Lecture 7: Formal Models of BDI Programming

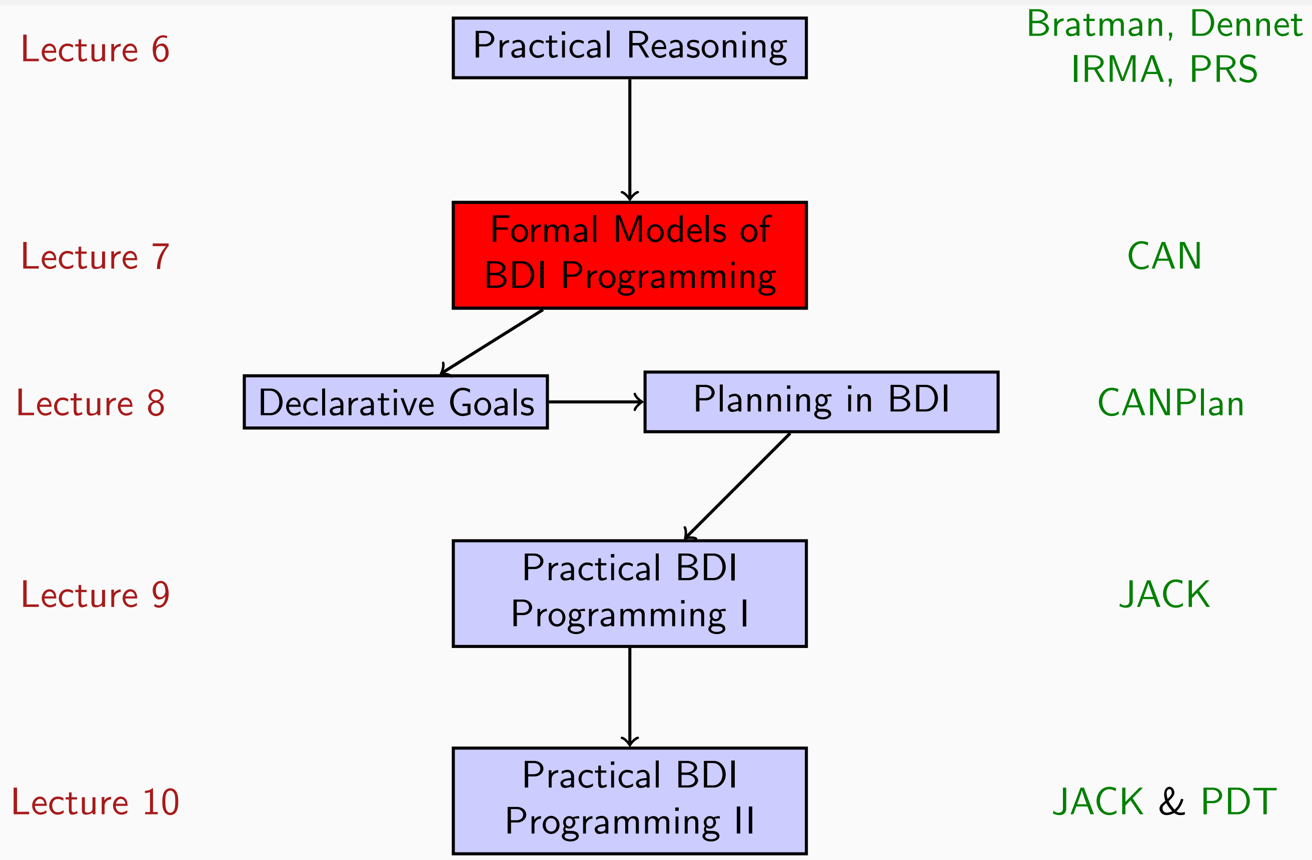
Autonomous Agents and Multiagent Systems
DIS, La Sapienza - PhD Course

Sebastian Sardina[1]

[1]Department of Computer Science and Information Technology
RMIT University
Melbourne, AUSTRALIA

November 27, 2007

Roadmap for Next Lectures



Outline

Review of Last Lecture: Practical Reasoning

1. Intentional stance (Daniel Dennett (1987))
   - High-level abstraction of behavior at the level of minds.
   - Rational behavior can be understood in terms of mental properties:
     - beliefs, desires, goals;
     - fear, hopes, etc.
2. Practical reasoning (Michael Bratman (1990))
   - Reasoning for acting: the process of figuring out what to do.
   - Two activities: deliberation and means-end analysis.
3. Commitments (on goals/intentions & plans)
   - fanatical, single-minded, open-minded.
4. Agent architectures
   - IRMA & PRS.
   - Built around: beliefs, desires, plan libraries, intentions, filter, etc.
5. Agent theory
   - Cohen & Levesque's "Intention = Choice+Commitment".
   - Rao & Georgeoff's BDI logic.

