Reasoning about agent execution strategies

(Short Paper)

Natasha Alechina
School of Computer Science
University of Nottingham
Nottingham NG8 1BB, UK
nza@cs.nott.ac.uk

Mehdi Dastani
Department of Information and Computing Sciences
Universiteit Utrecht
3584CH Utrecht, The Netherlands
mehdi@cs.uu.nl

Brian Logan
School of Computer Science
University of Nottingham
Nottingham NG8 1BB, UK
bsl@cs.nott.ac.uk

John-Jules Ch. Meyer
Department of Information and Computing Sciences
Universiteit Utrecht
3584CH Utrecht, The Netherlands
jj@cs.uu.nl

ABSTRACT

We present a logic for reasoning about properties of agent programs under different agent execution strategies. Using the agent programming language SimpleAPL as an example, we show how safety and liveness properties can be expressed by translating agent programs into expressions of the logic. We give sound and complete axiomatizations of two different program execution strategies for SimpleAPL programs, and, for each of those strategies, prove a correspondence between the operational semantics of SimpleAPL and the models of the corresponding logic.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]; F.3 [**Logics and Meanings of Programs**]

Keywords

Formalisms and logics

1. INTRODUCTION

A key issue in the design and development of BDI agents is the choice of execution strategy which governs the execution of the agent's program. The execution strategy determines which goals the agent will pursue and when, and which plans are adopted and how they are executed. It therefore plays a key role in determining the behaviour of the agent and hence whether the agent will achieve its goals. For example, even if the agent's program is capable in principle of achieving a particular goal in a given situation, a particular execution strategy may mean that the relevant actions never get executed, or are executed in such a way as not to achieve the goal.

In current BDI-based agent programming languages [5], the basic execution strategy is either an integral part of the semantics, e.g., Jason [4], or is defined by the interpreter that comes with the corresponding execution platform, e.g., 3APL [9], 2APL [7] and Jadex [13]. However, most implementations provide customization mechanisms which allow a developer to influence key aspects of program execution, e.g., the number of goals that can be pursued at the same time, or even to replace a default platform-specific execution strategy with one of their own devising. Such features can simplify program design by allowing developers to tailor the execution of their programs to the demands of a particular problem.

When verifying properties of agent programs it is therefore important that the effects of the execution strategy are taken into account. Of course, one could ignore the impact of any particular execution strategy and consider only those properties of an agent program that are valid under *all* strategies. However, we believe that most interesting properties of agent programs depend on the execution strategy adopted and programs should therefore be verified in the context of a particular strategy. To determine the correctness of agent programs, we therefore need to be able to reason about execution strategies. While there has been considerable research on reasoning about and verification of BDI agents, e.g., [11, 3, 6, 12, 14], to the best of our knowledge the question of execution strategies has not been investigated before.

In this paper we present a logic for reasoning about execution strategies. We consider execution strategies in the context of a simple APL-like [8, 5] agent programming language, SimpleAPL. We briefly describe the syntax and operational semantics of SimpleAPL and define two alternative execution strategies for SimpleAPL programs. We then introduce the syntax and semantics of a logic to reason about safety and liveness properties of SimpleAPL programs under these execution strategies. We show how to translate agent programs written in SimpleAPL into expressions of the logic. We provide sound and complete axiomatizations of the logic for both execution strategies, and prove a correspondence between the operational semantics of SimpleAPL and the models of the logic.

The main contribution of the paper is to show how execution strategies of BDI-based agent programming languages can be axiomatized. Although we focus on a particular (simple) agent programming language, our approach is general enough to accommodate any execution strategy that can formulated in terms of distinct

Cite as: Reasoning about agent execution strategies (Short Paper), Natasha Alechina, Mehdi Dastani, Brian Logan and John-Jules Ch. Meyer, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 1455-1458.

phases of execution and the kinds of operations that can be performed in each phase.

2. SimpleAPL

SimpleAPL is a fragment of the agent-oriented programming language 3APL [8, 5]. SimpleAPL contains the core features of 3APL and allows the implementation of agents with beliefs, goals, actions, plans, and planning rules. The main features of 3APL not present in SimpleAPL are a first order language for beliefs and goals, and rules for dropping goals and for revising plans. We have omitted these features in order to simplify the presentation; they do not present a significant technical challenge for our approach.

