The Agent Programming Language 3APL

Koen Hindriks
Utrecht University
email: koenh@cs.uu.nl
homepage: www.cs.uu.nl/people/koenh

The Metaphor of Intelligent Agents

A New Paradigm: Agent Oriented Programming

view programs as intelligent agents

Taking a Design Stance: the basic idea is to:

- design, analyse, understand, and reason about computational systems as
- systems having a mental state consisting of goals and beliefs, where
- traditional programs are viewed as goals or plans of the agent,
- traditional states are viewed as the beliefs of the agent.

Associating a physical state with agents is less useful for designing software agents; however, it might be useful for robot applications.

Purpose: managing complexity of software by means of abstraction.

Programs as Personal Agents

The metaphor of intelligent agents is a *natural* one, in particular for applications where

agents act on our behalf

Viewing programs as Personal Agents introduces a powerful metaphor to deal with User-Oriented Issues:

- a personal agent's goal is to support the goals of its user,
- metaphor seems to support design of natural user-agent interface,
- issues like autonomy, trust, etc arise.

A Programming Language for Building Agents

To support this style of programming, we propose
the agent programming language 3APL

From a software engineering point of view, we think an agent programming language is the most natural approach to build agents.

In contrast:

- Formal Logic for Agents for *specification only*: Not clear how to refine such specifcations to known programming languages like Java, etc.,
- Agent Architectures for building agent systems: Result in complicated systems with many features which make it hard to program agents

Our Approach

Our approach to agent programming is to use as much results as possible from existing paradigms and the area of programming language semantics

The agent language 3APL is a combination of features of Logic and Imperative Programming

We use Transition Systems to specify formal *operational* semantics.

Ongoing research:
Denotational Semantics and Programming Logic for the programming language 3APL.

A Definition of Intelligent Agents

A Symbolic Intelligent Agent is defined by:

- a complex mental state incorporating:
  - beliefs,
  - goals, and
  - practical reasoning rules,
- a set of mechanisms to manipulate this state:
  - to execute goals, i.e. perform belief updates, and
  - to apply rules, i.e. perform goal updates,
- a set of capabilities, i.e. basic actions, which define the goals the agent can achieve.

The Agent Programming Language 3APL Knowledge Representation

First order logic as the knowledge representation language:

- Beliefs are first order formula.
  Examples:
  $meet(MeetId, Time, Length, Loc)$,
  $\forall\, Id, T, L, Loc(meet(Id, T, L, Loc) \rightarrow 0:00 < L < 8:00)$.

- $\models$ denotes the usual entailment relation,

- A substitution is a finite set of constraints $x = t$

The Agent Programming Language
3APL
Goals

Goals are imperative-like programs.

- Basic Actions:
  Example: ins$(meet(talk, 11:00, 1:00, LP266))$,

- Tests:
  Example: $meet(talk, Time, Length, LP266)?$,

- Sequential Composition:
  Example:
  $meet(talk, Time, Length, LP266)?;$
  ins$(meet(lunch, Time{+}Length, 1:00, cafeteria))$,

- Nondeterministic Choice: $\pi_1 + \pi_2$,

- Parallel Composition: $\pi_1 \| \pi_2$

The Agent Programming Language 3APL Rules and Agents

Simple Practical Reasoning Rules:

- Rules are guarded clauses of the form:
  $p(\vec{t}) \leftarrow \varphi \mid \pi$

  Example:
  $schedmeet(Id, Time, Length, Loc) \leftarrow \text{true} \mid \text{ins}(meet(Id, Time, Length, Loc))$

An Intelligent Agent is a tuple $\langle a, \pi, \sigma, \Gamma \rangle$ where

- a is the name of the agent,
- π is the agent's goal,
- σ is the agent's belief base, and
- Γ is a set of practical reasoning rules

Operational Semantics: Transition Systems

A 3APL Configuration is a triple:

$\langle \pi, \sigma, \theta \rangle$

where

- π is a goal,
- σ is a belief base,
- θ is a substitution

A Transition Rule is of the form:

$$\frac{\langle \pi_1, \sigma_1, \theta_1 \rangle \longrightarrow \langle \pi'_1, \sigma'_1, \theta'_1 \rangle, \ldots \langle \pi_n, \sigma_n, \theta_n \rangle \longrightarrow \langle \pi'_n, \sigma'_n, \theta'_n \rangle}{\langle \pi, \sigma, \theta \rangle \longrightarrow \pi', \sigma' \rangle}$$

The Transition Relation $\longrightarrow$ specifies *possible computation steps*.

