2b.1 Outline

AgentSpeak:
- Beliefs
- Goals
- Plans
  - Triggering events
  - Context
  - Body
AgentSpeak – background

- AgentSpeak (Rao, 1996) is an agent programming language whose semantics implements the functionalities of the Procedural Reasoning System
- Based on the BDI architecture (as is the PRS)
- AgentSpeak agents are reactive (in that they do not terminate) planning agents
- The Jason agent programming system (Bordini et al., 2007) uses an extension of the AgentSpeak language

2b.2 AgentSpeak

- Main notions
  - Beliefs
    - Belief change
    - Belief negation
    - Rules
  - Goals
    - New goals
  - Plans
    - Events
    - Context
    - Body
  - Acknowledgments

Jason & AgentSpeak

- Beliefs: represent the information available to an agent (e.g. about the environment or other agents)
- Goals: represent states of affairs the agent wants to bring about
- Events: happen as a consequence to changes in the agent's beliefs or goals
- Plans: are recipes for action, representing the agent's know-how
- Intentions: plans instantiated to achieve some goal
Belief representation

Syntax
Beliefs are represented by annotated literals of first order logic

\[
\text{functor}(\text{term}_1, \ldots, \text{term}_n)[\text{annot}_1, \ldots, \text{annot}_m]
\]

Example belief base of agent Tom:
- \text{red(box1)}[source\text{(percept)}].
- \text{friend(bob,alice)}[source\text{(bob)}].
- \text{liar(alice)}[source\text{(self)}, source\text{(bob)}].
- \sim\text{liar(bob)}[source\text{(self)}].

Annotations & Information sources

The interpreter can be programmed to use annotations, which is mainly used to remember a belief’s source.

Annotation

\text{busy(john)}[\text{expires(autumn)}].

During the agent’s execution, beliefs can be updated with information coming from the following sources:

- **perception**: resulting from a continual sensing of the environment
- **communication**: resulting from another agent’s message
- **mental notes**: added by the agent to its own belief base during the execution of a plan (analogous to asserts in Prolog)

Changes in the belief base

The belief base changes by:

- **perception**: beliefs annotated with source\text{(percept)} are automatically updated accordingly to the perception of the agent
- **intention**: the operators + and - can be used to add and remove beliefs annotated with source\text{(self)}
- **communication**: when an agent receives a \text{tell} message, the content is a new belief annotated with the sender of the message

Examples:

- **belief update by intention**:
  +\text{liar(alice)} adds \text{liar(alice)}[source\text{(self)}]
  -\text{liar(john)} \text{removes lien(john)}[source\text{(self)}]

- **belief update by communication**:
  +\text{liar(alice)}[source\text{(bob)}] in Tom’s BB
  -\text{liar(alice)}[source\text{(bob)}] sent by Bob removes
  \text{liar(alice)}[source\text{(bob)}] from Tom’s BB

Predicates

Beliefs describe what the agent knows about its environment.
In their simplest form \Rightarrow logic programming facts.

Property

\text{tall(john)}.

Relationship

\text{likes(john,music)}.

These atoms are meant to express the agent’s belief and not the absolute truth!
Negation of beliefs

A belief can be negated in at least two senses:
- meaning that the agent does not believe B
- meaning that the agent believes the opposite of B

Therefore, AgentSpeak provides two types of negation:
- Negation as failure (not)
- Strong negation (∼)

This relies on the closed world assumption:
Anything that is neither known to be true, nor derivable from the known facts using the rules in the program, is assumed to be false.

Negation as failure ⇒ not operator is true, if the interpreter fails to derive its argument.

Negation as failure versus Strong negation

Negation as failure is simplistic in open environments. One cannot realistically assume that everything that one does not know positively is false!

Better solution is to distinguish:
1. what the agent believes to be true
2. what the agent believes to be false
3. what the agent believes nothing about

∼ operator is true if the agent has an explicit belief that its argument is false.
⇒ Strong negation

Rules

Used for theoretical reasoning ⇒ deriving knowledge from existing knowledge.

Rule syntax

<head> :- <body>

- head is one atom
- body can contain disjunctions (symbol: |) and conjunctions (symbol: &)

Examples:

likely_colour(C,B) :- colour(C,B) [source(S)] & (S==self | S==percept).
likely_colour(C,B) :- colour(C,B) [degOfCert(D1)] &
not (colour(_,B) [degOfCert(D2)] & D2 > D1) &
not ∼colour(C,B).

Goals in AgentSpeak

In general, what the agent wishes to bring about.
In AgentSpeak, two types of goals:
- achievement goals (some formula that the agent wants to make true)
- test goals (some formula whose truth the agent wants to check)

Achievement goal syntax

! operator applied to the atom describing the goal.

Test goal syntax

? operator applied to the atom describing the goal.
Achievement versus Test goals

If an agent has an **achievement goal**, it will try to **achieve a state of affairs** where it believes that the goal is true. For example:

!own(house).

If an agent has a **test goal**, it will try to **derive the goal** from its beliefs. For example:

?bank_balance(100).

Test goals are analogous to Prolog goals.

