Multi-Agent Programming

Brian Logan

School of Computer Science
University of Nottingham

Midlands Graduate School
8th – 12th April 2013

Course Overview

Lecture 1:	*Programming agents* BDI model; PRS and other BDI languages
Lecture 2:	*Programming multi-agent systems* Coordination in MAS; agent communication languages & protocols; programming with obligations and prohibitions
Lecture 3:	*Logics for MAS* LTL, CTL; Rao and Georgeff's BDI logics; Coalition Logic, ATL
Lecture 4:	*Verification of MAS* A tractable APL and BDI logic: SimpleAPL and PDL-APL

Lecture 4: Verification of MAS

Outline of this lecture

- model checking BDI logics
- model checking based on interpreted systems
- workarounds: computational grounding for classical BDI logics
- doing it right: syntactic belief ascription
- example: PDL-APL

Model checking

Model-checking

- a model-checker is a tool for automatically checking whether a state transition system (model) satisfies a formula ϕ
- basic idea:
    - encode the system to be verified as an input to the model-checker
    - this encoding uses some way of (compactly) describing the corresponding state transition system M
    - formulate the property ϕ (usually in a temporal logic)
    - the model-checker checks whether the property ϕ holds for M (for all states, or in the initial state)
    - returns a counterexample if it doesn't hold

Model-checking for BDI logics

- programs in BDI agent programming languages can be model-checked just like any other program
- however it would be good if the model-checker 'understood' beliefs, desires, plans (so the property ϕ can talk about them)
- the only model-checker which 'understands' epistemic operators (knowledge) is MCMAS (Lomuscio, Qu and Raimondi, MCMAS: A Model Checker for the Verification of Multi-Agent Systems, CAV'09)
- MCMAS implements checking properties along both *temporal* and *knowledge* accessibility relations

Interpreted systems

Interpreted systems

- MCMAS model-checks properties of interpreted systems (Fagin, Halpern, Moses, and Vardi. *Reasoning about Knowledge*, MIT Press, 1995)
- in interpreted systems each state is an n-tuple of the agents' local states and the state of the environment
- indistinguishability relation for agent i, $s \sim_i s'$, holds if the local state of agent i is the same in states s and s'
- the logic over interpreted systems has both temporal and epistemic operators
- formulas are interpreted over computational runs (sequences of states)

Are interpreted systems computationally grounded?

- some have argued that interpreted systems can be seen as a computationally grounded semantics for intentional logics:
- system description in terms of runs (involving local states, protocols, etc.) immediately provides a logical model to evaluate formulae
- epistemic properties are based on the equivalence of local states (which is a concrete computational notion)
- local states could be represented as, e.g., arrays of variables, allowing a ‘fine grained’ description of agents

Example: grounding and interpreted systems

- e.g., a propositional variable $paid_i$ may mean that a variable v_1 in agent i's local state has the value *Paid*
- since $paid_i$ holds in all global states where agent i's state contains $v_1 = Paid$, $K_i\, paid_i$ holds in these global states, which is what we want
- while this "works", it seems a very roundabout way of defining an agent's knowledge:
    - to determine the truth of such formulas we need to examine all *global* states related by $\sim_i$
    - however what the agent actually knows depends on the properties of the agent's *local* state, so why consider all *global* states?

Other problems with interpreted systems

- K_i holds not only for the formulas which correspond to some properties of the agent's state but also for many other formulas including:
    - all tautologies, all logical consequences of the (real) knowledge of the agent, all consequences by introspection and formulas talking about the global properties of the system
    - e.g., if there is just one global state s^0 where agent i's local state is s_i^0, and s^0 has a single successor s^1, then i 'knows' precisely what the next global state looks like
- this is not grounded knowledge ascription — even if the system is entirely deterministic, agent i does not necessarily know this

Computational grounding for classical BDI logics

Verifying agent programming languages

An alternative approach is to model check an agent programming language, e.g.:

