Recommended reading

Communicating agents in AgentSpeak

What we learned so far:
- Writing plans of different types
- Catching plan failures with special plan types
- Modifying the environment code (Java)
- Sending (simple) messages

At least, the last point deserves more attention. How does communication in AgentSpeak actually work?

The checkMail method

Remember this part of the Jason reasoning cycle?

Structure of received messages:

\[ \langle \text{sender}, \text{iloc\_force}, \text{content} \rangle \]

Available performatives

Pre-defined internal action for sending messages:

\[ .\text{send}(	ext{receiver}, \text{iloc\_force}, \text{prop\_content}) \]

iloc\_force can be:
- **tell**: s intends r to believe (that s believes) the literal to be true;
- **untell**: s intends r not to believe (that s believes) the literal to be true;
- **achieve**: s requests r to try and achieve a state where the literal is true (i.e., goal delegation);
- **unachieve**: s requests r to drop the goal of achieving a state where the literal is true;
- ...

(continued on next slide)
Available performatives cont.

#illoc_force# can be:

- **askOne**: s wants to know if literal is true for r (i.e., if there is an answer that makes the content a logical consequence of r's belief base, by appropriate substitution of variables);
- **askAll**: s wants all of r’s answers to a question;
- **tellHow**: s informs r of a plan (s’s know-how);
- **untellHow**: s requests that r disregard a certain plan (i.e., to delete that plan from the plan library);
- **askHow**: s wants all of r’s plans that are relevant for the triggering event in the message content.

Possible variations of sending messages:

1. \( \text{send}(r, \text{tell}, \text{open}(l\_door)) \) ⇒ r believes \( \text{open}(l\_door) \) [source(s)]
2. \( \text{send}([r1, r2], \text{tell}, \text{open}(l\_door)) \) ⇒ both r1 and r2 believe \( \text{open}(l\_door) \) [source(s)]
3. \( \text{send}(r, \text{tell}, [\text{open}(l\_door), \text{open}(r\_door)]) \) ⇒ r1 believes both \( \text{open}(l\_door) \) [source(s)] and \( \text{open}(r\_door) \) [source(s)]
4. \( \text{send}(r, \text{untell}, \text{open}(l\_door)) \) ⇒ updates in r beliefs like \( \text{open}(l\_door) \) [source(s), source(percept)] to \( \text{open}(l\_door) \) [source(percept)] or deletes belief

Tell message example

(Bordini et al., 2007, p. 121)

Goal delegation

For **goal delegation** the performatives are:

- \( \text{send}(r, \text{achieve}, \text{open}(l\_door)) \) ⇒ the event \( \langle +!\text{open}(l\_door)[\text{source(s)}], T \rangle \) is created in r’s set of events and r will **intend** a plan, if relevant and applicable
- \( \text{send}(r, \text{unachieve}, \text{open}(l\_door)) \) ⇒ the internal action \( \text{drop_desire(open}(l\_door)) \) is executed in r without generating a plan failure event

<table>
<thead>
<tr>
<th>Cycle</th>
<th>s actions</th>
<th>r beliefs</th>
<th>r events</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.send(r, tell, open(left_door))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>open(left_door)</td>
<td>+open(left_door) [source(s)]</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.send(r, untell, open(left_door))</td>
<td>open(left_door) [source(s)]</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-open(left_door) [source(s)]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6.1 Changes in the receiver's state for (un)tell messages.
Information seeking

For information seeking the performatives are:

- ▶ .send(r, askOne, open(Door), Reply) ⇒ s’s intention that executed this internal action is suspended until reply from r, which receives ⟨s, askOne, open(Door)⟩ and (if accepted) a reply (e.g., with open(l_door)) is sent back.
- ▶ .send(r, askOne, open(Door)) ⇒ like above but asynchronous, i.e. s’s intention is not suspended
- ▶ .send(r, askAll, open(Door, Reply)) ⇒ like the first one but Reply is a list with all possible answers, which may be empty

Examples of ask protocols (p. 125):

<table>
<thead>
<tr>
<th>Cycle</th>
<th>s actions / unifier / events</th>
<th>r actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.send(r, askOne, open(Door), Reply)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.send(s, tell(open(left_door)))</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reply ⇒ open(left_door)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.send(r, askOne, closed(Door), Reply)</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.send(s, tell(false))</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Reply ⇒ false</td>
<td></td>
</tr>
</tbody>
</table>

Know-how related

For know-how related information the performatives are:

- ▶ .send(r, tellHow, "@pOD +!open(Door)
  not locked(Door) ← turn_handle(Door); push(Door);
  ?open(Door).") ⇒ the plan @pOD will be added to r’s plan library
- ▶ .send(r, tellHow, ["+!...","+!..."]) ⇒ each member of the list of strings representing plans is added to r’s plan library
- ▶ .send(r, untellHow, "@pOD"): ⇒ if plan with that label exists in r’s plan library, it is removed
- ▶ .send(r, askHow, "+!open(Door)"): string must match triggering event of a plan in r’s plan library, then list with relevant plans is sent back

6b.2 Summary

B. Nebel, C. Becker-Asano, S. Wöl (Universität Freiburg) Multiagent Systems June 6, 2014 15 / 1
Summary

In summary, the following performatives can be used for:

- Information exchange:
  - tell
  - untell
- Goal delegation:
  - achieve
  - unachieve
- Information seeking:
  - askOne
  - askAll
- Exchanging means (plans):
  - tellHow
  - untellHow
  - askHow