



BDI Agents and AgentSpeak(L)

Romelia Plesa
PhD Candidate
SITE, University of Ottawa



Presentation Outline

- BDI
- AgentSpeak(L) – Basic Notions
- AgentSpeak(L) – Syntax
- AgentSpeak(L) – Informal Semantics
- AgentSpeak(L) – Example
- Jason

BDI Agents

- Systems that are situated in a changing environment
- Receive continuous perceptual input
- Take actions to affect their environment

From the various options and alternatives available to it at a certain moment in time, the agent needs to select the appropriate actions or procedures to execute.

The ***selection function*** should enable the system to achieve its objectives, given

- the computational resources available to the system
- the characteristics of the environment in which the system is situated.

BDI Agents

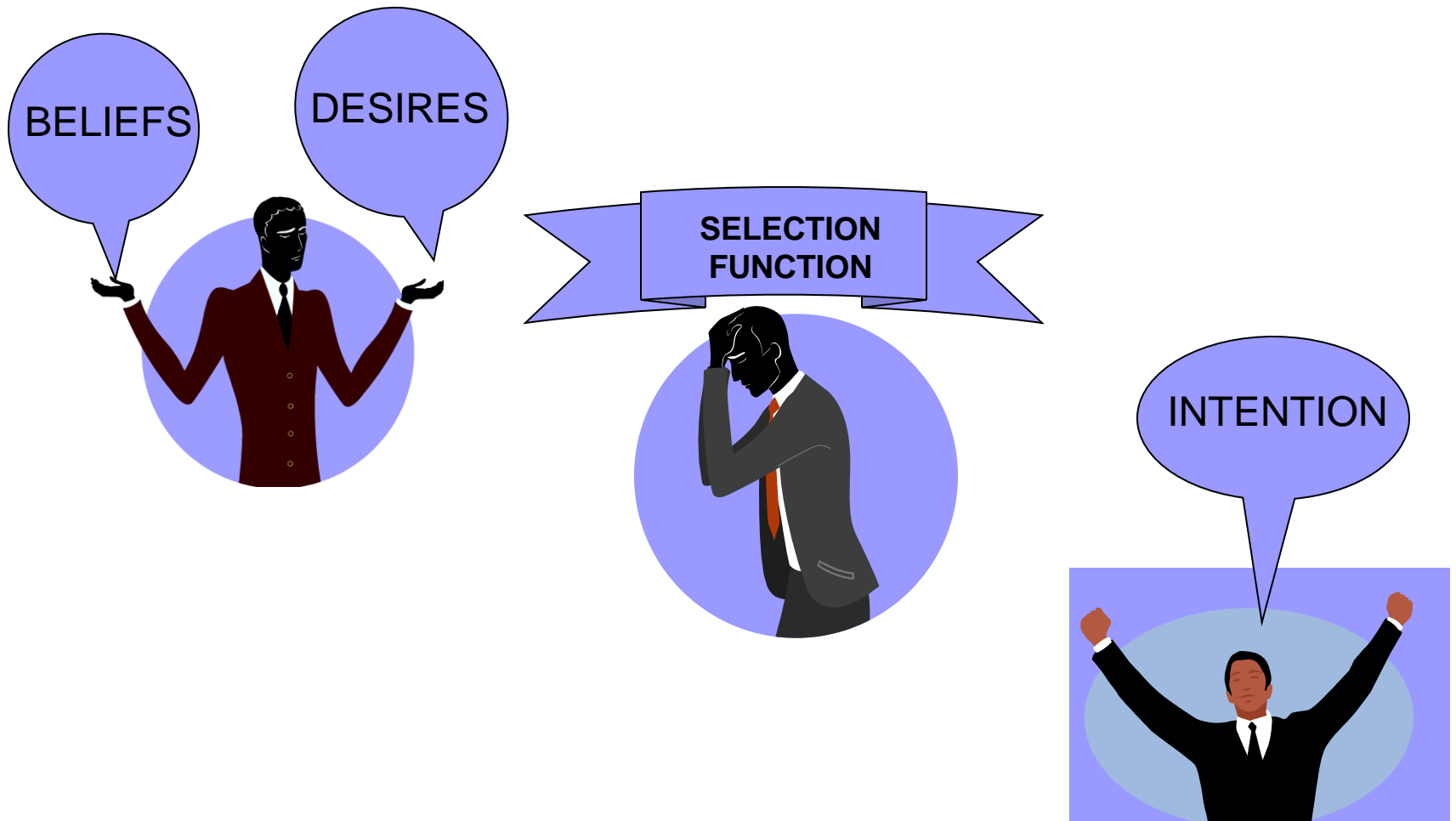
- two types of input data required for the selection function:
- **Beliefs:**
 - represent the characteristics of the environment
 - are updated appropriately after each sensing action.
 - can be viewed as the *informative* component of the system.
- **Desires**
 - contain the information about the objectives to be accomplished, the priorities and payoffs associated with the various objectives
 - can be thought as representing the *motivational* state of the system.