This Lecture: BDI Programming

Objective: a programming language that can provide:

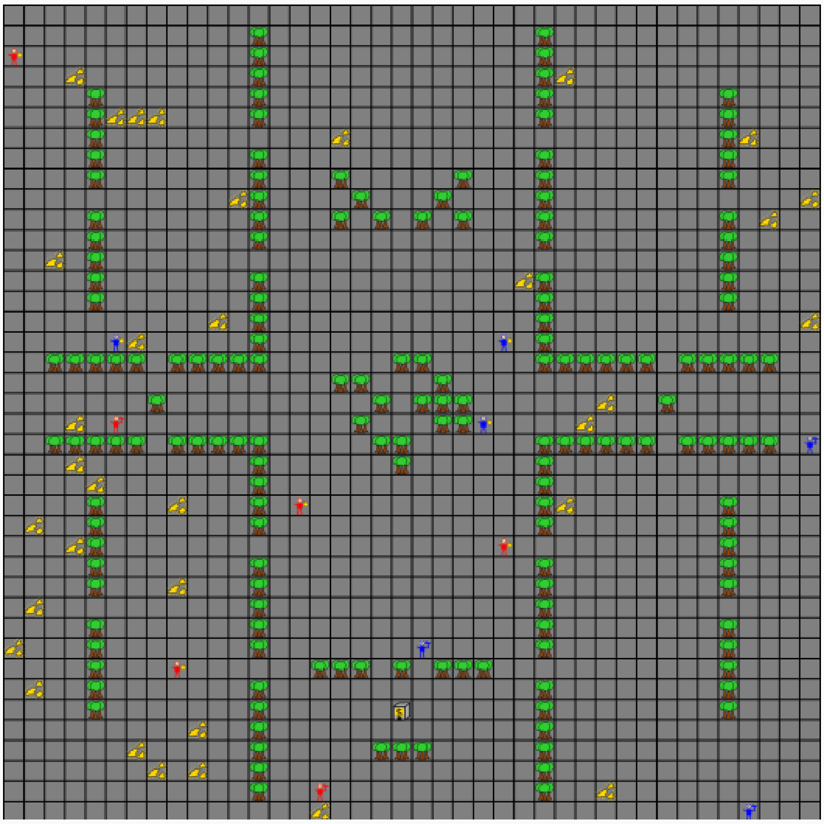
- autonomy: does not require continuous external control;
- pro-activity: pursues goals over time; goal directed behavior;
- situatedness: observe & act in the environment;
- reactivity: perceives the environment and responds to it.
- flexibility: achieve goals in several ways.
- robustness: will try hard to achieve goals.

And also: modular scalability & adaptability!

An Example: Gold Mining Game

2 teams competing to collect and drop gold in the depot

- dynamic
- complex
- unknown information
- failing actions
- failing sensors
- multi-agents



Some Constraints

We want to program intelligent systems under the following constraints:

1. The agent interacts with an external environment.
   - A grid world with gold pieces, obstacles, and other agents.
2. The environment is (highly) dynamic; may change in unexpected ways.
   - Gold pieces appear randomly.
3. Things can go wrong; plans and strategies may fail.
   - A path may end up being blocked.
4. Agents have dynamic and multiple objectives.
   - Explore, collect, be safe, communicate, etc.
   - Motivations/goals/desires may come and go.

Some Assumptions

Luckily, we can also assume that:

1. Failure is generally not catastrophic.
   - If gold is dropped, we just pick it up again.
   - If a tree blocks the path, we just go around it.
2. We can understand the system at the "intentional" level.
   - Agents "desire" to collect as much gold as possible.
   - Agents "believe" they are close to the depot.
3. There is "some" sensible known procedural knowledge of the domain.
   - It is "good" to avoid obstacles.
   - If we see gold close, then go and collect it.
   - If we bump into an unknown wall, then walk along it.

