the standard so-called *unique name axioms* for actions, that is, $(\forall x)\, \mathit{Go} \neq \mathit{Grab}(x)$ etc.

$$\begin{array}{l}
\mathit{Holds}(\mathit{Has}(x, y), \mathit{Do}(a, s)) \;\equiv \\
\quad x = \mathit{Agent} \wedge y = \mathit{Gold} \wedge a = \mathit{Grab}(y) \\
\quad \vee \\
\quad (\,\mathit{Holds}(\mathit{Has}(x, y), s) \wedge \\
\qquad \neg\,[\, x = \mathit{Agent} \wedge y = \mathit{Gold} \wedge a = \mathit{Drop}(y)\,]\,)
\end{array}$$

The basic axiomatisation technique of the Situation Calculus has been extended in (Scherl and Levesque 2003) by an explicit formalisation of the *knowledge* of an agent and how it evolves. This extension is based on the special *epistemic* predicate $\mathbf{K}(s', s)$, which is to be read as: in the actual situation s, the agent considers it possible to be in situation s'. This allows to form expressions about the knowledge of the agent in specific situations in analogy to the usual possible worlds semantics (Moore 1985). The following, for example, says that our gold mining robot knows that it is initially at location 4 while it considers it possible that the gold is in any of the four locations:

$$\begin{array}{l}
(\forall s')\,(\mathbf{K}(s', S_0) \supset \mathit{Holds}(\mathit{At}(\mathit{Agent}, 4), s')) \\
\wedge \\
(\forall y\colon 1 \ldots 4)\,(\exists s')\,(\mathbf{K}(s', S_0) \wedge \mathit{Holds}(\mathit{At}(\mathit{Gold}, y), s'))
\end{array} \tag{9}$$

The solution to the frame problem extends to knowledge by the following successor state axiom for the special epistemic predicate (Scherl and Levesque 2003):

$$\begin{array}{rl}
\mathbf{K}(s'', \mathit{Do}(a, s)) \;\equiv\; (\exists s')\,(\, & \mathbf{K}(s', s) \wedge \mathit{Poss}(a, s') \wedge \\
& s'' = \mathit{Do}(a, s') \wedge \\
& \mathit{SR}(a, s) = \mathit{SR}(a, s')\,)
\end{array}$$

Here, the domain-dependent function $\mathit{SR}(a, s)$ characterises the sensing information the agent gets when performing action a in situation s. For non-sensing actions, this can simply be set to an arbitrary unique constant; e.g.,

$$\begin{array}{r}
\mathit{SR}(\mathit{Go}, s) = \top \\
\mathit{SR}(\mathit{Grab}(x), s) = \top \\
\mathit{SR}(\mathit{Drop}(x), s) = \top
\end{array}$$

This brings us to the last missing piece of a complete domain axiomatisation in conjunction with an AgentSpeak program: the special action $\mathit{SenseAct}$ used in the generic AgentLP of Figure 2 and how it affects the knowledge of the agent. This includes the perception of new external triggers, formalised by the special fluent $\mathit{Goals}(e)$ in our generic AgentLP. We begin by asserting

$$\mathit{Poss}(\mathit{SenseAct}, s) \;\equiv\; \mathit{True}$$

Now, recall that our example agent is assumed to be equipped with a sensor that indicates whether gold is at the current position of the agent. This, together with learning about new goals, is suitably axiomatised as follows:

$$\begin{array}{l}
\mathit{SR}(\mathit{SenseAct}, s) = r \;\equiv \\
\quad (\exists e, x)\,(\mathit{Holds}(\mathit{Goals}(e), s) \wedge \mathit{Holds}(\mathit{At}(\mathit{Agent}, x), s) \wedge \\
\qquad\qquad [\, r = (e, \top) \wedge \mathit{Holds}(\mathit{At}(\mathit{Gold}, x), s) \\
\qquad\qquad\quad \vee \\
\qquad\qquad\quad r = (e, \bot) \wedge \neg\mathit{Holds}(\mathit{At}(\mathit{Gold}, x), s)\,]\,)
\end{array}$$

