REASONING AND PROGRAMMING WITH RATIONAL AGENTS

M Fisher

Rapporteur: Dr B N Rossiter
Lecture Two

- Proof in agent theories
  - translation methods for modal logics
  - clausal resolution for temporal + modal logics

- Programming based on rational agent theories
  - extending Java with BDI concepts
  - agent-oriented programming
  - executable temporal + modal logics
We have seen how to represent rational agents (and their activity) in a logical framework.

Most of the uses we might have for such agent theories will involve logical reasoning.

So, we here consider the mechanisation of proof in the combined modal and temporal logics required.

There are several techniques that can be used, but we will only consider two based on resolution, and so provide

1. a brief outline of the translation approach to modal logics, and

2. a more detailed description of the clausal resolution approach to combined modal and temporal logics.
Translation Methods

The translation approach to proof in modal logic is based on the idea that inference in (combinations of) modal logics can be carried out by translating modal formulae into first-order logic and utilising conventional first-order theorem proving.

To describe this approach, we will consider one particular modal logic, namely S5, which is widely used as a logic of knowledge.

Thus, a translation function, $\pi$, from S5 (evaluated at a specific world) to first-order logic, can be provided.

First, classical operators have simple translations:

- $\pi(\text{true}, x) = \text{true}$
- $\pi(\neg \varphi, x) = \neg \pi(\varphi, x)$
- $\pi(\varphi \lor \psi, x) = \pi(\varphi, x) \lor \pi(\psi, x)$

Propositions in the modal logic become identifiable predicates in first-order logic:

- $\pi(p, x) = q_p(x)$

Finally, modalities (represented by 'K') also become (different) identifiable predicates in first-order logic:

- $\pi(K\varphi, x) = q_{K\varphi}(x)$

Translating Axioms

A translation of the definition of the modal operator 'K' is also provided:

$$\forall x. (q_{K\varphi}(x) \iff \forall y. (R(x, y) \Rightarrow \pi(\varphi, y)))$$

as are translations of the properties of the accessibility relation (in this case, $R$) relevant to S5:

- transitivity — $\forall x, y. (q_{K\varphi}(x) \land R(x, y)) \Rightarrow q_{K\varphi}(y)$
- symmetry — $\forall x, y. (q_{K\varphi}(y) \land R(x, y)) \Rightarrow q_{K\varphi}(x)$
- reflexivity — $\forall x. R(x, x)$

This ensures that the full translation function $\Pi$ (a combination of $\pi$ and the above axiom translations) satisfies

**Theorem 1** $\varphi$ is satisfiable in a model $M$ and a world $w$ if, and only if, $\Pi(\varphi, w)$ is first-order satisfiable.
Translation Method: Summary

The translation approach has many advantages:

- by translating to classical first-order logic, we are able to utilise current technology in automated reasoning;
- as can be seen above, any modal logic that is first-order definable can be translated in this way;
- completeness of the translation method often follows easily from completeness of the first-order counterpart.

However, this approach also has one main problem that causes us difficulties, namely:

- the translation approach is not yet able to handle temporal logics such as PTL.

Clausal Resolution

Next, we consider a proof method for combinations of temporal and modal logics based upon the use of clausal resolution.

As in classical logic, this clausal approach involves first transforming formulae into a particular normal form, in our case called “Separated Normal Form”.

Again, to describe how this method works, we will consider one particular combination, namely PTL combined with an S5 modal logic — widely used as a temporal logic of knowledge.

We begin with resolution in the temporal case.
Recall that PTL embodies a discrete, linear model of time, with a finite past and infinite future, i.e.,

$$\sigma = w_0, w_1, w_2, w_3, \ldots$$

Note that $\sigma$ encapsulates both a set of worlds and accessibility relation on these worlds.

The language is that of classical logic extended with various modalities, e.g., $\Diamond$, $\Box$, $U$, $W$ start, $\square$.