- verifying agent programs written in AgentSpeak(F) using the *Spin* model checker (Bordini, Fisher, Pardavila and Wooldridge *Model checking AgentSpeak*, Proc. AAMAS 2004 (2004))
- verifying agent programs directly using Agent Infrastructure Layer/Java Pathfinder (Bordini, Dennis, Farwer, Fisher, *Automated Verification of Multi-Agent Programs*, ASE 2008)

AgentSpeak(F)

- AgentSpeak(L) (Rao 1996) is a descendent of PRS
- AgentSpeak(F) (Bordini et al, 2004) is a finite state version of AgentSpeak(L)
- to ensure that the states of the system are finite, the sizes of types, data structures and communication channels must be specified as part of the system description, e.g.:
  - the maximum number of beliefs in the agent's belief base
  - maximum number of intentions
  - maximum number of pending events, actions and messages
- no function terms and other minor restrictions on syntax

Verifying AgentSpeak(F)

- AgentSpeak(F) programs are translated into the *Promela* modelling language of the *Spin* model checker
- properties are expressed in a simplified BDI logic translated into the LTL based property specification language used by *Spin*
- used to verify, e.g., properties of a simulated auction system implemented in AgentSpeak(F)
- beliefs, goals and intentions are essentially interpreted syntactically as a finite list of formulas rather than using an accessibility relation

Specifying properties of AgentSpeak(F) agents

- belief, desire and intention are defined in terms of the operational semantics of AgentSpeak(F)
    - agent *i* *believes* ϕ, (*Bel* *i* ϕ), if ϕ is present in its belief base (CWA is assumed)
    - an agent *desires* ϕ, and (*Des* *i* ϕ), if ϕ is an achievement goal in the agent's set of events or ϕ is one of the agent's current intentions
    - agent *i* *intends* ϕ, (*Int* *i* ϕ), if ϕ is an achievement goal that appears in the agent's set of intentions—i.e., in the agent's currently executing or suspended plans
- before verification, *Bel*, *Des* and *Int* expressions are translated into expressions that access the AgentSpeak(L) data structures modelled in Promela

Syntactic Belief Ascription

Doing it right

- it is non-trivial to relate 'possible worlds' semantics of classical BDI logics to the knowledge or beliefs of implemented BDI agents
- need to somehow extract the belief etc. accessibility relations from the program, but:
    - interpreted systems approach is rather cumbersome and in many cases does not ground ascription of the agent's beliefs
    - workarounds involving grounding agent's beliefs, goals and intentions in the model checker encoding of the agent's data structures are rather ad hoc
- what we need is a systematic approach to computationally grounded belief ascription

Syntactic belief ascription

- distinguishes between beliefs and reasoning abilities that we *ascribe to* the agent ('the agent's logic') and the logic we use to *reason about* the agent
- agent's beliefs and goals are interpreted syntactically as formulas 'translating' some particular configuration of variables in the agent's internal state, rather than as propositions corresponding to sets of possible worlds or runs of the agent's program
- allows explicit modelling of the computational delay involved in updating the agent's state
- avoids modelling the agent as logically omniscient

Grounded belief ascription

- states are $n + 1$-tuples of local states of n agents and the state of the environment $s = (s_1, \ldots, s_n, e)$
- properties of the system are specified in a language built from a set of propositional variables $\mathcal{P}$
- the set of beliefs ascribable to an agent i, L_i, is a finite set of literals over $\mathcal{P}$
- each agent's state consists of finitely many 'memory locations' $l_1, \ldots, l_m$, and that each location l_j can contain (exactly) one of finitely many values, $v_{j1}, \ldots, v_{jk}$
- each literal in L_i corresponds to a set of memory locations having a particular set of values, but 'translates' this into a statement about the world

Grounded belief ascription 2

- we assume a mapping $\mathcal{A}_i$ which assigns to each state s a set of literals that form the beliefs of agent i in state s
- this 'translation' is fixed and does not depend on the truth or falsity of the formulas in the real world
- in general, there is no requirement that $\mathcal{A}_i$ be consistent — if a propositional variable and its negation are associated with two different memory locations then the agent may simultaneously believe that p and $\neg p$
- nor does $\mathcal{A}_i$ have to map a single value to a single belief
- conversely, we don't assume that for every $p \in \mathcal{P}$ either p or $\neg p$ belongs to $\mathcal{A}_i$