Put in words, the sensing result is formulated as a pair consisting of the list of external goals that hold in the current situation plus a constant symbol indicating whether or not there is gold at the current location.[10] Finally, we assert that the special fluent $\mathit{Goals}(e)$, which otherwise may change arbitrarily between situations, is not affected by sensing:[11]

$$\mathit{Holds}(\mathit{Goals}(e), \mathit{Do}(\mathit{SenseAct}, s)) \equiv \mathit{Holds}(\mathit{Goals}(e), s)$$

As an example, consider a scenario where gold is actually at 4 (but without the agent knowing this). Formally,

$$\mathit{Holds}(\mathit{At}(\mathit{Gold}, 4), S_0) \tag{10}$$

Furthermore, suppose that only once the agent is given an external goal: $\mathit{Goals}([+!\,\mathit{Has}(\mathit{Agent}, \mathit{Gold})])$ holds in S_0, and hence in $\mathit{Do}(\mathit{SenseAct}, S_0)$, while $\mathit{Goals}([\,])$ holds in all later situations. From this and the initial knowledge given in (9) along with the successor state axioms for, respectively, the epistemic predicate and the other fluents, we can infer the following about the effect of sensing in the initial situation:

$$\begin{aligned}(\forall s'')\,(\mathbf{K}(s'', \mathit{Do}(\mathit{SenseAct}, S_0)) \supset \\ \mathit{Holds}(\mathit{At}(\mathit{Agent}, 4), s'') \wedge \\ \mathit{Holds}(\mathit{At}(\mathit{Gold}, 4), s'') \wedge \\ \mathit{Holds}(\mathit{Goals}([+!\,\mathit{Has}(\mathit{Agent}, \mathit{Gold})]), s''))\end{aligned} \tag{11}$$

Put in words, after sensing the agent knows that gold is at the present location and that the goal is to get it. In turn, this implies (cf. (8)) that the agent knows it can now pick up the gold, then go forward to reach location 1, and finally drop the gold there.

This concludes the domain axiomatisation for the gold mining agent. The formulae in this section are completed by the standard foundational axioms of the situation calculus, which provide a formal characterisation of the tree-like structure of situations and which include a second-order induction principle on situations. The details need not concern us here; we refer the interested reader to (Reiter 2001). The example axiomatisation illustrates some of the expressive power of the Situation Calculus and how such action calculi can be used to formulate rich and elaborate background action theories.

Putting Part 1 and 2 Together

With its purely declarative interpretation $\mathit{Ax}(P)$, any AgentSpeak program P can simply be conjoined with any rich theory of actions like the one just presented. Let Ax_D be a set of action domain axioms plus the foundational axioms of the chosen calculus, then the combination of AgentSpeak program and action theory is given by the union $\mathit{Ax}(P) \cup \mathit{Ax}_D$, where the two parts are linked by the special predicates $\mathit{Knows}(f, s)$ and $\mathit{Poss}(a, s, t)$. For our example of the Situation Calculus, this is provided by the following definition, the first of which is identical to a macro used in (Scherl and Levesque 2003):

$$\mathit{Knows}(f, s) \stackrel{\text{def}}{=} (\forall s')\,(\mathbf{K}(s', s) \supset \mathit{Holds}(f, s'))$$

$$\begin{aligned}\mathit{Poss}(a, s, t) \stackrel{\text{def}}{=} {} & (\forall s')\,(\mathbf{K}(s', s) \supset \mathit{Poss}(a, s')) \wedge \\ & t = \mathit{Do}(a, s)\end{aligned}$$

[10]We remark that the axiom entails that both a unique instance of $\mathit{Goals}(e)$ and an instance of $\mathit{At}(\mathit{Agent}, x)$ holds in every situation.

[11]For all other fluents this follows from their successor state axioms and uniqueness-of-names for actions.

This completes the combined semantics, with which we have closed the gap between BDI-based agent programming and action theories. Our result opens up a whole range of potential applications. First, the semantics allows to apply logical inference to investigate properties of AgentSpeak programs, like for example what execution traces they allow under given circumstances, or whether they satisfy given formal specifications. Second, thanks to the modularity of the combined semantics, the AgentSpeak execution mechanism can be coupled with any reasoner that implements a suitable action calculus, thus allowing for much richer action theories than originally provided for AgentSpeak. Third, with the declarative reformulation of the operational semantics for AgentSpeak given by the generic AgentLP in Figure 2, it is now possible to define and analyse modifications and extensions of the basic BDI-model in a purely declarative setting.