BDI Agents

■ Intentions

- represent the currently chosen course of action (the output of the most recent call to the selection function)
- capture the *deliberative* component of the system.

BDI Agents



AgentSpeak(L)

- attempt to bridge the gap between theory and practice
- a model that shows a one-to-one correspondence between the model theory, proof theory and the abstract interpreter.
 - natural extension of logic programming for the BDI agent architecture
 - provides an elegant abstract framework for programming BDI agents.
 - based on a restricted first-order language with events and actions.
 - the behavior of the agent (i.e., its interaction with the environment) is dictated by the programs written in AgentSpeak(L).

AgentSpeak(L) and B-D-Is

- An agent's ***belief state*** is the current state of the agent, which is a model of itself, its environment, and other agents.
- The agent's ***desires*** are the states that the agent wants to bring about based on its external or internal stimuli.
- When an agent commits to a particular set of plans to achieve some goal, these partially instantiated plans are referred to as an ***intention*** associated with that goal. Thus, ***intentions*** are active plans that the agent adopts in an attempt to achieve its goals.

AgentSpeak(L) - Basic Notions

- The specification of an agent in AgentSpeak(L) consists of:
 - a set of base *beliefs*
 - facts in the logic programming sense
 - a set of *plans*.
 - context-sensitive, event-invoked recipes that allow hierarchical decomposition of goals as well as the execution of actions with the purpose of accomplishing a goal.

AgentSpeak(L) - Basic Notions

- ***belief atom***

- is a first-order predicate in the usual notation
- belief atoms or their negations are termed ***belief literals***.

AgentSpeak(L) - Basic Notions

- ***goal***

- is a state of the system, which the agent wants to achieve.

- two types of goals:

- ***achievement goals***

- predicates prefixed with the operator “!”
 - state that the agent wants to achieve a state of the world where the associated predicate is true.
 - in practice, these initiate the execution of *subplans*.

- ***test goals***

- predicates prefixed with the operator ‘?’
 - returns a unification for the associated predicate with one of the agent’s beliefs; it fails if no unification is found.

AgentSpeak(L) - Basic Notions

■ *triggering event*

- defines which events may initiate the execution of a plan.
- an *event* can be
 - internal, when a subgoal needs to be achieved
 - external, when generated from belief updates as a result of perceiving the environment.
- two types of triggering events:
 - related to the *addition* ('+') and *deletion* ('-') of attitudes (beliefs or goals).

AgentSpeak(L) - Basic Notions

■ *Plans*

- refer to the *basic actions* that an agent is able to perform on its environment.

$$p ::= te : ct \leftarrow h$$

Where:

- *te* - triggering event (denoting the purpose for that plan)
- *ct* - a conjunction of belief literals representing a context.
 - The context must be a logical consequence of that agent's current beliefs for the plan to be applicable.
- *h* - a sequence of basic actions or (sub)goals that the agent has to achieve (or test) when the plan, if applicable, is chosen for execution.