Semantics

- the transitions of the agent-environment system are modelled as a kind of Kripke structure
- beliefs of agents are modelled as a local property of each agent's state using the syntactic assignment $\mathcal{A}_i$ corresponding to agent i's beliefs
- state of the environment e corresponds to a classical possible world (complete truth assignment to propositional variables in $\mathcal{P}$)
- agent i believes that p in state s, $M, s \models B_i p$, if $p \in \mathcal{A}_i(s)$
- technically equivalent to the syntactic model of belief in interpreted systems, but we show how to ground $\mathcal{A}_i$ in the values of variables in the agent's state

Closure assumptions

- for an agent which choses actions based on its beliefs, assuming deductive closure of its beliefs is only safe if:
    - closure is with respect to the agent's 'internal logic' implemented by the agent program (i.e., the postulated consequences are actually derivable)
    - it is reasonable to assume that the agent's deductive algorithm completes within the timestep implied by the modelling requirements (i.e., we are not concerned with the precise timing of the agent's response to a query)
- otherwise we need to model each inference step in the agent's internal logic as an explicit transition of the system — dynamic syntactic logics

Example: PDL-APL

Verification of SimpleAPL programs

- PDL-APL is a logic based on syntactic belief ascription which can be used to verify SimpleAPL agents
- SimpleAPL is a simple BDI agent programming language allows the implementation of agents with beliefs, goals, actions, plans, and planning rules
- ‘translate’ a SimpleAPL agent program Λ into a PDL program expression $\xi(\Lambda)$
- state properties of the program in PDL-APL, e.g., $Gp \rightarrow \langle \xi(\Lambda) \rangle Bp$
- show that the property can be derived from the axioms (using a PDL theorem prover)

SimpleAPL beliefs & goals

- the **beliefs** of a SimpleAPL agent represent its information about its environment and itself
    - beliefs are represented by a set of positive literals
- the agent's **goals** represent situations the agent wants to realise (not necessarily all at once)
    - goals are represented by a set of arbitrary literals
- the beliefs and goals of an agent are related to each other:
    - if an agent believes p, then it will not pursue p as a goal
    - if an agent does not believe that p, it will not have $-p$ as a goal
- the initial beliefs and goals of an agent are specified by its program

SimpleAPL basic actions

- a **belief test action** ϕ? tests whether a boolean belief expression ϕ is entailed (CWA) by the agent's beliefs
- a **goal test action** ψ! tests whether a disjunction of goals ψ is entailed (classically) by the agent's goals
- **belief update actions** ("external actions) change the beliefs (and goals) of the agent
    - a belief update action is specified in terms of its pre- and postconditions (sets of literals)
    - an action can be executed if one of its pre-conditions is entailed by the agent's current beliefs
    - executing the action updates the agent's beliefs to make the corresponding postcondition entailed by the agent's beliefs

SimpleAPL plans

- **plans** are sequences of basic actions composed by plan composition operators:
  - **sequence:** "$\pi_1 ; \pi_2$" (do π_1 then π_2)
  - **conditional choice:** "$\texttt{if } \phi \texttt{ then } \{\pi_1\} \texttt{ else } \{\pi_2\}$"
  - **conditional iteration:** "$\texttt{while } \phi \texttt{ do } \{\pi\}$"

SimpleAPL rules

- **planning goal rules** are used for plan selection based on the agent's current goals and beliefs
- a planning goal rule $\kappa \leftarrow \beta \,|\, \pi$ consists of three parts:
  - κ: an (optional) **goal query** which specifies which goal(s) the plan achieves
  - β: a **belief query** which characterises the situation(s) in which it could be a good idea to execute the plan
  - π: a **plan**
- a rule can be applied if κ is entailed by the agent's goals and β is entailed by the agent's beliefs
- applying the rule adds π to the agent's plans