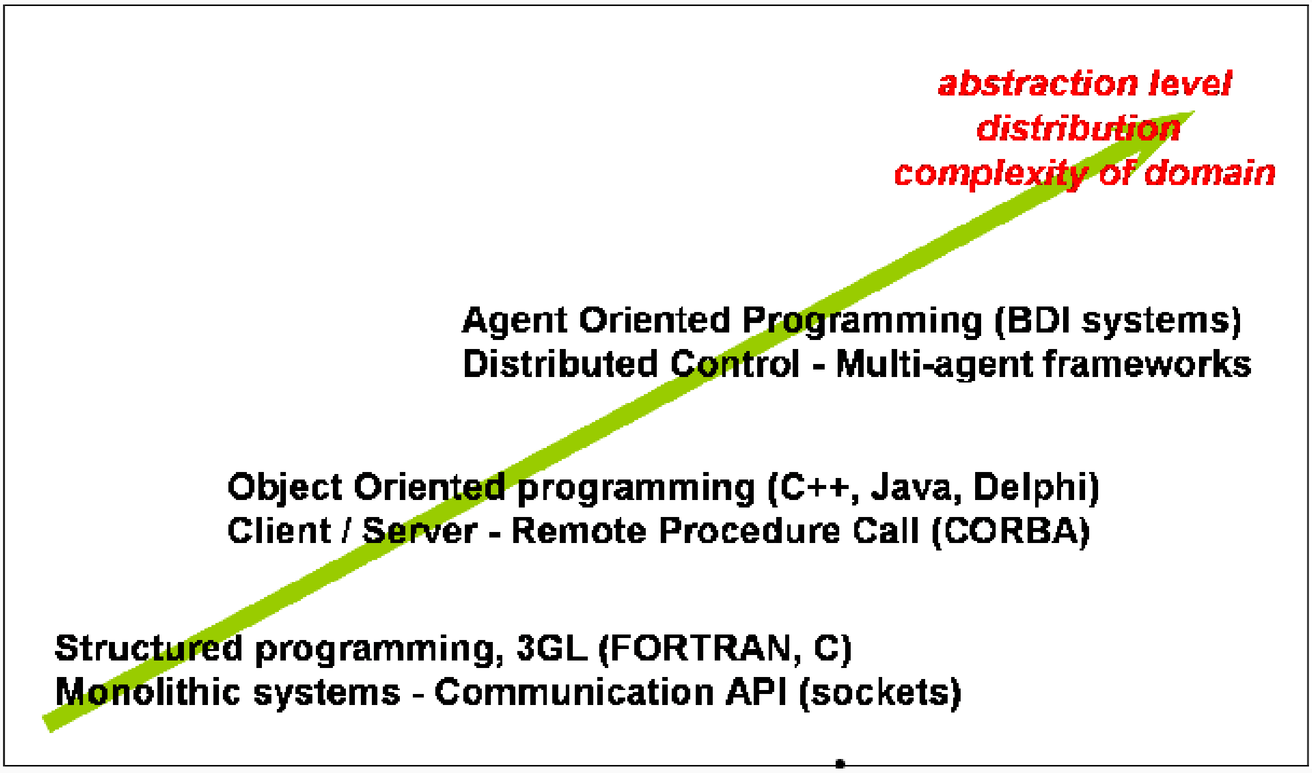
Agent-Oriented Programming (Shoam 1993)

1. Philosophers have produced theories of (human) rational action:
    - Folk psychology.
    - Practical reasoning.
    - Intentional systems.
2. Theorists have taken this and developed theories to represent the properties of agents (humans or not).
    - Relation between mental attitudes.
    - Commitment.
    - Rational architectures.

So, why not directly program agents in terms of the mentalistic, intentional notions?

∴ we will study one agent-oriented approach: BDI-style Programming.