Triggering
event

Context

```
+concert (A,V) : likes (A) <-  
  !book_tickets (A,V) .
```

Achievement
goal added

```
+!book_tickets (A, V) :  
  ¬busy (phone)  
  <- call (V) ;  
  ... ;  
  !choose seats (A,V) .
```

Basic action

AgentSpeak(L) - Basic Notions

■ *Intentions*

- plans the agent has chosen for execution.
- Intentions are executed one step at a time.
- A step can
 - query or change the beliefs
 - perform actions on the external world
 - suspend the execution until a certain condition is met
 - submit new goals.
- The operations performed by a step may generate new events, which, in turn, may start new intentions.
- An intention succeeds when all its steps have been completed. It fails when certain conditions are not met or actions being performed report errors.

AgentSpeak(L) Syntax

```
ag ::= bs ps
bs ::= at1. ... atn. (n≥0)
at ::= P(t1, ... tn) (n≥0)
ps ::= p1 ... pn (n≥1)
p ::= te : ct <- h.
te ::= +at | -at | +g | -g
ct ::= true | l1 & ... & ln (n≥1)
h ::= true | f1 ; ... ; fn (n≥1)
l ::= at | not (at)
f ::= A(t1, ... tn) | g | u (n≥0)
g ::= !at | ?at
u ::= +at | -at
```


AgentSpeak(L)- Informal Semantic

- The interpreter for AgentSpeak(L) manages
 - a set of *events*
 - a set of *intentions*
 - three *selection functions*.

AgentSpeak(L)- Informal Semantic Events

- **Events**, which may start off the execution of plans that have relevant triggering events, can be:
 - **external**, when originating from perception of the agent's environment (i.e., addition and deletion of beliefs based on perception are external events). External events create new intentions.
 - **internal**, when generated from the agent's own execution of a plan (i.e., a subgoal in a plan generates an event of type "addition of achievement goal").

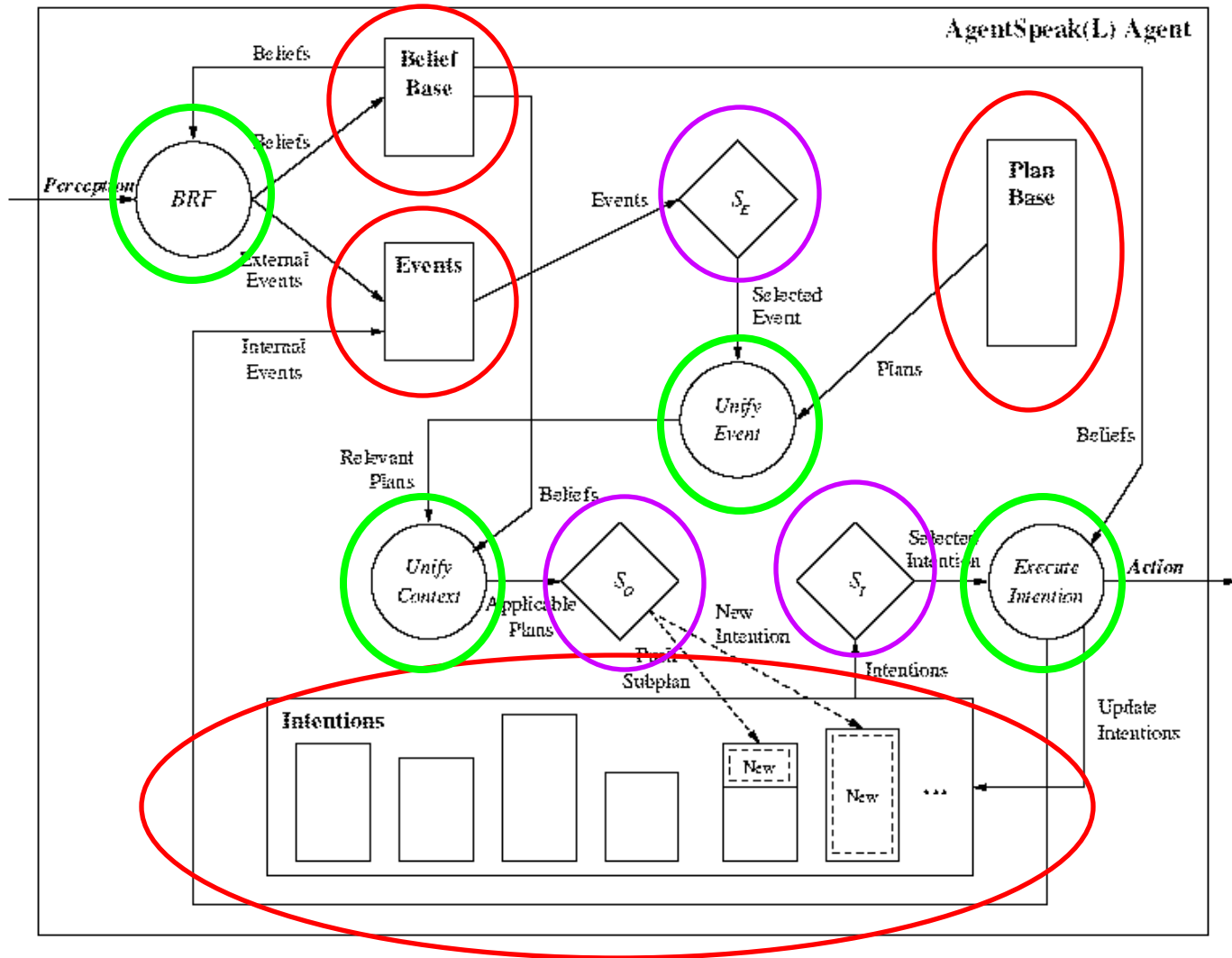
AgentSpeak(L)- Informal Semantic Intentions