As a simple example for using the axiomatisation to infer properties about an AgentSpeak program, recall the declarative reading of the program for the gold mining agent along with the action theory for this domain from above, including the axioms about the initial state and knowledge, (9) and (10). Let $S_1 = \mathit{Do}(\mathit{SenseAct}, S_0)$, then from (11) and the second formula in (7) we can conclude that

$$\begin{aligned}\mathit{Procedure}(&+!\,\mathit{Has}(\mathit{Agent}, Gold), \\ & [\mathit{Grab}(\mathit{Gold}), !\,\mathit{At}(\mathit{Agent}, 1), \mathit{Drop}(\mathit{Gold})], \\ & S_1, S_1)\end{aligned}$$

This forms the basis for a simple derivation using (the logical reading of) the clauses in Figure 2 by which it can be shown that the following "execution trace" is a logical consequence of $\mathit{Ax}(P) \cup \mathit{Ax}_D$ (for the sake of brevity, we have omitted all $\mathit{SenseAct}$ except for the initial one):

$$\begin{aligned}\mathit{Agent\_Speak}(S_0, \mathit{Do}(&\mathit{Drop}(\mathit{Gold}), \\ & \mathit{Do}(\mathit{Go}, \mathit{Do}(\mathit{Grab}(\mathit{Gold}), \\ & \quad \mathit{Do}(\mathit{SenseAct}, S_0)))))\end{aligned}$$

Declarative vs. Operational Semantics

As we have just shown, our new declarative semantics entails possible execution traces of an AgentSpeak program. In this section, we will formally prove, as the main technical result of our work, that this provides a correct characterisation of the standard operational semantics for AgentSpeak.

The operational semantics for AgentSpeak (Rao 1996) can be given by a system of transformation rules on the internal state of an agent, formalised as belief-desire-intention triple $(\mathcal{B}, \mathcal{D}, \mathcal{I})$ along with $\sigma \in \{\texttt{sense}, \texttt{select}, \texttt{act}\}$ to indicate the current stage in the sense-select-act cycle. A successful derivation transforms an initial internal state $(\mathcal{B}_0, \mathcal{D}_0 = \{\}, \mathcal{I}_0 = \{\}, \sigma_0 = \texttt{sense})_{S_0}$ (where S_0 stands for the empty sequence of actions) into some final state $(\mathcal{B}_n, \mathcal{D}_n = \{\}, \mathcal{I}_n = \{\}, \sigma_n = \texttt{act})_{S_n}$, where S_n is the sequence of actions performed by the agent.

Before we present the individual transformation rules and show the equivalence to the declarative semantics, we make the following technical remarks and assumptions.

1. The transformation rules for AgentSpeak are defined relative to an underlying model for the evolution of the

agent's belief base in the course of executing a program. Our combined semantics allows to use any suitable background action theory to this end. For the purpose of relating the operational to the declarative semantics, we of course assume that both share the same background theory. In order to do so, we identify each timepoint used in the underlying action theory with the actions that have been executed up to this point.[12] In view of the operational semantics, this requires the further assumption that the background theory gives a decidable account of what the agent knows and does not know at any time.

2. As a set of definite clauses, $\mathit{Ax}(P)$ is consistent for every AgentSpeak program P. We assume the same of Ax_D, which then implies consistency of $\mathit{Ax}(P) \cup \mathit{Ax}_D$ since the linking predicates, *Poss* and *Knows*, do not occur in the head of a clause in P.

3. Since our AgentLP is a definite program and because we assume that the background theory gives a complete and decidable account of the special atoms $\mathit{Knows}(f,s)$ and $\mathit{Poss}(a,s,t)$, standard SLD-resolution can be used as a sound and complete derivation mechanism for queries to the program (Drescher, Schiffel, and Thielscher 2009). We will use the notation
$$Q \implies Q'\theta$$
to denote a finite number of SLD-resolution steps from Q to $Q'\theta$. This also applies to resolving our two special atoms against the background action theory to determine some θ such that $\mathit{Ax}_D \models \mathit{Knows}(f,s)\theta$ or $\mathit{Ax}_D \models \mathit{Poss}(a,s,t)\theta$, respectively.