**Semantics:**

- $\langle \sigma, \pi, i \rangle \models \Diamond A$ iff $\langle \sigma, \pi, i + 1 \rangle \models A$
- $\langle \sigma, \pi, i \rangle \models \Box A$ iff for all $j \geq i. \langle \sigma, \pi, j \rangle \models A$
- $\langle \sigma, \pi, i \rangle \models \Diamond A$ iff exists $j \geq i. \langle \sigma, \pi, j \rangle \models A$
- $\langle \sigma, \pi, i \rangle \models A \cup B$ iff exists $k \geq i. \langle \sigma, \pi, k \rangle \models B$
  and
  for all $k > j \geq i. \langle \sigma, \pi, j \rangle \models A$
- $\langle \sigma, \pi, i \rangle \models \text{start}$ iff $i = 0$

**Separated Normal Form**

A temporal formula in Separated Normal Form (SNF) is of the form

$$\Box \bigwedge_{i=1}^{n} (P_i \Rightarrow F_i)$$

where each of the $'P_i \Rightarrow F_i'$ (called clauses) is one of the following

- $\text{start} \Rightarrow \bigvee_{k=1}^{r} l_k$ (an initial clause)
- $\bigwedge_{j=1}^{q} m_j \Rightarrow \bigcap_{j=1}^{q} \bigvee_{k=1}^{r} l_k$ (a step clause)
- $\bigwedge_{j=1}^{q} m_j \Rightarrow \Diamond l$ (a sometime clause)

where each $l$, $l_k$ or $m_j$ is a literal.

Initial clauses — provide initial constraints.

Step clauses — provide constraints on the next step.

Sometime clauses — provide constraints on the future.
Examples: SNF

We can provide simple examples showing some of the properties that might be represented directly as SNF clauses.

- Specifying initial conditions:
  
  \[ \text{start} \Rightarrow \text{sad} \]

- Defining transitions between states:
  
  \[ (\text{sad} \land \neg \text{rich}) \Rightarrow \Diamond \text{sad} \]

- Introducing new eventualities (goals):
  
  \[ (\neg \text{resigned} \land \text{sad}) \Rightarrow \Diamond \text{famous} \]

  \[ \text{sad} \Rightarrow \Diamond \text{happy} \]

- Introducing permanent properties:
  
  \[ \text{lottery-win} \Rightarrow \Box \Diamond \text{rich} \]

  which, in SNF, becomes

  \[ \text{lottery-win} \Rightarrow \Box \text{rich} \]

  \[ \text{lottery-win} \Rightarrow \Box x \]

  \[ x \Rightarrow \Box \text{rich} \]

  \[ x \Rightarrow \Box x \]

Resolution Rules

- Initial Resolution:
  
  \[ \begin{align*}
  \text{[IRES]} & \quad \text{start} \Rightarrow (A \lor I) \\
  & \quad \text{start} \Rightarrow (B \lor \neg I) \\
  & \quad \text{start} \Rightarrow (A \lor B)
  \end{align*} \]

- Step Resolution:
  
  \[ \begin{align*}
  \text{[SRES]} & \quad P \Rightarrow \Box (A \lor I) \\
  & \quad Q \Rightarrow \Diamond (B \lor \neg I) \\
  \Rightarrow (P \land Q) \Rightarrow \Box (A \lor B)
  \end{align*} \]

- Temporal Resolution:
  
  \[ \begin{align*}
  \text{[TRES]} & \quad A \Rightarrow \Box \square \neg I \\
  & \quad Q \Rightarrow \Diamond I \\
  \Rightarrow Q \Rightarrow (\neg A) \Diamond I
  \end{align*} \]

- Transferral:
  
  \[ \begin{align*}
  \text{[TRAN]} & \quad A \Rightarrow \text{false} \\
  & \quad \text{start} \Rightarrow \neg A \\
  & \quad \text{true} \Rightarrow \Box \neg A
  \end{align*} \]

- Termination:
  
  \[ \text{start} \Rightarrow \text{false} \]
Adding Modal Dimensions

The previous resolution rules provide a complete deductive system for PTL.

Now, if we want to add a modal dimension, say one characterised by the S5 modality ‘K’ we must extend the normal form and add appropriate resolution rules.

$\text{SNF}_K$ is essentially SNF extended with an additional type of clause, namely a modal clause:

$$\bigwedge_{j=1}^q m_j \Rightarrow \bigvee_{k=1}^r M l_k$$

where $M$ is either ‘$K$’ or ‘$\neg K$’.