Below we briefly sketch the main features of SimpleAPL. A detailed presentation of the syntax and operational semantics of the language can be found in [2].

The state or configuration of a SimpleAPL agent is defined as $\langle \sigma, \gamma, \Pi \rangle$, where σ is a set of literals representing the agent's beliefs, γ is a set of literals representing the agent's goals, and Π is a set of current active plans of the agent. The *beliefs* of an agent represent its information about its environment, while its *goals* represent situations the agent wants to realize (not necessary all at once). *Basic actions* specify the capabilities an agent can use to achieve its goals. There are three types of basic actions: those that update the agent's beliefs and those which test its beliefs and goals. A belief update action is specified in terms of its pre- and postconditions, and can be executed if its pre-condition is derivable from the agent's current beliefs. Executing the action adds its postcondition to the agent's beliefs. We assume that the agent's beliefs are always correct and that executing an action always achieves the action's postcondition, so we can specify preconditions and results of actions in terms of the agent's beliefs. Belief and goal queries are boolean combinations of belief or goal literals, respectively.

In order to achieve its goals, an agent adopts *plans*. A plan consists of basic actions composed by sequence ;, conditional choice `if...then...else` and conditional iteration `while...do` operators. *Planning goal rules* are used by the agent to select a plan based on its current goals and beliefs. A planning goal rule consists of three parts: an (optional) goal query κ, a belief query β, and the body of the rule, a plan π. A rule $\kappa \leftarrow \beta \mid \pi$ can be executed if κ follows from the agent's goals, β follows from the agent's beliefs, and π in its full or partially executed form is not in the current plan base Π. The result of executing the rule is that π is added to Π. An agent's *program* is defined as a set of planning goal rules.

In this paper, we consider two execution strategies for SimpleAPL programs: a *fully-interleaved* execution strategy (which we denote **(i)**) which is defined as: "either apply a planning goal rule, or execute the first step in any of the current plans; repeat"; and a *non-interleaved* (**ni**) execution strategy which executes a single plan to completion before choosing another plan, i.e., "when in a configuration with no plan, choose a planning goal rule nondeterministically, apply it, execute the resulting plan; repeat". The non-interleaved strategy prohibits execution paths in which the application of planning goal rules is interleaved with the execution of plans. Note that, under both execution strategies, plan execution halts if a basic action is not executable but the plan remains in the plan base. We assume that the agent starts in a configuration with an empty plan base.

3. LOGIC

The language of our logic is essentially the language of PDL extended with belief and goal operators. Both the language and the models are defined relative to an agent program (set of planning goal rules) Λ with a given set of plans $\Pi(\Lambda)$ and pre- and post conditions for belief updates $C(\Lambda)$.

Let $\Lambda = \{r_1, \ldots, r_n\}$ be the set of agent's planning goal rules, each of which is of the form $r_i = \kappa_i \leftarrow \beta_i \mid \pi_i$. Let $\Pi(\Lambda) = \{\pi_1, \ldots \pi_n\}$ be the set of agent's plans occurring in the rules, and $Ac(\Lambda)$ the finite set of belief update actions occurring in those plans. We will denote the elements of $Ac(\Lambda)$ by $\alpha, \alpha', \ldots$. Finally, let P_b be the set of positive belief literals occurring in Λ or in the pre- and postconditions of belief updates in Λ, and P_g be the set of positive goal literals occurring in Λ. For each belief update α, we have a set of pre- and postcondition pairs $C(\alpha) = \{(\mathtt{prec}_1(\alpha), \mathtt{post}_1(\alpha)), \ldots, (\mathtt{prec}_k(\alpha), \mathtt{post}_k(\alpha))\}$. Each $\mathtt{prec}_j(\alpha)$ is a set of propositional variables from P_b or their negations. We assume that every such set is finite and that any two preconditions for an action α, $\mathtt{prec}_j(\alpha)$ and $\mathtt{prec}_i(\alpha)$, are mutually exclusive (both sets of propositional variables cannot be satisfied simultaneously). For each set $\mathtt{prec}_j(\alpha)$ there is a unique corresponding postcondition $\mathtt{post}_j(\alpha)$, which is also a finite set of literals. We denote the set of all pre- and postconditions for all belief updates in Λ by $C(\Lambda)$.