4. We assume that external desires are variable-free.[13]

5. We assume that an AgentSpeak program contains at least one applicable procedure (cf. Definition 4) for every triggering event that actually occurs.[14]

6. We assume that Ax_D entails
$$(\forall s,t)\,(\mathit{Poss}(\mathit{SenseAct},s,t) \equiv t = (s,\mathit{SenseAct})) \quad (12)$$
We also assume that an intention with a leading action can only be selected if the action is possible according to what the agent knows at the current stage.[15]

[12]This is obviously inherent in the time structure based on situations, but can also be defined if the action calculus chosen for the background theory uses a linear time structure. To stress this generality, we will write (α, a) to indicate the timepoint identified with the sequence of actions α followed by a; see e.g. (12) below.

[13]This does not restrict the expressiveness of AgentSpeak, because a trigger like, say, $+?p(\vec{x})$ can always be replaced by a ground trigger $+?p$ along with the procedure $p : \mathit{True} \leftarrow ?p(\vec{x})$.

[14]With this assumption we avoid, for the sake of simplicity, having to deal with deferring or ignoring desires; a triggering event without a relevant procedure can never be acted upon anyway, and deferring a triggering event in case a relevant but no applicable procedure exists can always be simulated by a recursive, i.e. cyclic, procedure with empty context.

[15]The original operational semantics ignores action preconditions.

7. Beliefs and intentions in AgentSpeak are modelled as sets while they are represented as lists in the AgentLP of Figure 2. For the sake of simplicity, we will treat these two representations as interchangeable. It is easy to prove that this is justified from the clausal definition for predicate *Member* in Figure 2.

8. The transformation rules given in the following are adaptations of the original operational semantics in view of the simpler representation of intentions as stacks of (partially executed) procedure bodies rather than of entire procedures as in (Rao 1996) (cf. Definition 3).

With these preparatory remarks we are now ready to present the transformation rules of the operational semantics and to relate these to our new declarative semantics.

The Transformation Rules

The transformation rules can be grouped together for each step in the sense-select-act cycle.

First Step There is only one derivation rule for the stage in which the agent senses:
$$(\mathcal{B},\mathcal{D},\mathcal{I},\texttt{sense})_s \;\mapsto\; (\mathcal{B}',\mathcal{D}',\mathcal{I},\texttt{select})_{s,\mathit{SenseAct}}$$
where $\mathcal{B}'$ is obtained by updating $\mathcal{B}$ according to the sensing result and where $\mathcal{D}'$ is $\mathcal{D}$ augmented by all external desires sensed by the agent.

Lemma 1.
$$(\mathcal{B},\mathcal{D},\mathcal{I},\texttt{sense})_s \;\mapsto\; (\mathcal{B}',\mathcal{D}',\mathcal{I},\texttt{select})_{s,\mathit{SenseAct}}$$
if and only if
$$\mathit{Sense}(\mathcal{D},\mathcal{I},s,t) \implies \mathit{Select}(\mathcal{D}',\mathcal{I},(s,\mathit{SenseAct}),t)$$

Proof. The second clause in Figure 2 is the only one in $\mathit{Ax}(P)$ with head *Sense*. Recall its temporal extension, (6). Assumption 6 from above implies $\mathit{Poss}(\mathit{SenseAct},s,s_2)$ iff $s_2 = (s,\mathit{SenseAct})$. Furthermore, from the assumption that Ax_D characterises precisely the evolution of the agent's knowledge, it follows that $\mathit{Ax}_D \models \mathit{Knows}(\mathit{Goals}(e),s_2)$ iff e coincides with the external events sensed by the agent. Moreover, the only two clauses with head *Append* in Figure 2 imply that $\mathit{Append}(\mathcal{D},e,f,(s,\mathit{SenseAct}),s_4)$ iff f coincides with $\mathcal{D}'$ (which is $\mathcal{D}$ augmented by e) and $s_4 = (s,\mathit{SenseAct})$. The claim then follows by (6) and $\{s_1/s, s_5/t\}$. □

Second Step For the selection of a desire, two cases are distinguished: If there is no desire, the agent proceeds by acting according to one of the current intentions; otherwise, a desire is selected together with an applicable procedure, which is added to the intentions.