Modal clauses provide constraints relating to the modal dimension, in this case S5.

Examples: $\text{SNF}_K$

Two simple examples of $\text{SNF}_K$ clauses are as follows.

- Positive knowledge: $\text{(sad} \land \neg \text{oblivious) } \Rightarrow K \text{sad}$
- Negative knowledge: $\text{happy } \Rightarrow \neg K \text{sad}$

(Sample) Modal Resolution Rules

\[
\begin{align*}
\text{[MRES1]} & \quad P \Rightarrow D \lor KI \\
& \quad Q \Rightarrow D' \lor \neg KI \\
& \quad (P \land Q) \Rightarrow D \lor D'
\end{align*}
\]

\[
\begin{align*}
\text{[MRES2]} & \quad P \Rightarrow D \lor KI \\
& \quad Q \Rightarrow D' \lor K \neg I \\
& \quad (P \land Q) \Rightarrow D \lor D'
\end{align*}
\]

N.B.:

- modal resolution rules apply to modal clauses;
- step and temporal resolution rules apply to step and sometime clauses;
- initial resolution rules apply to initial clauses.
**Simple Example**

Attempt to prove

\( \Box Ks\text{ad} \Rightarrow \Box \neg Ks\text{ad} \)

Negate, giving

\( \Box Ks\text{ad} \land \Box \neg Ks\text{ad} \)

and rewrite to SNF\(K\):

1. \text{start} \Rightarrow x
2. \quad x \Rightarrow \Box y
3. \quad y \Rightarrow Ks\text{ad}
4. \quad x \Rightarrow \Box z
5. \quad z \Rightarrow Ks\text{ad}

Now carry out resolution as follows:

6. \( y \land z \Rightarrow \text{false} \) \( [3,5, \text{MRES2}] \)
7. \( \text{true} \Rightarrow \Box (\neg y \lor \neg z) \) \( [6, \text{TRAN}] \)
8. \( x \Rightarrow \Box \neg z \) \( [2,7, \text{SRES}] \)
9. \( x \Rightarrow \Box \text{false} \) \( [4,8, \text{SRES}] \)
10. \text{start} \Rightarrow \neg x \) \( [9, \text{TRAN}] \)
11. \text{start} \Rightarrow \text{false} \) \( [1,9, \text{IRES}] \)

---

**Multi-Modal Version**

In order to add multiple modal dimensions, we just

- allow a new type of modal clause for every modality allowed; and
- add modal resolution rules corresponding to the particular properties of that modal dimension.

**Example**

Imagine we have multiple S5 modalities, \( K_{me}, K_{you} \), etc.

We can see how a formula within the logic, for example

\[ [K_{me}K_{you} \text{my.key} \land K_{me} \text{msg.sent}] \Rightarrow \Box K_{you} \text{msg.contents} \]

becomes

\[ [K_{me}K_{you} \text{my.key} \land K_{me} \text{msg.sent}] \Rightarrow x \]

\[ x \Rightarrow \Box K_{you} \text{msg.contents} \]

and then

1. \( x \Rightarrow \Box y \)
2. \( y \Rightarrow K_{you} \text{msg.contents} \)
3. \( \neg x \Rightarrow \neg K_{me} \neg z \lor \neg K_{me} \text{msg.sent} \)
4. \( z \Rightarrow \neg K_{you} \text{my.key} \)
The combination of (multi-) modal and temporal logics is very powerful.

If there is little interaction between the various temporal and modal dimensions, then reasoning within such logics is tractable.

However, once we model agents, we tend to introduce many axioms incorporating interactions, for example

Perfect Recall: models agents whose knowledge grows or increases over time (also called no forgetting).

The axiom characterising this interaction is:

\((\text{synchrony +) perfect recall: } K\Box \varphi \Rightarrow \Box K\varphi)\)

Unfortunately, interactions such as this make manipulation (e.g. proof) harder — other combinations are much worse leading to undecidability even in the propositional case!

- The clausal resolution approach is able to handle a range of temporal and modal logics, together with their combinations.

- Interactions between dimensions introduce complexity and require resolution rules that apply to, for example, both modal and step clauses simultaneously.