SimpleAPL operational semantics

- we define the operational semantics of SimpleAPL in terms of a transition system
- states are **agent configurations** $\langle \sigma, \gamma, \Pi \rangle$ where σ, γ are sets of literals representing the agent's beliefs and goals, and Π is a set of plan entries representing the agent's current active plans
- each transition corresponds to a single step in the execution of the agent
- different execution strategies give rise to different semantics
- for simplicity we focus on non-interleaved execution — i.e., the agent executes a single plan to completion before choosing another plan

Formal entailment definitions

- $\models_{cwa}$ (belief entailment for closed world assumption):

  $\sigma \models_{cwa} p$ iff $p \in \sigma$

  $\sigma \models_{cwa} -p$ iff $p \notin \sigma$

  $\sigma \models_{cwa} \phi$ and ψ iff $\sigma \models_{cwa} \phi$ and $\sigma \models_{cwa} \psi$

  $\sigma \models_{cwa} \phi$ or ψ iff $\sigma \models_{cwa} \phi$ or $\sigma \models_{cwa} \psi$

  $\sigma \models_{cwa} \{\phi_1, \ldots, \phi_n\}$ iff $\forall 1 \leq i \leq n \;\; \sigma \models_{cwa} \phi_i$

- $\models_g$ (goal entailment):

  $\gamma \models_g p$ iff $p \in \gamma$

  $\gamma \models_g -p$ iff $-p \in \gamma$

  $\gamma \models_g \phi$ or ψ iff $\gamma \models_g \phi$ or $\gamma \models_g \psi$

Belief update function

- let a be a belief update action and σ a belief base such that $\sigma \models_{cwa} \mathtt{prec_j}(a)$
- intuitively, $\sigma \models_{cwa} \mathtt{prec_j}(a)$ if it contains all positive literals in $\mathtt{prec_j}(a)$ and does not contain the negative ones
- the result of executing belief update action a with respect to σ is defined as:

$$T_j(a, \sigma) = (\sigma \cup \{p : p \in \mathtt{post_j(a)}\}) \setminus \{p : \neg p \in \mathtt{post_j(a)}\}$$

- intuitively, the result of the update satisfies (entails under $\models_{cwa}$) the corresponding postcondition $\mathtt{post_j(a)}$

Transitions: basic actions

- belief and goal test actions

$$\frac{\sigma \models_{cwa} \beta}{\langle \sigma, \gamma, \{\beta?; \pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi\}\rangle}$$

$$\frac{\gamma \models_{g} \kappa}{\langle \sigma, \gamma, \{\kappa!; \pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi\}\rangle}$$

- belief update actions

$$\frac{\sigma \models_{cwa} \mathtt{prec}_j(a) \qquad T_j(a, \sigma) = \sigma'}{\langle \sigma, \gamma, \{a; \pi\}\rangle \longrightarrow \langle \sigma', \gamma', \{\pi\}\rangle}$$

where $\gamma' = \gamma \setminus (\{p : p \in \sigma'\} \cup \{-p : p \notin \sigma'\})$

Transitions: plans

- conditional choice

$$\frac{\sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \{(\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2); \pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_1; \pi\}\rangle}$$

$$\frac{\sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \{(\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2); \pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_2; \pi\}\rangle}$$

- conditional iteration

$$\frac{\sigma \models_{cwa} \phi}{\langle \sigma, \gamma, \{(\texttt{while } \phi \texttt{ do } \pi_1); \pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi_1; (\texttt{while } \phi \texttt{ do } \pi_1); \pi\}\rangle}$$

$$\frac{\sigma \not\models_{cwa} \phi}{\langle \sigma, \gamma, \{(\texttt{while } \phi \texttt{ do } \pi_1); \pi\}\rangle \longrightarrow \langle \sigma, \gamma, \{\pi\}\rangle}$$

Transitions: rules

- planning goal rules $\kappa \leftarrow \beta \,|\, \pi$

$$\frac{\gamma \models_{g} \kappa \qquad \sigma_{cwa} \models \beta}{\langle \sigma, \gamma, \{\} \rangle \longrightarrow \langle \sigma, \gamma, \{\pi\} \rangle}$$