The alphabet of our logic consists of (1) a set of propositional variables $P = P_b \cup P_g \cup P_c$, where P_c is a set of boolean flags $start_i$ (for every i) for plan π_i has started; (2) a set of atomic actions $Ac(\Lambda)$ (3) for every rule $r_i \in \Lambda$, an atomic action δ_{ri} (for apply r_i); (4) for every i, an atomic action e_i executed at the end of each plan π_i indicating that the end of the plan has been reached and which re-enables firing planning goal rules.

PDL program expressions ρ are built out of atomic actions by using sequential composition ';', test on formulas '?', union '$\cup$', finite iteration '$*$' and interleaving $\|$ [1]. Note that every formula with the interleaving operator can be rewritten without the interleaving operator, but the resulting formula might be doubly exponentially larger [1]. The formulas on which we can test are any formulas of the language L defined below. The language L for talking about the agent's beliefs, goals and plans is the language of PDL extended with a belief operator B and a goal operator G. A formula of L is defined as follows: if $p \in P$, then $B(\neg)p$ and $G(\neg)p$ are formulas; if $p \in P_c$, then p is a formula; if ρ is a program expression and ϕ a formula, then $\langle \rho \rangle \phi$ is a formula; and L is closed under the usual boolean connectives. We define $[\rho]\phi$ as $\neg\langle\rho\rangle\neg\phi$ and $\langle[\rho]\rangle\phi$ as $\langle\rho\rangle\phi \wedge [\rho]\phi$.

3.1 Verifying agent programs in PDL

We distinguish two types of properties of agent programs: safety properties and liveness properties. Let $\phi \in L$ denote the initial beliefs and goals of an agent and $\psi \in L$ denote states in which certain beliefs and goals hold (i.e., ϕ, ψ are formulas of L containing only $B(\neg)p$ and $G(\neg)q$ atoms). The general form of safety and liveness properties is then: $\phi \rightarrow [\, \xi(\Lambda) \,]\psi$ and $\phi \rightarrow \langle \xi(\Lambda) \rangle \, \psi$, respectively (where $\xi(\Lambda)$ describes the execution of the agent's program Λ).

The beliefs, goals and plans of agent programs can be translated into PDL expressions as follows. Belief formulas of SimpleAPL are mapped to formulas of L by a function f_b. For $p \in P$, $f_b((\neg)p) = B(\neg)p$; $f_b(\phi \texttt{ and } \psi) = f_b(\phi) \wedge f_b(\psi)$ and $f_b(\phi \texttt{ or } \psi) = f_b(\phi) \vee f_b(\psi)$.

Translation of goal formulas is analogous to beliefs, with ϕ, ψ replaced by goal query expressions, B replaced by G and f_b replaced by f_g.

Translation of plan expressions: let α be a belief update action, ϕ and ψ be belief and goal query expressions, and π, π_1, π_2 be plan expressions of SimpleAPL:
$f_p(\alpha) = \alpha$
$f_p(\phi?) = f_b(\phi)?$

$f_p(\psi!) = f_g(\psi)?$
$f_p(\pi_1; \pi_2) = f_p(\pi_1); f_p(\pi_2)$
$f_p(\texttt{if}\ \phi\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2) = (f_b(\phi)?; f_p(\pi_1)) \cup (\neg f_b(\phi)?; f_p(\pi_2))$
$f_p(\texttt{while}\ \phi\ \texttt{do}\ \pi) = (f_b(\phi)?; f_p(\pi))^*; \neg f_b(\phi)?.$

Using this translation, we translate the agent's program Λ as follows:

$$\xi(\Lambda) = (\bigcup_{\Lambda' \subseteq \Lambda, \Lambda' \neq \emptyset} \|_{r_i \in \Lambda'} (\delta_{ri}; f_p(\pi_i); e_i))^*$$

Our intention is to verify properties of the agent in PDL by saying 'in all states (or in some state) reachable by a path described by $\xi(\Lambda)$, property ϕ holds'.

In order to do this, we must ensure that there is a correspondence between paths in the operational semantics plus an execution strategy, and paths in the PDL models satisfying the axioms for this strategy: if a path exists in the operational semantics, then there is a corresponding path in the PDL model. The converse is not true; for example in the PDL model from any state there is a transition by a belief update action, and in the operational semantics this only holds if the belief update is the first action of some plan which is in the plan base in that state. However, we can show that if a there is a path in the PDL model which is described by $\xi(\Lambda)$, then there is a corresponding path in the operational semantics between two configurations with empty plan bases.