- Resolution rules have not yet been defined for the wide variety of modal logics.

- One hybrid approach is to use translation methods for the modal dimensions and clausal resolution methods for the temporal dimensions.
References

Translation Approach:


Clausal Resolution Approach:

Introduction

As we have seen, a wide range of agent theories can be developed, based on different combinations of modal and temporal logics.

But how might we use them?

One approach is to use such a theory as the basis for a high-level agent programming language.

This, hopefully, provides a close link between the theory and implementation, as well as providing high-level concepts within the programming language.

We will look, in varying detail, at several programming languages that have attempted to capture the ideas behind rational agency.

Overview

To show the variety of approaches followed, we will briefly examine:

- **JACK**
  — extending Java with rational agent concepts

- **Agent-Oriented Programming (AOP)**
  — a high-level view of agent programming based upon rational agent concepts

- **METATEM**
  — directly executing combined temporal and modal logic specifications in order to implement rational agents
Extending Java with BDI Concepts

We begin by looking at an extension of a standard programming language that attempts to capture elements of rationality.

Ten years ago, developing any sort of interacting agent was quite difficult. However, since the Java language became widely available, programming simple agents became much easier.

There have been several attempts to provide packages/libraries on top of Java that implement concepts from rational agency.

Here, we briefly overview JACK, developed by Agent Oriented Software Ltd., Australia.

The developers of JACK claim that it “allows for a variety of types of software agent to be layered on top of the base kernel, from simple agents (e.g., information retrieval agents) through to more capable Belief Desire Intent (BDI) agents . . .”

JACK Components

JACK consists of extensions to Java providing, for example:

- core Agent classes that provide basic agent behaviour, e.g.
  
  \[ \text{agent MyAgent extends Agent} \]

- a Database class which captures beliefs;

- other key classes such as Capability, Event and Plan;

- the ability to define methods that can carry out reasoning over the agent's attributes;

- extension of the event model to allow objects such as those of class BDIGoalEvent to be communicated;

- a set of statements for the manipulation of an agent's state (for instance, additions of new goals or sub-goals to be achieved, changes of beliefs, interaction with other agents);

- the automatic management of concurrency among tasks being pursued in parallel (intentions in the BDI terminology);
JACK Computation

JACK follows the standard BDI architectural approach, in that an agent
"in reaction to an event, for instance a change in the environment or its own beliefs, adopts an appropriate plan as one of its intentions"

As usual with BDI architectures, plans are precompiled procedures that each have a set of conditions that are used to determine their applicability.

The agent executes the steps of the plans that it has adopted as intentions until further deliberation is required; this may happen because of new events or the failure or successful conclusion of existing intentions.

A step in a plan can consist of
- adding a desire to achieve a certain objective to the agent itself,
- changing the agent’s beliefs,
- interacting with other agents, and
- any other atomic action on the agent’s own state or the external world.

JACK: Summary

- Good in that it is an extension to a standard (and popular) language, i.e. Java
- On initial examination, programming an agent appears to be quite complex
- The language utilises the names of concepts from rational agent theory, without ever implementing the semantics of these concepts.
- This is because it implements the BDI architecture, which does not have a strong link to BDI theory.
We now consider the work by Shoham who first identified the development of agent-based systems as being different to the development of object-based systems.

He introduced the term "Agent Oriented Programming" (AOP) and claimed that it represented a "new programming paradigm, based on a societal view of computation".

AOP embodies a notion of agency based upon concepts from rational agents.

Thus, in AOP, agents are directly programmed in terms of concepts such as their beliefs, commitments, and intentions.

Shoham suggested that a complete AOP system will have 3 components:
1. a logic for specifying agents and describing their mental states;
2. an interpreted programming language for programming agents;
3. an 'agentification' process, for converting 'neutral applications' (e.g., databases) into agents.

Results only reported on first two components.

Relationship between (1) and (2) was to be provided by an appropriate semantics.

We will skip over the first AOP language, AGENT0, and briefly consider a more refined language, called PLACA.

PLACA was a refined AOP implementation was developed by Thomas in 1993.