- ***Intentions*** are particular courses of actions to which an agent has committed in order to handle certain events. Each intention is a stack of partially instantiated plans

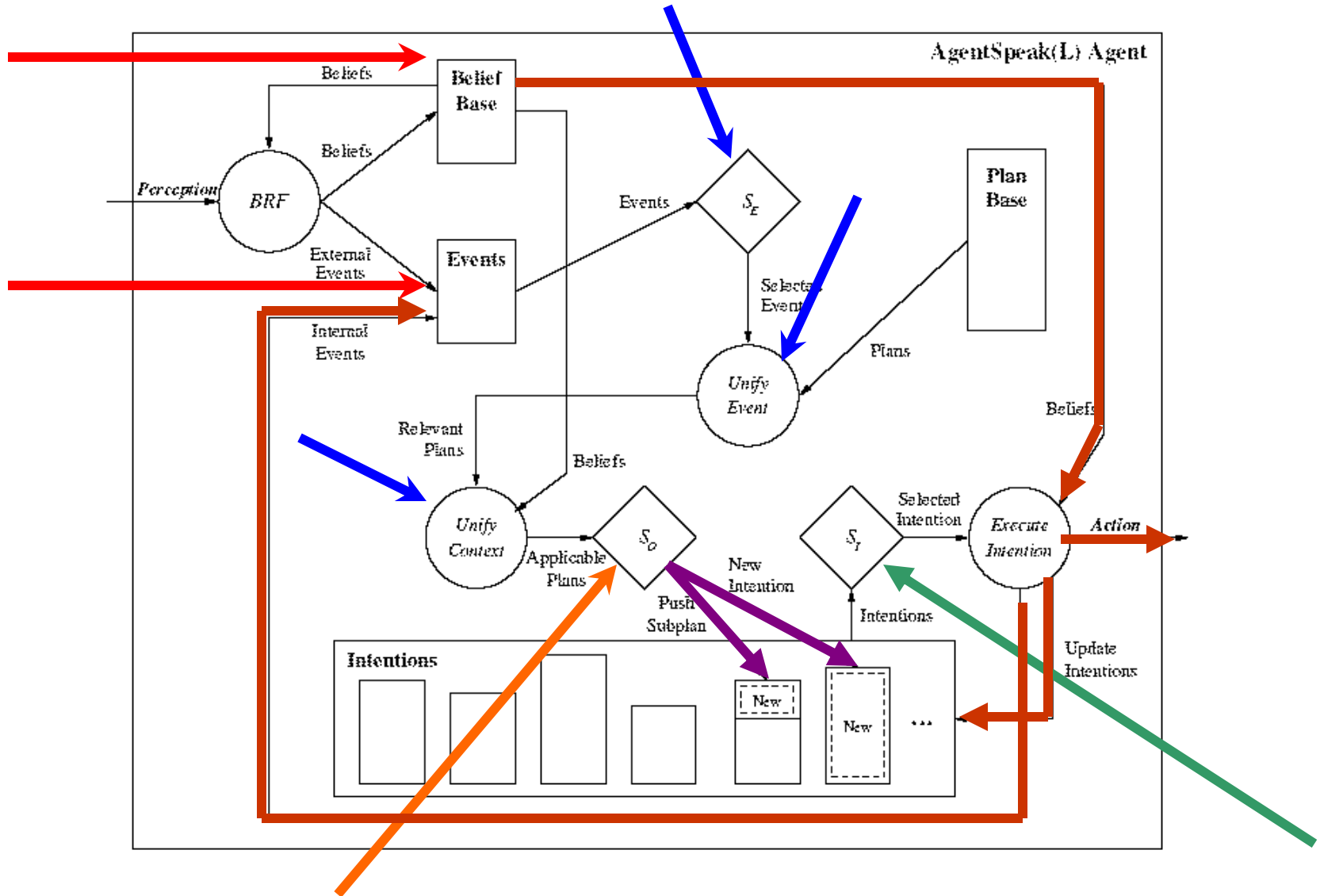
AgentSpeak(L)- Informal Semantic Selection Functions