1. If $\mathcal{D} = \{\}$ and $\mathcal{I} \neq \{\}$ then
$$(\mathcal{B},\mathcal{D},\mathcal{I},\texttt{select})_s \;\mapsto\; (\mathcal{B},\mathcal{D},\mathcal{I},\texttt{act})_s$$

2. If $\langle e; i\rangle \in \mathcal{D}$ and $p\theta\eta$ is the body of an applicable (to e) procedure instance in the AgentSpeak program P, then
$$\begin{aligned}&(\mathcal{B},\mathcal{D},\mathcal{I},\texttt{select})_s\\ &\quad\mapsto\; (\mathcal{B},\mathcal{D}\setminus\{\langle e;i\rangle\},\mathcal{I}\cup\{[p;i]\}\theta\eta,\texttt{act})_s\end{aligned}$$

Lemma 2.

$$(\mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{select})_s \;\mapsto\; (\mathcal{B}, \mathcal{D}', \mathcal{I}', \texttt{act})_s$$

if and only if

$$\mathit{Select}(\mathcal{D}, \mathcal{I}, s, t) \;\Longrightarrow\; \mathit{Act}(\mathcal{D}', \mathcal{I}', s, t)$$

Proof. The second clause with head *Select* in Figure 2 is obviously equivalent to the corresponding transformation rule.

By the third clause, an element $\langle x; j\rangle \in \mathcal{D}$ is selected such that $\mathcal{D}'$ is $\mathcal{D} \setminus \{\langle x; j\rangle\}$ (according to the clauses for *Member*) and such that an applicable procedure instance with body $p\theta\eta$ exists (according to Definition 6 and 4). Furthermore, $\mathcal{I}'$ is the result of adding $[p|j]$ to $\mathcal{I}$ and applying substitution $\theta\eta$. □

Third Step For the last step in the sense-select-act cycle, if an empty intention is selected, the intention has been fully achieved, and in case the first element of the selected intention is the empty sequence, the next element in the intention becomes active. Otherwise, the first step of the first element in the selected intention is executed. If it is an action, then the actual execution of the action leads to an updated belief base according to the effects of that action. If the first element is an achievement goal $!f$, then a new desire is obtained with this goal as triggering event. Finally, if the first element is a test goal $?f$, then this must be established from the current belief base.

1. For $[\,] \in \mathcal{I}$,
$$(\mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{act})_s \;\mapsto\; (\mathcal{B}, \mathcal{D}, \mathcal{I} \setminus \{[\,]\}, \texttt{sense})_s$$
2. For $[\mathit{True}; P_2; \ldots; P_k] \in \mathcal{I}$,
$$(\mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{act})_s \;\mapsto\; (\mathcal{B}, \mathcal{D}, \mathcal{I}', \texttt{sense})_s$$
where $\mathcal{I}' = \mathcal{I} \setminus \{[\mathit{True}; P_2; \ldots; P_k]\} \cup \{[P_2; \ldots; P_k]\}$.
3. For $[a, P_1; \ldots; P_k] \in \mathcal{I}$ with a an action,
$$(\mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{act})_s \;\mapsto\; (\mathcal{B}', \mathcal{D}, \mathcal{I}', \texttt{sense})_{s,a}$$
where $\mathcal{I}' = \mathcal{I} \setminus \{[a, P_1; \ldots; P_k]\} \cup \{[P_1; \ldots; P_k]$ and $\mathcal{B}'$ is the result of updating $\mathcal{B}$ by the effects of action a.
4. For $[!f, P_1; \ldots; P_k] \in \mathcal{I}$,
$$(\mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{act})_s \;\mapsto\; (\mathcal{B}, \mathcal{D}', \mathcal{I}', \texttt{sense})_s$$
where $\mathcal{D}' = \mathcal{D} \cup \{\langle +!f; [P_1; \ldots; P_k]\rangle\}$ and $\mathcal{I}' = \mathcal{I} \setminus \{[!f, P_1; \ldots; P_k]\}$.
5. For $[?f, P_1; \ldots; P_k] \in \mathcal{I}$ and θ a substitution such that $\mathcal{B}$ entails $f\theta$,
$$(\mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{act})_s \;\mapsto\; (\mathcal{B}, \mathcal{D}, \mathcal{I}', \texttt{sense})_s$$
where $\mathcal{I}' = \mathcal{I} \setminus \{[?f, P_1; \ldots; P_k]\} \cup \{[P_1; \ldots; P_k]\theta\}$.