This Planning Communicating Agents (PLACA) language was intended to address drawbacks in earlier AOP languages, but was still based upon the concept of programming an agent in terms of mental change rules.
Example: PLACA rule

An example mental change rule:

```prolog
(((self ?agent REQUEST (?t (xeroxed ?x)))))
(AND (CAN-ACHIEVE (?t xeroxed ?x)))
(NOT (BEL (*now* (manager ?agent))))
(NOT (BEL (*now* shelving)))
((ADOPT (INTEND (5pm (xeroxed ?x))))))
((?agent self INFORM
   (*now* (INTEND (5pm (xeroxed ?x))))))
```

Paraphrased as:

if someone asks you to xerox something, and you can, and you don’t believe that they’re a manager, or that you’re supposed to be shelving books, then

- adopt the intention to xerox it by 5pm, and
- inform other agents of your newly adopted intention.

AOP: Summary

While AOP was a significant motivating breakthrough, the approach had problems.

Most notable amongst these was that there appears to be no formal link between linguistic notions such as ‘BEL’ and ‘INTEND’ and the appropriate behaviour for such operators.

As we have seen, it is common that programming languages utilise the names of concepts from rational agent theory, without ever implementing the semantics of these concepts.
Programmimg with Rational Agents

Executable Logics

Since AOP and JACK use the language of rational agency, but do not obviously follow the logical semantics for the agent concepts, how might we become more confident that our language actually implements the required behaviours?

One approach is to attempt to directly execute the logics that provide the semantics for our agent theories — in this way, we can be sure that the required behaviour is being exhibited.

Thus, we now animate an agent specification by directly executing the specification.

Here, execution of a formula, $\varphi$, of a logic, $L$, is taken to mean constructing a model, $M$, for $\varphi$, i.e. $M \models_L \varphi$.

Single Agent Execution

We begin by executing the basic temporal logic described earlier and then extend the logic in various ways.

1. Transform the temporal specification into SNF.
2. From the initial constraints, forward chain through the set of temporal rules representing the agent.
3. Constrain the execution by attempting to satisfy goals, such as $\Diamond g$ (i.e. $g$ eventually becomes true).

Basic strategy is to attempt to satisfy the oldest outstanding eventualities first and keep a record of the others, retrying them as execution proceeds.
Example: Temporal Execution

Imagine a 'car' agent which can \texttt{go}, \texttt{turn} and \texttt{stop}, but can also run out of fuel (empty) and overheat.

The agent's internal definition might be given by a temporal logic specification in SNP, for example,

\begin{align*}
\text{start} & \Rightarrow \neg \text{moving} \\
\text{go} & \Rightarrow \Diamond \text{moving} \\
(\text{moving} \wedge \text{go}) & \Rightarrow \Box (\text{overheat} \lor \text{empty})
\end{align*}

The car agent's behaviour is implemented by forward-chaining through these formulae.

- Thus, \text{moving} is false at the beginning of time.
- Whenever \text{go} is true, a commitment to eventually make \text{moving} true is given.
- Whenever both \text{go} and \text{moving} are true, then either \text{overheat} or \text{empty} will be made true in the next moment in time.

Introducing Deliberation

In the basic execution, there is a fixed strategy for implementing eventualities (e.g. \texttt{\Diamond go}), namely to attempt the oldest outstanding eventuality first.

We can extend this approach to allow the user to define the strategy/algorithm for deciding which eventualities to attempt first/next.

Note that, this is related to the way that deliberation occurs in BDI systems.

In BDI systems, desires represent statements/goals that the system must eventually satisfy; intentions represent statements/goals that the system is actively trying to satisfy.

If we consider both desires and intentions as eventualities, then the choice of which eventuality to attempt next (partially) captures the BDI notion of deliberation.
Example: Priority Functions

Desires = [be_famous, sleep, eat_lunch, make_lunch]

Standard approach would execute these oldest-first.

A first priority function might order these into most important:
Intentions = [be_famous, eat_lunch, sleep, make_lunch]

A second priority function might decide what we can't do at the moment (e.g. because we don't have a plan of how to achieve it) and what we need to do in order to make more important desires true, giving

Attempt = [make_lunch, eat_lunch, sleep, be_famous]

Resource-Bounded Reasoning

Aim: to allow agents to carry out resource-bounded reasoning about beliefs.