Logic

- logic allows us to specify properties of:
    - a particular SimpleAPL program Λ, e.g., "for every execution of Λ, ϕ holds"
    - the SimpleAPL architecture, e.g., blind commitment: "the agent either keeps its goal or believes that it's been achieved"
- the language is PDL plus syntactic belief and goal operators
- the models capture *all possible basic transitions between the belief and goal states of an agent* (regardless of the agent's plans)

Syntax of PDL-APL

The syntax of PDL-APL is defined relative to a SimpleAPL program Λ

- *Prop*: the set of positive literals in Λ
- *Ac*: the set of basic actions in Λ
- *Prog*: $a \in Ac \mid \phi? \mid \alpha_1 ; \alpha_2 \mid \alpha_1 \cup \alpha_2 \mid \alpha^*$
- formulas ($p \in Prop$):

$$Bp \mid Gp \mid G{-}p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid [\alpha]\phi$$

Semantics of PDL-APL

Models are also defined relative to Λ (namely pre- and post-conditions of actions $C(a)$, for each $a \in Ac$)

$M = (S, \{R_a : a \in Ac\}, L)$, where

- S is a non-empty set of states
- $L = (L_b, L_g)$ is the labelling function consisting of belief and goal labelling functions L_b and L_g:
  - $L_b : S \longrightarrow 2^{Prop}$
  - $L_g : S \longrightarrow 2^{Prop \cup Prop^-}$ where $Prop^- = \{-q : q \in Prop\}$
- R_a for each $a \in Ac$ conforms to a's pre and postconditions in Λ; namely $R_a(s, s')$ iff for some $(\mathtt{prec}_j, \mathtt{post}_j) \in C(a)$,
  - $T_j(a, L_b(s)) = L_b(s')$
  - $L_g(s') = L_g(s) \setminus (\{p : p \in L_b(s')\} \cup \{-p : p \notin L_b(s')\}$

Truth definition

- $M, s \models Bp$ iff $p \in L_b(s)$
- $M, s \models Gp$ iff $p \in L_g(s)$
- $M, s \models G - p$ iff $-p \in L_g(s)$
- $M, s \models \neg\phi$ iff $M, s \not\models \phi$
- $M, s \models \phi \wedge \psi$ iff $M, s \models \phi$ and $M, s \models \psi$
- $M, s \models [\alpha]\phi$ iff for all $s' \in S$ such that $R_\alpha(s, s')$ we have that $M, s' \models \phi$.

Conditions on beliefs and goals (beliefs and goals are disjoint)

$$\neg(p \in L_b(s) \wedge p \in L_g(s)), \qquad \neg(p \notin L_b(s) \wedge -p \in L_g(s))$$

Translation of belief and goal queries

To express pre- and postconditions or actions and belief and goal queries in the language of PDL-APL, we introduce translation functions f_b and f_g:

$f_b(p) = Bp$

$f_b(-p) = \neg Bp$

$f_b(\phi \texttt{ and } \psi) = f_b(\phi) \wedge f_b(\psi)$

$f_b(\phi \texttt{ or } \psi) = f_b(\phi) \vee f_b(\psi)$

$f_b(\phi_1, \dots, \phi_k) = \bigwedge_{i \in \{1,\dots,k\}} f_b(\phi_i)$

$f_g(p) = Gp$

$f_g(-p) = G - p$

$f_g(\phi \texttt{ or } \psi) = f_g(\phi) \vee f_g(\psi)$

Complete and sound axiomatisation of PDL-APL(C)

Axioms are parameterised by a set of pre- and postconditions C

CL classical propositional logic

PDL axioms of PDL

A1 $Bp \rightarrow \neg Gp$

A2 $G - p \rightarrow Bp$

A3 $f_b(\mathtt{prec}_j) \wedge \Phi \rightarrow [a](f_b(\mathtt{post_j}) \wedge \Phi)$
where $(\mathtt{prec}_j, \mathtt{post}_j) \in C(a)$ and Φ does not contain propositional variables occurring in $\mathtt{post}_j$