Lemma 3.

$$(\mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{act})_s \mapsto (\mathcal{B}', \mathcal{D}', \mathcal{I}', \texttt{select})_{s'}$$

if and only if

$$\mathit{Act}(\mathcal{D}, \mathcal{I}, s, t) \;\Longrightarrow\; \mathit{Sense}(\mathcal{D}', \mathcal{I}', s', t)$$

Proof. It is straightforward to verify that the five AgentLP clauses for *Act* in Figure 2 are equivalent to the respective transformation rules. □

Putting everything together leads to our main result.

Theorem 4. Under the assumptions made above,

$$(\mathcal{B}_0, \{\}, \{\}, \texttt{sense})_{S_0} \;\mapsto\; \ldots \;\mapsto\; (\mathcal{B}_n, \{\}, \{\}, \texttt{act})_{S_n}$$

if and only if

$$\mathit{Ax}(P) \cup \mathit{Ax}_D \;\models\; \mathit{Agent\_Speak}(S_0, S_n)$$

Proof. According to the first clause in Figure 2,

$$\mathit{Agent\_Speak}(S_0, s) \;\Longrightarrow\; \mathit{Sense}([\,], [\,], S_0, s)$$

Moreover, the only fact in the program $\mathit{Ax}(P)$ entails that an SLD-derivation is successful if and only if it ends in

$$\mathit{Select}([\,], [\,], s, t) \;\Longrightarrow\; \Box$$

with $\{t/s\}$. The claim follows by induction over the length of a derivation using Lemma 1 to 3, and by soundness and completeness of SLD-resolution for definite programs. □

Conclusion

Since AgentSpeak uses many elements from logic programming, Agent Logic Programs turned out to provide the ideal link between knowledge representation formalisms for actions on the one hand, and a practically oriented agent programming language based on the BDI-model on the other hand. This can be seen, for example, by contrasting the AgentLP in Figure 2 to the existing axiomatisation of AgentSpeak using the Z specification language (d'Inverno and Luck 1998), which is significantly more involved. Our work also differs from other existing approaches to the formal semantics of AgentSpeak, for instance (Moreira, Vieira, and Bordini 2003; Bordini and Moreira 2004), in that the formulation as an AgentLP allows to combine AgentSpeak-based strategies with arbitrary theories for reasoning about actions. An advantage of this is that agents can use much richer state and action representations compared to the usual definition of a belief base and its update in AgentSpeak. Moreover, existing language extensions (e.g., (Moreira, Vieira, and Bordini 2003)) as well as new features can be realised in the underlying background theory—by specifying how beliefs and goals are affected by, say, speech-acts—in combination with the fixed execution strategy of Figure 2. Conversely, execution strategies that are more sophisticated can be defined without affecting the reasoning module of an agent.

Existing approaches to the integration of beliefs, desires, and intentions into action theories, such as (Lespérance, Levesque, and Reiter 1999; Parra, Nayak, and Demolombe 2005), have a focus different from our work. There, beliefs, desires, and intentions are rigorously formalised as state properties with the help of modal operators and by formalising the handling of desires and intentions as actions of the agent, too. This results in comparatively complex axiomatisations, lacking the simple elegance of practical programming languages such as AgentSpeak. In contrast, our semantics keeps the conceptual separation between desires and intentions on the one hand, which are used to program the agent's behaviour, and the background action theory on the other hand. Also related is the work of (Hindriks, Lespérance, and Levesque 2000), where it is shown that the GOLOG variant of (Giacomo, Lespérance, and Levesque 2000) can be embedded in the agent implementation language 3APL (Hindriks et al. 1999). Abstracting away from the specifics of the languages, this can be considered the reversal of our investigation, but in order to exploit the strengths of the two lines of research, it seems much more interesting to use a practical BDI-language for behaviours in combination with a rich theory for action knowledge.