$\langle \pi, \sigma, \theta \rangle \longrightarrow \langle \pi', \sigma', \theta' \rangle$ means that

- π can perform *one* computation step, which updates
- σ to σ',
- θ to θ',
- and transforms π to π'.

A Transition System is a set of transition rules.

Basic Actions

The semantics of Basic Actions is given by a transition function $\mathcal{T} : Act \times \mathcal{L} \to \mathcal{L}$.

- Basic Actions are Belief Updates
- "Reasonable" Transition Functions:
  For example, $\mathcal{T}(a, p \wedge q) = \mathcal{T}(a, q \wedge p)$
- Only defined for Closed Actions;
  What does it mean to execute
  ins$(meet(talk, Time, 1:00, LP266))$?

Definition 1. (basic actions)

$$\frac{\mathcal{T}(a\theta, \sigma) = \sigma'}{\langle a, \sigma, \theta\rangle \longrightarrow \langle E, \sigma', \theta\rangle}$$

Example 2.

$\mathcal{T}(\mathsf{del}(meet(talk, 10:00, 1:00, LP266)),$
$\quad meet(talk, 10:00, 1:00, LP266)) = \mathsf{true}$.

Then:

$\langle \mathsf{del}(meet(talk, 10:00, 1:00, LP266)),$
$\quad meet(talk, 10:00, 1:00, LP266), \theta\rangle$
$\longrightarrow \langle E, \mathsf{true}, \theta\rangle$

Tests

The semantics of Tests is provided by an
entailment relation $\models$

- Tests are Checks on the Belief Base, and Updates on Substitutions
- Tests initialise variables by retrieving values from the belief base.

Definition 3. (tests)

$$\frac{\sigma \models \phi\theta\gamma}{\langle \phi?, \sigma, \theta \rangle \longrightarrow \langle E, \sigma, \theta\gamma \rangle}$$

Example 4.

$\langle meet(talk, 10:00, Len, Loc)?, meet(talk, 10:00, 1:00, LP266), \varnothing \rangle$
$\longrightarrow$
$\langle E, meet(talk, 10:00, 1:00, LP266), \{Len = 1:00, Loc = LP266\} \rangle$

Sequential Composition

- Implicit Scoping in 3APL (compare Logic Programming),
- First Occurrence of Variable in goal implicitly binds later occurrences.

Definition 5. (sequential composition)

$$\frac{\langle \pi_1, \sigma, \theta \rangle \longrightarrow \langle \pi_1', \sigma', \theta' \rangle}{\langle \pi_1;\ \pi_2, \sigma, \theta \rangle \longrightarrow \langle \pi_1';\ \pi_2, \sigma', \theta' \rangle}$$

Example 6.

$\langle meet(talk, 10:00, Len, Loc)?;\ \mathsf{del}(meet(talk, 10:00, Len, Loc)),$
$meet(talk, 10:00, 1:00, LP266), \varnothing \rangle$
$\longrightarrow$
$\langle \mathsf{del}(meet(talk, 10:00, Len, Loc)),$
$meet(talk, 10:00, 1:00, LP266), \{Len = 1:00, Loc = LP266\} \rangle$
$\longrightarrow$
$\langle E, \mathsf{true}, \{Len = 1:00, Loc = LP266\} \rangle$

Nondeterministic Choice and Parallel Composition

Definition 7. (nondeterministic choice)

$$\frac{\langle \pi_1, \sigma, \theta \rangle \longrightarrow \langle \pi_1', \sigma', \theta' \rangle}{\langle \pi_1 + \pi_2, \sigma, \theta \rangle \longrightarrow \langle \pi_1', \sigma', \theta' \rangle}$$