3.2 General conditions on models

A model $M = (W, \{R_\alpha : \alpha \in Ac\}, R_{\delta_{ri}}, R_{e_i}, V)$ where

- W is a non-empty set of states;
- $V = (V_b, V_g, V_c)$ is the evaluation function consisting of belief and goal valuation functions V_b and V_g and control valuation function V_c such that for every $s \in W$, $V_b(s) = \{(-)p_1, \ldots, (-)p_m : p_i \in P_b\}$ is a set of agent's beliefs in s, $V_g(s) = \{(-)u_1, \ldots, (-)u_n : u_i \in P_g\}$ is a set of agent's goals in s, and $V_c(s) \subseteq P_c$ is a set of control variables true in s.
- R_α, $R_{\delta_{ri}}$ and R_{e_i} are binary relations on W such that R_α satisfies pre- and postconditions for α and $R_{\delta_{ri}}$ satisfies preconditions for firing planning goal rules and only changes the values of control variables.

The conditions on R_α, $R_{\delta_{ri}}$ and R_{e_i} depend on the execution strategy and are defined below.

Given the relations corresponding to basic actions in M, we can define sets of paths in the model corresponding to any PDL program expression ρ in M. A set of paths $\tau(\rho) \subseteq (W \times W)^*$ is defined inductively:

- $\tau(\alpha) = \{(s, s') : R_\alpha(s, s')\}$
- $\tau(\phi?) = \{(s, s) : M, s \models \phi\}$
- $\tau(\rho_1 \cup \rho_2) = \{z : z \in \tau(\rho_1) \cup \tau(\rho_2)\}$
- $\tau(\rho_1; \rho_2) = \{z_1 \circ z_2 : z_1 \in \tau(\rho_1),\ z_2 \in \tau(\rho_2)\}$, where $\circ$ is concatenation of paths, such that $z_1 \circ z_2$ is only defined if z_1 ends in the state where z_2 starts
- $\tau(\rho^*)$ is the set of all paths consisting of zero or finitely many concatenations of paths in $\tau(\rho)$
- $\tau(\rho_1 \parallel \rho_2)$ is the set of all paths obtained by interleaving atomic actions and tests from $\tau(\rho_1)$ and $\tau(\rho_2)$.

The relation $\models$ of a formula being true in a state of a model is defined inductively as follows:

- $M, s \models B(-)p$ iff $(-)p \in V_b(s)$, where $p \in P_b$
- $M, s \models G(-)p$ iff $(-)p \in V_g(s)$, where $p \in P_g$
- $M, s \models p$ iff $p \in V_c(s)$, where $p \in P_c$
- $M, s \models \neg\phi$ iff $M, s \not\models \phi$
- $M, s \models \phi \wedge \psi$ iff $M, s \models \phi$ and $M, s \models \psi$
- $M, s \models \langle\rho\rangle\phi$ iff there is a path in $\tau(\rho)$ starting in s which ends in a state s' such that $M, s' \models \phi$.

Models for all execution strategies satisfy the following condition:

Beliefs and goals $V_b(s) \cap V_g(s) = \emptyset$ and $V_b(s)$ is consistent, i.e., for no $p \in P_b$ both p and $\neg p \in V_b(s)$.

In addition, different strategies require different conditions on applicability of actions and rules.

3.3 Interleaved execution strategy

A model for Λ in conjunction with the interleaved execution strategy in addition conforms to the following constraints.

We use the notation $\{\phi_1, \ldots, \phi_n\}R_u\{l_1, \ldots, l_m\}$ as a shorthand for saying that in a state s satisfying ϕ_i there is always a transition by R_u to some state t, and all such states t reachable from s by R_u satisfy belief or control literals $\{l_1, \ldots, l_m\}$. If a belief or control literal l is not mentioned in the condition on t, then l is true in t iff it is true in s. This explicitly specifies $V_b(t)$ and $V_c(t)$. For the goal literals, $V_g(t) = V_g(s) \setminus S$ where $S = \{l : l \in V_b(t) \setminus V_b(s)\}$ is the set of belief literals made true as the result of transition. If a state s does not satisfy any of the explicitly stated preconditions for the existence of an R_u transition, then there is no R_u transition out of s.