Technology Development



Parent Technologies

1. artificial intelligence
2. software engineering
3. distributed systems
4. organizational science
5. databases
6. economics
7. game theory
8. artificial life

Some Agent Companies

1. Agent Oriented Software (JACK Intelligent Agents)
2. Reticular (AgentBuilder intelligent agents)
3. Gensym Corp. (G2: real-time business rule engine platform)
4. Agentis Software (AdaptivEnterprise: goal-oriented system)
5. Whitestein (autonomic self-managing, goal-oriented business solutions)
6. IBM (Aglets: Java based mobile agent platform)
7. Genesys Telecommunications (customer service)
8. Hewlett Packard
9. ...

Core Aspects

Most basic concepts:

- **Agent** Autonomous SW entity.
- **Percepts** Information perceived from the environment.
- **Actions** Affects the environment.

Some additional concepts:

1. Message (inter agent communication)
2. Event (trigger)
3. Goal (what to do)
4. Plan (how to do)
5. Interaction protocol (conversation pattern)
6. Organization (also team, institution)
7. Role (agent abstraction)
8. ...

Building Agents: BDI Agent-oriented Programming

A new programming model and architecture to simplify the construction of todays large complex systems situated in dynamic environments:

1. View a system as composed of autonomous interacting entities (agents) which pursue their own goals and act in a rational manner.
2. Internal state and decision process of agents is modelled in an intuitive manner following the notion of mental attitudes.
3. Goal orientation: instead of directly requesting the agents to perform certain actions, the developer can define more abstract goals for the agents.
   - provides a certain degree of flexibility on how to achieve the goals.

Can be seen as a "successor" of object-oriented programming

Some BDI Agent-oriented Programming Languages

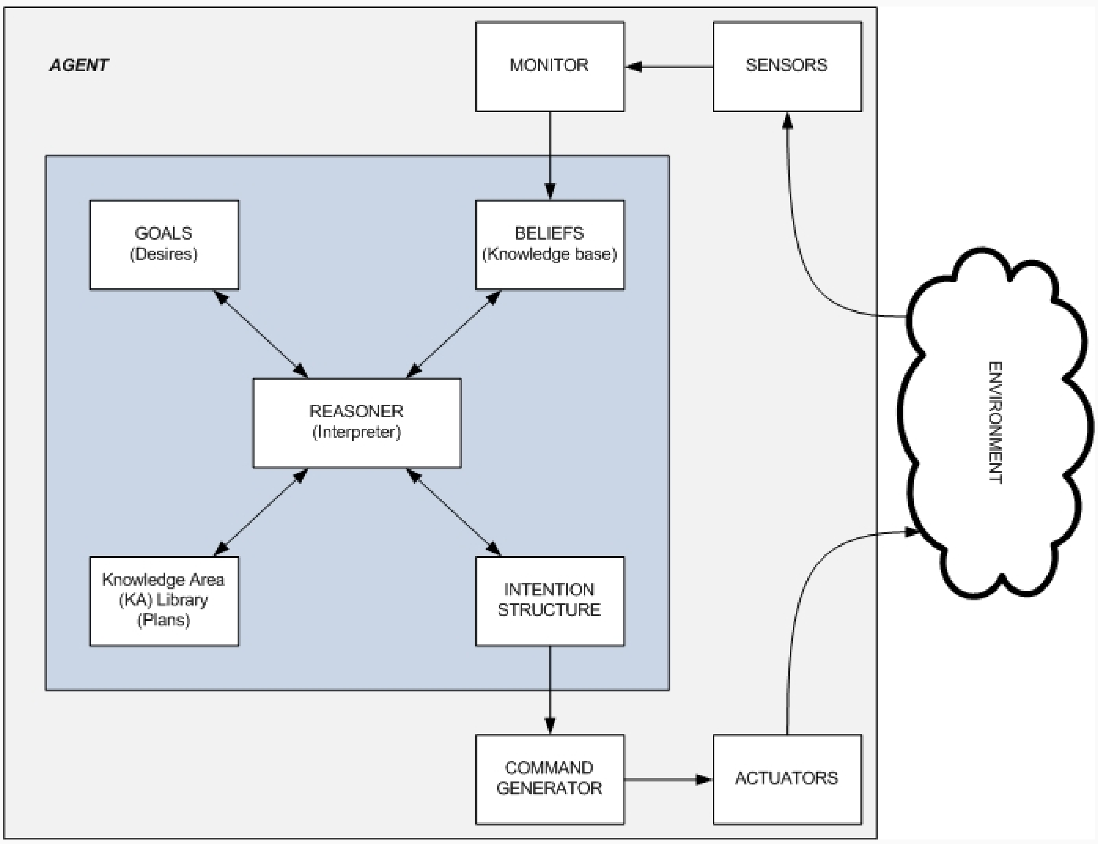
Some formal BDI programming languages:

