A computationally grounded account of belief and awareness for AI agents

Natasha Alechina & Brian Logan

University of Nottingham
School of Computer Science

Outline

- logic for verification
- the *Belief-Desire-Intention* model of agency
- standard epistemic logic is not computationally grounded
- syntactic belief ascription
- verifying agent programs

Logic for verification

- why model agents in logic?
- specification and verification of agent programs is a key problem in agent research and development
- formal verification provides a degree of certainty regarding system behaviour which is difficult or impossible to obtain using testing or simulation
- e.g., to check that an agent architecture conforms to general principles of rational agency, or that an agent program will achieve the agent's goals
- to be able to verify behaviour of agent programs our logic must be *computationally grounded*

The *Belief-Desire-Intention* model of agency

- arguably the most widely adopted approach to modelling AI agents
- agents are both characterised and programmed in terms of propositional attitudes such as *beliefs* and *goals* and the relationships between them
- BDI model is often formalised in *BDI logics*
- for BDI logics to be useful we must be able to correctly ascribe beliefs and other propositional attitudes to agents
- how can we ensure that BDI logics are computationally grounded?

Standard BDI logics

- BDI logics for modelling an agent's knowledge and belief are typically based on standard epistemic logics with possible worlds semantics
    - an agent i believes ϕ in state s if ϕ is true in all states s' which are belief-accessible from s, $s \sim_i s'$
        - for knowledge the accessibility relation $\sim_i$ is usually assumed to be an equivalence relation between states
- these logics have attractive formal properties and allow properties of beliefs and other intentional attitudes of agents to be formalised
- agent specifications can be verified using model-checkers such as MCMAS

Standard BDI logics are not computationally grounded

- however with standard BDI logics it's not clear how to connect the computational implementation of an agent to the beliefs we ascribe to it

- two main problems:

  - how to ground the agent's beliefs in its computational state (belief ascription)

  - how to ground the evolution of the agent's beliefs in its computation (logical omniscience)

Belief ascription

- possible worlds semantics are generally *ungrounded*

- "*there is usually no precise relationship between the abstract accessibility relations that are used to characterise an agent's state, and any concrete computational model*" (van der Hoek & Wooldridge 2003)

- to generate a logical model we need to somehow extract the belief accessibility relations from the program

- difficult to use BDI logics for verification of *agent programs*

- various workarounds in the literature – e.g., when model checking the LORA architecture Bordini et al (2004) model beliefs syntactically as a finite list of formulas rather than using an accessibility relation

Logical omniscience

- agent knows all logical tautologies and *all consequences of its knowledge*
- problematic when attempting to build realistic models of agent behaviour
    - closure under logical consequence implies that deliberation takes no time
    - however processes such belief revision, planning and problem solving do take time (often exponential time) and a whether an agent can find a solution *in time* is often a key issue
- modelling such processes as derivations in a logical language can over-state the capabilities of the agent

Interpreted systems

- 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 *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 grounded semantics for intentional logics:

    - since the semantics of interpreted systems refers to computational runs, a system description in terms of runs (using 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, thereby allowing for a ‘fine grained’ description of agents

Example: grounding an interpreted system

- 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

- however this 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 we already decided that what the agent actually knows depends on the properties of the agent's *local* state

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:
    - 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^0_i , 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

Syntactic belief ascription

- alternative approach to computationally grounded belief ascription
- distinguishes between beliefs and reasoning abilities which 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 states 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 $s = (s_1, ..., s_n, e)$

- 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' $l_1, ... , l_m$, and that each location l_j can contain (exactly) one of finitely many values, $v_{j1}, ..., v_{jk}$

- 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 ¬*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 ¬*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: verifying SimpleAPL agents

- use of syntactic epistemic logic in verifying *SimpleAPL* programs (e.g., Alechina et al 2007)

- *SimpleAPL* is simplified version of the BDI-based agent programming languages 3APL and 2APL

- programs are specified in terms of the agent's beliefs, goals and planning goal rules which specify which plans the agent should adopt given its goals and beliefs

- programs have explicit data structures for beliefs and goals and *SimpleAPL* agent's don't do any inference, so belief ascription is straightforward

Example: verifying SimpleAPL agents

- an agent program is axiomatised in Propositional Dynamic Logic (PDL) extended with *syntactic* belief and goal operators

- properties such as “all executions of this program starting in a state with initial beliefs p_1, ..., p_n and goals κ_1, ..., κ_m will achieve the agent’s goals” can be verified by checking whether

$$\bigwedge_{i=n}^{i=1} Bp_i \wedge \bigwedge_{j=m}^{j=1} G\kappa_j \rightarrow [prog] \bigwedge_{j=m}^{j=1} B\kappa_j$$

- is derivable from the axiomatisation of the agent program, where *prog* is a translation of the agent’s program into PDL with syntactic belief and goal operators

Summary

- to be useful, logics for agents must be computationally grounded
- standard BDI logics (e.g., interpreted systems) are not computationally grounded and may incorrectly ascribe beliefs to agents
- ‘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)

Future work

- 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)