Parallel composition is modeled by interleaving

Definition 8. (parallel composition)

$$\frac{\langle \pi_1, \sigma, \theta \rangle \longrightarrow \langle \pi_1', \sigma', \theta' \rangle}{\langle \pi_1 \| \pi_2, \sigma, \theta \rangle \longrightarrow \langle \pi_1' \| \pi_2, \sigma', \theta' \rangle}$$

Rule Application

Plan rules of the form $p(\vec{t}) \leftarrow \varphi \mid \pi$ provide for an abstraction mechanism which is equivalent to recursive procedures in imperative programming.

Definition 9. (application of rule)
Let η be a substitution such that $p(\vec{t})\theta = p(\vec{t'})\eta$.

$$\frac{\sigma \models \phi\eta\gamma}{\langle p(\vec{t}), \sigma, \theta\rangle \longrightarrow \langle \pi, \sigma, \theta\gamma\rangle}$$

where $p(\vec{t'}) \leftarrow \varphi \mid \pi$ is a variant of a rule of the agent

Example 10.

Plan rule:
$schedmeet(Id, Time, Len, Loc) \leftarrow \mathsf{true} \mid$
$\mathsf{ins}(meet(Id, Time, Len, Loc)$
Rule application:
$\langle schedmeet(talk, 11:00, 1:00, LP266), \mathsf{true}, \varnothing\rangle$
$\longrightarrow$
$\langle \mathsf{ins}(meet(talk, 11:00, 1:00, LP266), \mathsf{true}, \varnothing\rangle$

Multi-Agent Systems

A Multi-Agent System is a finite set of distinct agents $\mathcal{A}_1, \ldots, \mathcal{A}_n$.

- the execution of a multi-agent system *without communication* is just the execution of each of these agents in parallel,
- we extend the single agent language 3APL with two pairs of synchronous communication primitives,
- tell/ask for information exchange,
- req/offer for exchange of a request,
- transition style semantics (cf. Semantics of Communicating Agents based on Deduction and Abduction)

Exchange of Information

Focus on the receiving agent:
Aim: capture the successful processing of a message by the receiving agent.

Example: Consider Agent A and B:
A Answers: In the room we met yesterday. *B Asks*: Where do we meet?

- agent A needs to compute an answer from the information provided to be able to use it to its advantage.

How does agent A compute the answer?
The appropriate reasoning process involved is that of deduction.

Semantics of Information Exchange

Definition 11. *(transition rule for* tell*)*

$$\frac{\varphi \text{ is closed}}{\langle \mathsf{tell}(b,\varphi),\sigma\rangle_V \xrightarrow{b!_i\varphi}_{\varnothing} \langle E,\sigma\rangle}$$

Definition 12. *(*ask*)*
Let θ be a substitution restricted to the free variables of ψ.

$$\frac{\psi\theta \text{ is closed}}{\langle \mathsf{ask}(a,\psi),\sigma\rangle_V \xrightarrow{a?_i\psi\theta}_{\theta} \langle E,\sigma\rangle}$$

Definition 13. *(exchange of information)*
Let $A = \langle a,\Pi_a,\sigma_a,\Gamma_a\rangle$ and $B = \langle b,\Pi_b,\sigma_b,\Gamma_b\rangle$ be two agents such that $a \neq b$, and let $\mathcal{M}$ be a (possibly empty) multi-agent system such that $A \notin \mathcal{M}, B \notin \mathcal{M}$.

$$\frac{A \xrightarrow{b!_i\varphi} A'\ ,\ B \xrightarrow{a?_i\psi} B' \text{ and } \sigma_b \cup \varphi \models \psi}{\mathcal{M}, A, B \longrightarrow \mathcal{M}, A', B'}$$

Requests and Offers

Focus on the receiving agent:
Aim: capture the successful processing of a message by the receiving agent.

Example: Consider Agent A and B:
A Requests: Shall we meet somewhere 2pm?
B Offers: Let's meet in my office.