New goals

New goals can be added by:

- **intention**: the operators ! and ? can be used to add a new goal annotated with source(self)
- **communication**: when an agent receives an achieve message, the content is a new achievement goal annotated with the sender of the message

Examples:

- !write(book) adds a new achievement goal !write(book)[source(self)]
- ?publisher(P) adds a new test goal ?publisher(P)[source(self)]
- .send(tom,achieve,write(book)) sent by Bob adds new goal !write(book)[source(bob)] for Tom
- .send(tom,unachieve,write(book)) sent by Bob removes goal !write(book)[source(bob)] for Tom

Plan library

**Plans** are an agent’s means to achieve a **goal**. Plans are:

- initial plans defined by the programmer
- plans added dynamically and intentionally by
  - .add_plan
  - .remove_plan
- plans received from other agents by messages
  - tellHow
  - untellHow

They are composed of three parts:

- **triggering event**: the event the plan is meant to handle
- **context**: the circumstances in which the plan can be used
- **body**: the course of action to be used to handle the event if the context is believed true when the plan is being chosen

Plan syntax

<triggering event> : <context> <- <body>.

Events

For an agent it is necessary to balance **reactivity** and **proactiveness**:

- Goals support **proactiveness**
- For **reactivity**: agent needs to be aware of changes that occur in the agent itself and in the environment
- **Events** are generated in case of changes (addition, deletion) in:
  - the agent’s beliefs
  - the agent’s goals
  - The <triggering event> part of plans
**AgentSpeak Plans**

### Triggering events

Under which circumstances should a plan be considered?

⇒ Check the plan's **triggering** part.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>+l</td>
<td>Belief addition</td>
</tr>
<tr>
<td>-l</td>
<td>Belief deletion</td>
</tr>
<tr>
<td>+!l</td>
<td>Achievement goal addition</td>
</tr>
<tr>
<td>-!l</td>
<td>Achievement goal deletion</td>
</tr>
<tr>
<td>+?l</td>
<td>Test goal addition</td>
</tr>
<tr>
<td>-?l</td>
<td>Test goal deletion</td>
</tr>
</tbody>
</table>

*Table: Notation for types of triggering events*

### Plan context

A plan's context defines in what circumstances it is **applicable**.

Atoms and relational expressions combined with the operators:

- **not** (negation as failure)
- **&** (conjunction)
- **|** (disjunction)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Agent beliefs $l$ to be true</td>
</tr>
<tr>
<td>~l</td>
<td>Agent beliefs $l$ to be false</td>
</tr>
<tr>
<td>not l</td>
<td>Agent does not believe $l$ to be true</td>
</tr>
<tr>
<td>not ~l</td>
<td>Agent does not believe $l$ to be false</td>
</tr>
</tbody>
</table>

*Table: Plan context is typically a conjunction of these types of literals*

### Internal actions

- Executed internally rather than managed by the interpreter.
- E.g., to execute Java code on partial results.
- If they have side effects they are not always appropriate in the context (which should be used to check if a plan is applicable).
- May be used to interface with other software components, possibly ROS / ROCON.

### Defined atoms

Rules in the belief base can be used to make plan contexts more compact.

#### Context with defined atoms

If the belief base contains a rule

```prolog
can_afford(Something) :- price(Something,P) & bank_balance(B) & B > P.
```

then we can write

```prolog
+!buy(Something) :- not ~legal(Something) & can_afford(Something)
```

<- ...
A plan's body

Represents the course of action to be executed. Can contain six types of formulæ:
1. actions
2. achievement goals
3. test goals
4. mental notes
5. internal actions
6. expressions

Formulæ are separated by semicolons (;).

Actions

The actions that the agent can perform are known beforehand and represented symbolically. In a plan's body, a ground predicate represents an action, e.g.:

- right
- left
- rotate(left_arm,45)
- rotate_right_arm(90)

Internal actions: run internally within the agent, rather than change the environment (see later).

Achievement goals

Complex plans require more than simple actions to be executed. It is sometimes necessary to achieve intermediate goals for the plan to be completed.

⇒ a plan's body can also contain achievement goals.

Two categories (w.r.t. suspension of plans):
- with !at(home);call(john) being part of an agent's plan, the action call(john) is executed only after it got home
- with !!at(home);call(john), the action call(john) is executed as soon as the agent's goal to be at home is set

Test goals

Test goals are used to retrieve information during plan execution and they are identified in a plan's body by the ? operator.

Example:

?coords(Target,X,Y)

Lets the agent know the current coordinates of a target.

Difference with test goals in context:
- If a test goal fails in the plan's context, the plan is not executed;
- If a test goal fails in the plan's body, the plan fails.
AgentSpeak Plans

Mental notes

Mental notes are **beliefs added** with the `+` operator during plan execution. They automatically get a `source(self)` annotation.

Example 1

```
+ mowed(lawn, Today)
```

(“Today” must have been initialized earlier.)

Such beliefs can also be **removed** with the `-` operator.

Example 2

```
- current_targets(_)
```

Furthermore:

```
+-some_predicate(arg)
```

is an abbreviation for

```
-some_predicate(_);+some_predicate(arg)
```

Internal actions

Internal actions are:

- executed internally, rather than to change the environment.
- identified by the presence of a . (also used to separate library name from action name) in their name.

Example of an internal action

The agent might be using a path-finding library `pf`, which makes available an internal action `get_path` to find the shortest path from its current position to a point with coordinates `X` and `Y`:

```
pf.get_path(X, Y, P)
```

Standard internal actions exist (with empty library name), e.g.:

- `.send`
- `.print`

Expressions

Most Prolog built-in expressions are available:

- `T1==T2` iff `T1` and `T2` are identical terms
- `T1\==T2` iff `T1` and `T2` are not identical terms
- `T1=T2` (tries to) unify `T1` and `T2`
- `=..` behaves slightly differently to support annotations:
  
  `p(b, c) [a1, a2] =.. [p, [b, c], [a1, a2]]`

Acknowledgments/Resources

The lecture slides are based on slides developed by Dr. Marco Alberti, Universidade Nova de Lisboa.