- **SE** (the event selection function)
 - selects a single event from the set of events
- **SO**
 - selects an “option” (i.e., an applicable plan) from a set of applicable plans
- **SI**
 - selects one particular intention from the set of intentions.
- *The selection functions are agent-specific, in the sense that they should make selections based on an agent’s characteristics.*

AgentSpeak(L)- Informal Semantic



AgentSpeak(L)- Informal Semantic



AgentSpeak(L) Example



ALICE

- During lunch time, forward all calls to Carla.
- When I am busy, incoming calls from colleagues should be forwarded to Denise.

AgentSpeak(L) Example

Beliefs

```
user (alice) .  
user (bob) .  
user (carla) .  
user (denise) .  
~status (alice, idle) .  
status (bob, idle) .  
colleague (bob) .  
lunch_time ("11:30") .
```


AgentSpeak(L) Example Plans

```
user(alice).  
user(bob).  
user(carla).  
user(denise).  
~status(alice, idle).  
status(bob, idle).  
colleague(bob).  
lunch_time("11:30").  
+invite(X, alice) : lunch_time(t) ← !call_forward(alice, X, carla). (p1)  
+invite(X, alice) : colleague(X) ← !call_forward_busy(alice, X, denise). (p2)  
+invite(X, Y): true ← connect(X, Y). (p3)
```

```
+!call_forward(X, From, To) : invite(From, X)  
← +invite(From, To), - invite(From, X) (p4)
```

```
+!call_forvard_busy(Y, From, To) : invite(From, Y) &  
not(status(Y, idle))  
← +invite(From, To), - invite(From, Y). (p5)
```

AgentSpeak(L) Example

```
user(alice).
user(bob).
user(carla).
user(denise).
~status(alice, idle).
status(bob, idle).
colleague(bob).
lunch_time("11:30").

+invite(X, alice) : lunch_time(t)
                    ← !call_forward(alice, X, carla).           (p1)
+invite(X, alice) :   colleague(X)
                    ← !call_forward_busy(alice, X, denise).     (p2)
+invite(X, Y) : true ← connect(X, Y).                            (p3)
+!call_forward(X, From, To) : invite(From, X)
                    ← +invite(From, To), - invite(From, X)      (p4)
+!call_forward_busy(Y, From, To) : invite(From, Y) &
    not(status(Y, idle))
                    ← +invite(From, To), - invite(From, Y).     (p5)
```

Execution - 1

- a new event is sensed from the environment, `+invite(Bob, Alice)` (there is a call for Alice from Bob).
- There are three *relevant* plans for this event (p1, p2 and p3)
 - the event matches the triggering event of those three plans.