- agent B needs to compute a *specific* reply to the request to be able to satisfy it.

How does agent B compute its reply?
The appropriate reasoning process involved is that of abduction.

Abductive Semantics for Request/Offer

Informally:
Abduction is reasoning from:
- effect to a cause.

Formally:
Given a number of 'background' beliefs σ and observation φ the task is:
- to find a 'cause' ψ such that $\sigma, \psi \models \varphi$,
- which is consistent with σ, ie $\sigma \not\models \neg\psi$.

Usually, a set of possible hypotheses H are provided to choose from.

The basic idea:
Abduction can also be used to compute a proposal which would satisfy the request.

Example:
A Requests: req$(B, \exists\, Loc(meet(14:00, Loc)))$,
B Offers: offer$(A, meet(14:00, Location))$

The offering agent needs to compute a value for the free variable $Location$.

Meeting Scheduling Example

A Multi-Stage Negotiation Protocol for meeting scheduling (Sen/Durfee)

Two-agent Case:

- Agent A and Agent B attempt to schedule a meeting,
- Both agents have agreed upon the type, length and location of the meeting,
- Task: negotiate a meeting time which suits both agents.

Simple Solution:

- Settle for first time (given some initial time) that suits both agents,
- By proposing and counterproposing meeting times.

Meeting Scheduling (2)

Constraints on meetings:

Constraint 1: Meeting Identifiers refer to unique meetings:

$$(meet(MeetId, T_1, Len_1, Loc_1) \wedge \\ meet(MeetId, T_2, Len_2, Loc_2)) \\ \rightarrow (T_1 = T_2 \wedge Len_1 = Len_2 \wedge Loc_1 = Loc_2)$$

Constraint 2: There are no overlapping meetings:

$$(meet(MeetId1, T_1, Len_1, Loc_1) \wedge \\ meet(MeetId2, T_2, Len_2, Loc_2) \wedge \\ T_1 \leq T_2 < T_1 + Len_1 \wedge MeetId1 \neq MeetId2) \\ \rightarrow \mathsf{false}$$

Meeting Scheduling (3)

Earliest Possible Meeting Time:

$$
\begin{aligned}
&epmeet(MeetId, PosTime, Len, Loc, InitT) \leftrightarrow\\
&(meet(MeetId, PosTime, Len, Loc) \wedge\\
&PosTime \geq InitT \wedge\\
&(\forall\, T' \cdot InitT \leq T' < PosTime \rightarrow\\
&\qquad \neg meet(MeetId, T', Len, Loc)))
\end{aligned}
$$

Meeting Scheduling (4)

Host Agent A invites Agent B by requesting to meet at earliest possible time after $InitT$:

$\texttt{invite}(Invitee, MeetId, InitT, Len, Loc)$

$\leftarrow$ true $|$

req$(Invitee, \exists\, T1 \cdot epmeet(MeetId, T1, Len, Loc, InitT))$;

offer$(Invitee, meet(MeetId, T', Len, Loc))$;

$\texttt{IF}\ InitT = T'$

$\texttt{THEN}$ tell$(Invitee, confirm(MeetId, InitT))$

$\texttt{ELSE}\ \texttt{invite}(Invitee, MeetId, T', Len, Loc)$

Host Agent B replies to Agent A by offering to meet at time $OfferT$:

$\texttt{reply}(Host) \leftarrow$ true $|$

begin

offer$(Host, meet(MeetId, OfferT, Len, Loc))$;

req$(Host, \exists\, U1 \cdot epmeet(MeetId, U1, Len, Loc, OfferT))$;

$\texttt{reply}(Host)$

end$+$

ask$(Host, confirm(MeetId, MeetT))$

Conclusions

- deduction provides appropriate semantics for information exchange,
- abduction provides appropriate semantics for making requests and offers,
- simple and formal semantics for communication,
- communication language integrated into agent programming language,
- agent programming language provides for expressive primitives

For more extensive discussion see papers on my homepage: www.cs.uu.nl/~koenh