Rules-i $\{\neg start_i, \kappa_i \in V_g(s), \beta_i \in V_b(s)\}R_{\delta_{ri}}\{start_i\}$

Actions $\{\texttt{prec}_\texttt{j}(\alpha) \subseteq V_b(s)\}R_\alpha\{\texttt{post}_\texttt{j}(\alpha)\}$.

End $\{\}e_i\{\neg start_i\}$

Let the class of transition systems defined above be denoted $\mathbf{M}(\mathbf{\Lambda}, \mathbf{i})$.

Note that for every pre- and postcondition pair $(\texttt{prec}_i, \texttt{post}_i)$ we can describe states satisfying $\texttt{prec}_i$ and states satisfying $\texttt{post}_i$ by formulas of L using the translation f_b defined above.

THEOREM 1. *The following axiom system* $\mathbf{Ax}(\mathbf{\Lambda}, \mathbf{i})$ *is sound and (weakly) complete for the class of models* $\mathbf{M}(\mathbf{\Lambda}, \mathbf{i})$.

CL *classical propositional logic*

PDL *axioms of PDL (see, e.g., [10]) excluding interleaving since it is redundant)*

A1 $\neg(Bp \wedge B{-}p)$

A2 $B(-)p \rightarrow \neg G(-)p$

R-i-1 $\neg start_i \wedge G\kappa_i \wedge B\beta_i \wedge \phi \rightarrow \langle[\delta_{ri}]\rangle(start_i \wedge \phi)$, *where* ϕ *does not contain* $start_i$

R-i-2 $start_i \vee \neg(G\kappa_i \wedge B\beta_i) \rightarrow [\delta_{ri}]\bot$

Ac $f_b(\texttt{prec}_\texttt{j}(\alpha)) \wedge \phi \rightarrow \langle[\alpha]\rangle(f_b(\texttt{post}_\texttt{j}(\alpha)) \wedge \phi)$, *where* ϕ *does not contain variables from* $\texttt{post}_\texttt{j}(\alpha)$

E $\phi \rightarrow \langle[e_i]\rangle(\phi \wedge \neg start_i)$ *for any formula* ϕ *not containing* $start_i$.

We can prove correspondence between paths in the transition system $S(\Lambda)$ generated by the operational semantics with the interleaved executions strategy, and paths described by $\xi(\Lambda)$ in $\mathbf{M}(\mathbf{\Lambda}, \mathbf{i})$.

THEOREM 2. *Let Λ be the program of an agent using the interleaving execution strategy. Let $S(\Lambda)$ be the transition system generated by the operational semantics for this agent. Let $M \in \mathbf{M}(\mathbf{\Lambda}, \mathbf{i})$ be a model generated by the initial state of S.*

Then there exists a path in S between two configurations with empty plan bases $\langle \sigma, \gamma, \{\}\rangle$ and $\langle \sigma', \gamma', \{\}\rangle$ iff there is a path in M between any two states s and s' such that $V_b(s) = \sigma$ and $V_g(s) = \gamma$, $V_b(s') = \sigma'$ and $V_g(s') = \gamma'$ with the labels of the path spelling a word w which is in the language of $\xi(\Lambda)$.

3.4 Non-interleaved execution strategy

The models for non-interleaved execution strategy satisfy **Actions** and **End** as before, and the following condition:

Rules-ni $\{\forall j \neg start_j, \kappa_i \in V_g(s), \beta_i \in V_b(s)\} R_{\delta_{r\,i}} \{start_i\}$

Let the class of transition systems defined above be denoted $\mathbf{M}(\mathbf{\Lambda}, \mathbf{ni})$.

THEOREM 3. *The following axiom system $\mathbf{Ax}(\mathbf{\Lambda}, \mathbf{ni})$ is sound and (weakly) complete for the class of models $\mathbf{M}(\mathbf{\Lambda}, \mathbf{ni})$:*

CL, PDL, A1, A2, Ac, E *as before;*

R-ni-1 $\bigwedge_j \neg start_j \wedge G\kappa_i \wedge B\beta_i \wedge \phi \rightarrow \langle[\delta_{r\,i}]\rangle(start_i \wedge \phi)$*, where ϕ does not contain $start_i$*

R-ni-2 $\bigvee_j start_j \vee \neg(G\kappa_i \wedge B\beta_i) \rightarrow [\delta_{r\,i}]\bot$

Similarly to the case of the interleaved execution strategy, we can prove correspondence between paths in the transition system $S(\Lambda)$ generated by the operational semantics with the non-interleaved executions strategy, and paths described by $\xi(\Lambda)$ in $\mathbf{M}(\mathbf{\Lambda}, \mathbf{ni})$.