So, extend logical basis by adding a multi-context logic of belief, giving a temporal logic of bounded belief (TLBB).

Add a new 'B_i' modality, representing belief, and model the possible nesting of belief operators using multiple contexts, e.g.

Formulae inside one 'B_i' operator occur at the first level, those inside two 'B_i' operators occur at the second level, etc.

Key aspect of such logics is that a bound on the 'depth' of reasoning allowed can be specified.
**Executing TLBB**

Can extend the normal form (SNF$_{BB}$) to

\[
\text{start} \Rightarrow \bigvee_{b=1}^{r} l_b \quad \text{(an initial rule)}
\]

\[
\bigwedge_{a=1}^{g} k_a \Rightarrow \bigcirc \left[ \bigvee_{b=1}^{r} l_b \right] \quad \text{(a step rule)}
\]

\[
\bigwedge_{a=1}^{g} k_a \Rightarrow \Diamond l \quad \text{(a sometime rule)}
\]

\[
\bigwedge_{a=1}^{g} k_a \Rightarrow B_i \left[ \bigvee_{b=1}^{r} l_b \right] \quad \text{(a belief rule)}
\]

Execution through belief rules now also involves checking the allowed depth of reasoning.

**Resource-Bounded Deliberative Agents**

Here, SNF$_{BB}$ formulae are executed as before, with belief rules being used to explore finite (and depth-bounded) models of belief at each state.
Dynamically Bounded Reasoning

Recent work involves allowing the bound on reasoning to change dynamically.

A 'typical' example:

- A spacecraft agent has beliefs about planet surface topology and its own position/orientation
  \[ B_{sat}(X_{coord}, Y_{coord}, Feature), B_{height}(H) . \]
- Reasoning required when landing
- 'danger' if near mountain
  \[ \left[ \text{sensor}(X, Y, \text{mountain}) \land \text{coordinates}(Z, W) \land \text{near}(X, Z) \land \text{near}(Y, W) \right] \Rightarrow \text{danger} \]
- 'vital' if fuel level is low
  \[ \left[ \text{fuel}_{remaining}(F) \land (F < 5) \right] \Rightarrow \text{vital} . \]
- Change reasoning bounds accordingly
  \[ \left[ \text{bound}(X) \land \text{danger} \right] \Rightarrow \Diamond \text{bound}(X/2) \]
  \[ \left[ \text{bound}(X) \land \neg \text{danger} \land \text{vital} \right] \Rightarrow \Diamond \text{bound}(X + 2) \]

Executable Logics: Summary

- Potentially very powerful and flexible
- However, they haven't been applied/distributed widely
- Adding more features of rational agents might make the logics too complex.
References

JACK:


Agent-Oriented Programming:
Shoham — “Agent-Oriented Programming”. Artificial Intelligence 60(1), 1993

Thomas — “The PLACA Agent Programming Language”. In Intelligent Agents, Springer-Verlag, 1995

Executable Logics:

DISCUSSION

Rapporteur: Dr B N Rossiter

Professor Jones asked whether the resolution rules work because the modalities employed are essentially weak and wondered whether tasks like counting could be simulated by linking modalities. Professor Fisher said this could be done if there were no interactions between dimensions. Dr Stroud asked whether JACK follows the axioms. Professor Fisher said that it provides some useful classes but does not follow BDI rules. Professor Fisher confirmed, in response to Professor Sloman, that it is the responsibility of the programmer to ensure that JACK conforms to BDI. He also confirmed, in response to Professor Henderson, that BDI could be implemented in Java. Professor de Marneffe thought that sophisticated ways for running resource-bounded deliberative agents were needed, analogous to those used in Prolog query execution. Professor Fisher thought we were not at this optimization stage yet. Professor Jones asked where probabilities are to be found in the approach. Professor Fisher considered that they resided in thermodynamics with his formalization of multi-agent systems being an analogy of a thermodynamic system. Dr Holt thought that the emerging properties of such a system are very ill-conditioned with slight changes to a condition giving very different emerging properties. Professor Fisher agreed. He also agreed with Professor Sloman who thought that as agents get sophisticated (as in the multi-agent context), they are uncontrollable and to be useful must be simple in their autonomy.