A4 $\neg f_b(\mathtt{prec}_1) \wedge \ldots \wedge \neg f_b(\mathtt{prec}_k) \rightarrow \neg\langle a\rangle \mathit{true}$
where $C(a) = \{(\mathtt{prec}_1, \mathtt{post}_1), \ldots, (\mathtt{prec}_k, \mathtt{post}_k)\}$

A5 $f_b(\mathtt{prec}_j) \rightarrow \langle a\rangle \mathit{true}$ where $(\mathtt{prec}_j, \mathtt{post}_j) \in C(a)$

Verification

Translation of plan expressions

We also need a translation function f_p to translate SimpleAPL plans into the language of PDL-APL

$f_p(a) = a$

$f_p(\phi?) = f_b(\phi)?$

$f_p(\psi!) = f_g(\psi)?$

$f_p(\pi_1; \pi_2) = f_p(\pi_1); f_p(\pi_2)$

$f_p(\texttt{if } \phi \texttt{ then } \pi_1 \texttt{ else } \pi_2) = (f_b(\phi)?; f_p(\pi_1)) \cup (\neg f_b(\phi)?; f_p(\pi_2))$

$f_p(\texttt{while } \phi \texttt{ do } \pi) = (f_b(\phi)?; f_p(\pi))^*; \neg f_b(\phi)?$

Translation of the agent's program

- suppose the agent's program Λ has a set of PG rules

$$PG = \{r_i | r_i = \kappa_i \leftarrow \beta_i | \pi_i\}$$

- the translation of the agent's program

$$\xi(\Lambda) = (\bigcup_{r_i \in PG} (f_g(\kappa_i) \wedge f_b(\beta_i))? \; ; \; f_p(\pi_i) \;)^+$$

- where $^+$ is the strict transitive closure operator: $\alpha^+ = \alpha; \alpha^*$
- this states that each planning goal rule is be applied zero or more times (but at least one planning goal rule will be applied)
- the idea is that this expression describes exactly the paths in the operational semantics

Correspondence

We can show that this translation is *faithful*: the PDL program expression which is the translation of the agent's program corresponds to the set of paths in the transition system generated by the operational semantics for that agent program

- a model *generated* by a state s_0 consists of all possible states which can be recursively reached from s_0 by following the basic relations
- a state s and a configuration $c = \langle \sigma, \gamma, \Pi \rangle$ are **matching**, $s \sim c$, if they have the same belief and goal bases, i.e., $L_b(s) = \sigma$ and $L_g(s) = \gamma$
- let transition system TS and model M correspond to an agent program Λ with a set of pre- and postconditions for actions C. TS and M are *matching* if they are generated by c_0 and s_0 such that $s_0 \sim c_0$

Correspondence theorem

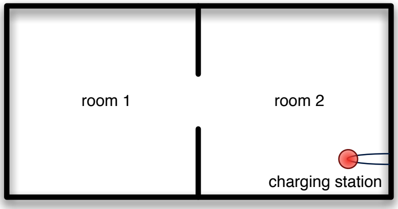
Theorem (Correspondence of TS and M)

If TS and M match, then a configuration c with an empty plan base is reachable from the initial configuration c_0 in TS iff a matching state s is reachable from the initial state s_0 along a path described by $\xi(\Lambda)$, i.e., $(s_0, s) \in R_{\xi(\Lambda)}$

If we want to know whether a configuration is reachable in the operational semantics, we can check whether a PDL-APL formula (the translation of the SimpleAPL program) is derivable from the axioms describing the corresponding models

Example: robot vacuum cleaner

- based on a simple agent described in Russell & Norvig (2003)
- agent has to clean two rooms: *room1* and *room2*
- agent has sensors that tell it if a room is clean and whether its battery is charged
- vacuuming a room results in the room being clean and discharges the agent's battery
- agent can recharge its battery at a recharging station in *room2*