Among the advantages of a purely declarative semantics for AgentSpeak is to provide new ways of proving the correctness of an agent program wrt. stated requirements and relative to a given specification of the underlying dynamic environment. This is similar to the use of model checking for AgentSpeak (Bordini et al. 2004) but allows to apply standard techniques like the induction principle of the Situation Calculus (Reiter 2001), which is not restricted to finite state spaces. In addition, our Agent Logic Program can be readily used as the basis for an implementation in which—thanks to the separation of behaviour and reasoning—BDI-based programs for the intended agent behaviour can be combined with any established system of reasoning about actions, such as the basis for implementations of GOLOG (Reiter 2001) or FLUX (Thielscher 2005a). It is worth noting that for this purpose the background action theory need not include a formal account of knowledge; instead it suffices to interpret the special predicates *Poss* and *Knows* as derivability wrt. the current world model of the agent. In this way our result paves the way for a practical integration of the BDI-based programming style with knowledge representation methods for agents, and it can be considered a model case for combining the advantages of these two hitherto mostly independent lines of research.

References

Bordini, R., and Moreira, Á. 2004. Proving BDI properties of agent-oriented programming languages. *Annals of Mathematics and Artificial Intelligence* 42(1–3):197–226.

Bordini, R.; Fisher, M.; Visser, W.; and Wooldridge, M. 2004. Model checking rational agents. *IEEE Intelligent Systems* 19(5):46–52.

Bordini, R.; Hübner, J.; and Wooldridge, M. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason*.

d'Inverno, M., and Luck, M. 1998. Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation* 8(3):233–260.

Drescher, C.; Schiffel, S.; and Thielscher, M. 2009. A declarative agent programming language based on action theories. In *Proc. of the Int.'l Conf. on Frontiers of Combining Systems*, vol. 5749 of *LNCS*, 230–245. Springer.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Georgeff, M., and Lansky, A. 1987. Reactive reasoning and planning. In *Proc. of AAAI*, 677–682.

Giacomo, G. D.; Lespérance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121:109–169.

Hindriks, K.; Boer, F. D.; der Hoek, W. V.; and Meyer, J.-J. 1999. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems* 2(4):357–401.

Hindriks, K.; Lespérance, Y.; and Levesque, H. 2000. An embedding of ConGolog in 3APL. In *Proc. of ECAI*, 558–562.

Lespérance, Y.; Levesque, H.; and Reiter, R. 1999. A situation calculus approach to modeling and programming agents. In Rao, A., and Wooldridge, M., eds., *Foundations and Theories of Rational Agents*.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31(1–3):59–83.

Lloyd, J. 1987. *Foundations of Logic Programming*.

Mascardi, V.; Demergasso, D.; and Ancona, D. 2005. Languages for programming BDI-style agents: an overview. *Proc. of the Workshop From Objects to Agents*, 9–15. Camerino, Italy: Pitagora Editrice Bologna.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence* 4:463–502.

Moore, R. 1985. A formal theory of knowledge and action. In *Formal Theories of the Commonsense World*. 319–358.

Moreira, Á.; Vieira, R.; and Bordini, R. 2003. Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. *Proc. of the Int.'l Workshop on Declarative Agent Languages and Technologies*, vol. 2990 of *LNCS*, 135–154. Springer.

Parra, P. P.; Nayak, A.; and Demolombe, R. 2005. Theories of intentions in the framework of situation calculus. *Proc. of the Int.'l Workshop on Declarative Agent Languages and Technologies*, vol. 3476 of *LNAI*, 19–34. Springer.

Rao, A. 1996. AgentSpeak(L): BDI agents speak out in a logical language. In *Agents Breaking Away*, vol. 1038 of *LNAI*, 42–55. Springer.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation*, 359–380.

Reiter, R. 2001. *Knowledge in Action*. MIT Press.

Scherl, R., and Levesque, H. 2003. Knowledge, action, and the frame problem. *Artificial Intelligence* 144(1):1–39.

Thielscher, M. 2005a. FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming* 5(4–5):533–565.

Thielscher, M. 2005b. *Reasoning Robots: The Art and Science of Programming Robotic Agents*, vol. 33 of *Applied Logic Series*. Springer.