Relevant Plans	Unifier
<code>p1: +invite(X, alice) : lunch_time(t) ←!call_forward(alice, X, carla)</code>	
<code>p2: +invite(X, alice) : colleague(Bob) ← !call_forward_busy(alice, X, denise).</code>	<code>{X=bob}</code>
<code>p3 : +invite(X, Y): true ← connect(X,Y).</code>	<code>{Y=alice, X=bob}</code>

Execution - 2

- only the context of plan p2 is satisfied - `colleague(bob)` => p2 is *applicable*.
- a new intention based on this plan is created in the set of intentions, because the event was external, generated from the perception of the environment.
- The plan starts to be executed. It adds a new event, this time an internal event: `!call_forward_busy(alice,bob,denise)` .

Intention ID	Intension Stack	Unifier
1	<code>+invite(X,alice) : colleague(X)</code> <code><- !call_forward_busy(alice,X,denise)</code>	<code>{X=bob}</code>

Execution - 3

- a plan relevant to this new event is found (p5):

Relevant Plans	Unifier
<pre>p5: +!call_forward_busy(Y, From, To) : invite(From, Y) & not(status(Y, idle)) ← +invite(From, To), - invite(From, Y).</pre>	<pre>{From=bob, Y=alice, To=denise}</pre>

- p5 has the context condition true, so it becomes an *applicable* plan and it is pushed on top of *intention 1* (it was generated by an internal event)

Intention ID	Intension Stack	Unifier
1	<pre>+!call_forward_busy(Y,From,To) : invite(From,Y) & not status(Y,idle) ← +invite(From,To); -invite(From,Y)</pre>	<pre>{From=bob, Y=alice, To=denise}</pre>
	<pre>+invite(X,alice) : colleague(X) ← !call_forward_busy(alice,X,denise)</pre>	<pre>{X=bob}</pre>

Execution - 4

- A new internal event is created, `+invite(bob, denise)`.
- three relevant plans for this event are found, p1, p2 and p3.
- However, only plan p3 is applicable in this case, since the others don't have the context condition true.
- The plan is pushed on top of the existing intention.

Intention ID	Intension Stack	Unifier
1	<code>+invite(X,Y) :- connect(X,Y)</code>	{Y=denise, X=bob}
	<code>+!call_forward_busy(Y,From,To) : invite(From,Y) & not status(Y,idle) <- +invite(From,To); -invite(From,Y)</code>	{From=bob, Y=alice, To=denise}
	<code>+invite(X,alice) : colleague(X) <- !call_forward_busy(alice,X,denise)</code>	{X=bob}

Execution - 5

- on top of the intention is a plan whose body contains an action.
- the action is executed, `connect(bob, denise)` and is removed from the intention.
- When all formulas in the body of a plan have been removed (i.e., have been executed), the whole plan is removed from the intention, and so is the achievement goal that generated it.

Intention ID	Intension Stack	Unifier
1	<code>+!call_forward_busy(Y,From,To) : invite(From,Y) & not status(Y,idle) <- -invite(From,Y)</code>	{From=bob, Y=alice, To=denise}
	<code>+invite(X,alice) : colleague(X) <- !call_forward_busy(alice,X,denise)</code>	{X=bob}

- The only thing that remains to be done is `-invite(bob, alice)` (this event is removed from the beliefs base).
- This ends a cycle of execution, and the process starts all over again, checking the state of the environment and reacting to events.

Jason

- a fully-fledged interpreter for AgentSpeak(L)
- many extensions, providing a very expressive programming language for agents.
- allows configuration of a multi-agent system to run on various hosts.
- implemented in Java (thus it is multi-platform)
- available *Open Source* and is distributed under GNU LGPL.
- <http://jason.sourceforge.net/>

Jason characteristics

- support for developing Environments (Java)
- the possibility to run a multi-agent system distributed over a network
- fully customisable (in Java) selection functions, trust functions, and overall agent architecture (perception, belief-revision, inter-agent communication, and acting)
- a library of essential "internal actions"
- straightforward extensibility by user-defined internal actions, which are programmed in Java.



Conclusion

- AgentSpeak(L) has many similarities with traditional logic programming, which would favor its becoming a popular language
- it proves quite intuitive for those familiar with logic programming.
- it has a neat notation, thus providing quite elegant specifications of BDI agents.