Note that on models corresponding to the non-interleaved execution strategy, there are no paths described by $\xi(\Lambda)$ which contain an application of a rule (a step with a $\delta_{r\,i}$ label) followed by steps of the corresponding plan $(f_p(\pi_i); e_i)$ interleaved with an application of another rule $\delta_{r\,j}$, since after the $\delta_{r\,i}$ transition, $start_i$ is set to true, so the preconditions for $\delta_{r\,j}$ are false until $(f_p(\pi_i); e_i)$ is executed and $start_i$ is set to false again. Hence on $\mathbf{M}(\mathbf{\Lambda}, \mathbf{ni})$ models, $\xi(\Lambda)$ is equivalent to $(\cup_i(\delta_{r\,i}; f_p(\pi_i); e_i))^*$.

4. CONCLUSION

In this paper, we analysed the implications of an agent's execution strategy in determining the behavior of BDI-based agent programs. In order to illustrate the problem, we presented a simple agent programming language, SimpleAPL, and explored two of its possible execution strategies. We proposed a logic to reason about these execution strategies and explained how it can be used to verify the correctness of agent programs. Although we investigated only a small number of execution strategies, our approach is general enough to accommodate any execution strategy that can be formulated in terms of distinct phases of execution and the kinds of operations that can be performed in each phase.

In future work we plan to investigate other execution strategies. For example, by adding more control variables we can formalise strategies which alternate between executing a number of steps from each current plan. It would also be interesting to investigate strategies which prioritize particular goals and the plans that achieve them. Another direction for future work is extending the programming language, e.g., to introduce variables in the language of beliefs, goals, plans and planning goal rules, and to extend the setting to include additional phases in the agent's cycle, such as events or sensing, and actions performed in an external environment.

Acknowledgements

Natasha Alechina and Brian Logan were supported by EPSRC grant no. EP/E031226.

5. REFERENCES

[1] K. R. Abrahamson. *Decidability and expressiveness of logics of processes*. PhD thesis, Department of Computer Science, University of Washington, 1980.

[2] N. Alechina, M. Dastani, B. Logan, and J.-J. C. Meyer. A logic of agent programs. In *Proc. AAAI 2007*, pages 795–800. AAAI Press, 2007.

[3] M. Benerecetti, F. Giunchiglia, and L. Serafini. Model checking multiagent systems. *J. Log. Comput.*, 8(3):401–423, 1998.

[4] R. Bordini, J. Hübner, and R. Vieira. *Multi-Agent Programming - Languages, Platforms and Applications*, chapter Jason and the Golden Fleece of agent-oriented programming. Springer, 2005.

[5] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.

[6] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[7] M. Dastani and J. Meyer. A Practical Agent Programming Language. In *Proc. of PROMAS*, 2007.

[8] M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. C. Meyer. A programming language for cognitive agents: Goal directed 3APL. In *Proc. ProMAS 2003*, volume 3067 of *LNCS*, pages 111–130. Springer, 2004.

[9] M. Dastani, M. B. van Riemsdijk, and J.-J. C. Meyer. *Multi-Agent Programming - Languages, Platforms and Applications*, chapter Programming multi-agent systems in 3APL. Springer, 2005.

[10] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[11] K. Hindriks and J.-J. C. Meyer. Agent logics as program logics: Grounding KARO. In *Proc. KI 2006*, volume 4314 of *LNAI*. Springer, 2007.

[12] A. Lomuscio and F. Raimondi. Mcmas: A model checker for multi-agent systems. In *Proc. TACAS 2006*, pages 450–454, 2006.

[13] A. Pokahr, L. Braubach, and W. Lamersdorf. *Multi-Agent Programming - Languages, Platforms and Applications*, chapter Jadex: A BDI Reasoning Engine. Springer, 2005.

[14] S. Shapiro, Y. Lespérance, and H. J. Levesque. The cognitive agents specification language and verification environment for multiagent systems. In *Proc. AAMAS 2002*, pages 19–26. ACM Press, 2002.