Example: vacuum cleaner

- suppose the agent has the belief update actions:

```
{room1}             moveR  {-room1, room2}
{room1, battery}    suck   {clean1, -battery}
{room2, battery}    suck   {clean2, -battery}
{room2}             moveL  {-room2, room1}
{room2, -battery}   charge {battery}
```

- and PG rules:

```
$c_1$ <-  $b$ | if $r_1$ then {$s$} else {$l$; $s$}
$c_2$ <-  $b$ | if $r_2$ then {$s$} else {$r$; $s$}
   <- $-b$ | if $r_2$ then {$c$} else {$r$; $c$}
```

- we abbreviate beliefs and goals as r_1, r_2, c_1, c_2, b and actions as r, s, l, c

Example: vacuum cleaner

- corresponding PDL-APL program expression:

$$\begin{aligned}\texttt{vac} \quad =_{df} \quad & ((Gc_1 \wedge Bb)?; (Br_1?; s) \cup (\neg Br_1?; l; s)) \cup \\ & ((Gc_2 \wedge Bb)?; (Br_2?; s) \cup (\neg Br_2?; r; s)) \cup \\ & (\neg Bb?; (Br_2?; c) \cup (\neg Br_2?; r; c))\end{aligned}$$

Sample axiom instances

A3r $Br_1 \wedge Bc_1 \wedge \neg Bb \wedge Gc_2 \rightarrow [r](Br_2 \wedge Bc_1 \wedge \neg Bb \wedge Gc_2)$

A3s1 $Br_1 \wedge Bb \wedge Gc_1 \wedge Gc_2 \rightarrow [s](Bc_1 \wedge Gc_2 \wedge Br_1 \wedge \neg Bb)$

A3s2 $Br_2 \wedge Bb \wedge Gc_1 \wedge Gc_2 \rightarrow [s](Bc_2 \wedge \neg Bb \wedge Br_2 \wedge Gc_1)$

A3c $Br_2 \wedge Bc_1 \wedge \neg Bb \wedge Gc_2 \rightarrow [c](Br_2 \wedge Bb \wedge Bc_1 \wedge Gc_2)$

A4r $\neg Br_1 \rightarrow \neg\langle r\rangle\top$

A5s $Br_1 \wedge Bb \rightarrow \langle s\rangle\top$.

Properties

We can use a PDL theorem prover to verify properties such as:

- if the agent has goals to clean rooms 1 and 2, and starts in the state where its battery is charged and it is in room 1, it can reach a state where both rooms are clean:

$$Gc_1 \wedge Gc_2 \wedge Bb \wedge Br_1 \rightarrow \langle \mathtt{vac}^3 \rangle (Bc_1 \wedge Bc_2)$$

- the agent is guaranteed to achieve its goal (after 3 iterations of the program)

$$Gc_1 \wedge Gc_2 \wedge Bb \wedge Br_1 \rightarrow [\mathtt{vac}^3](Bc_1 \wedge Bc_2)$$

  where $\mathtt{vac}^3$ stands for `vac` repeated three times

- the agent is blindly committed to its intentions:

$$Gc_1 \rightarrow [\mathtt{vac}^+](Bc_1 \vee Gc_1)$$

Summary

- "standard" BDI logics allow properties of beliefs, desires and intentions, committment strategies, communication semantics etc. to be formalised
- the resulting specifications can be model checked using model checkers such as MCMAS
- however it is not clear how to implement agents based on these specifications
- in particular, what corresponds to belief and goal accessibility relations in the agent programming language / implemented agent?

Summary 2

- "syntactic" BDI logics allow more accurate modelling of feasible agents
- we can verify properties of real agent programs at the belief and goal level (as opposed to simply verifying the agent program as just a computer program)
- many challenges remain:
  - formalising the agent's deliberation cycle (e.g., to allow verification of commitment strategies)
  - formalising beliefs, goals and intentions and interaction between agents in multi-agent systems (e.g., to allow verification of teamwork)
  - practical issues (e.g., scalability)