1. AgentSpeak: first formal BDI language.
2. 3APL: language with different kind of plan rules.
3. GOAL: based on declarative goals only.
4. CAN(Plan): failure-handling, declarative goals, & planning. ⟸ TODAY

Some BDI programming language systems/platforms/architectures:

1. PRS and dMars: first BDI-based systems; C++ based.
2. JAM: a hybrid extension of PRS.
3. JASON: JAVA-based implementation of AgentSpeak.
4. JADEX: based on JADE communication platform.
5. SPARK: SRI's BDI system.
6. JACK: powerful commercial JAVA-based BDI-system. ⟸ NEXT LECT.

Typical BDI-style System



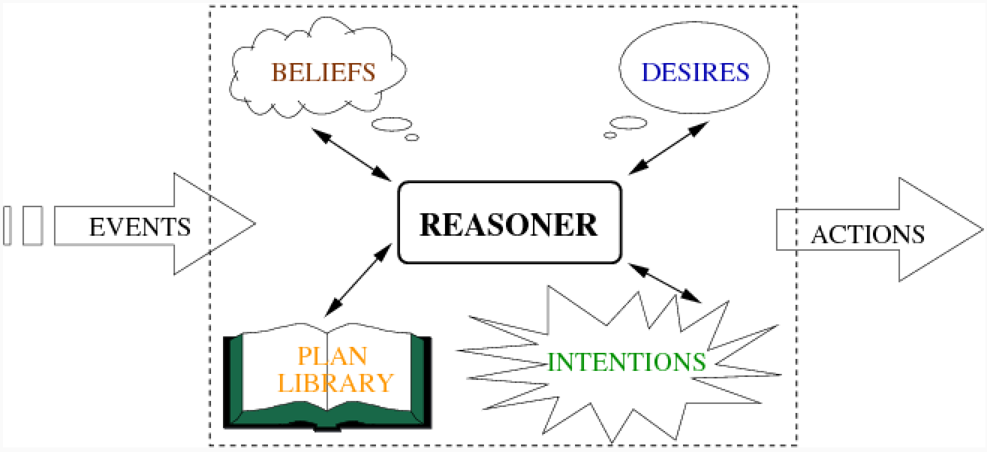
Key Features of BDI Agent-oriented Systems

Beliefs: information about the world.

Events: goals/desires to resolve; internal or external.

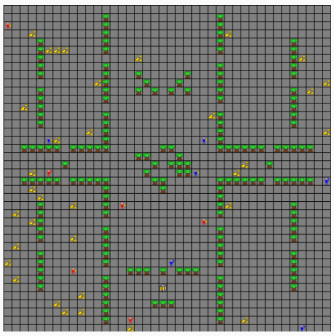
Plan library: recipes for handling goals-events.

Intentions: partially uninstantiated programs with commitment.



Key Features of BDI Agent-oriented Systems (cont.)

In the gold-mining game:



Beliefs: current location & location of depot.
size of grid, # of gold pieces carrying, etc.

Events: a gold piece is observed east;
player3 communicates its location;
the coordinator requests to explore the grid;
we formed the *internal goal* to travel to $loc(10, 22)$

Plan library: if I see gold here & I am not full, collect it.
if I hit an obstacle, go around it.
if I don't know of any gold, explore grid.
if I see gold around, move there and collect.

Intentions: I am currently traveling to the depot.
I am informing my team-mates of new obstacles I find.

Events & Plans

1. Events stand for the goals/desires/tasks to be achieved or resolved:
   - percepts: $goldAt(east)$, $goldDropped$, etc;
   - communication: $told(player3, loc(3, 2))$;
   - external request/goal: $achieve(explore\_grid)$;
   - internal sub-goal: $go\_to(loc(10, 22))$.

2. Plans stand for strategies useful to resolve (pending) events:
   - encode typical operational procedures in the domain;
   - non-deterministic;
   - event & context dependent;

$$e : \psi \longleftarrow P$$

P is a good strategy to resolve event e if context ψ is believed true.

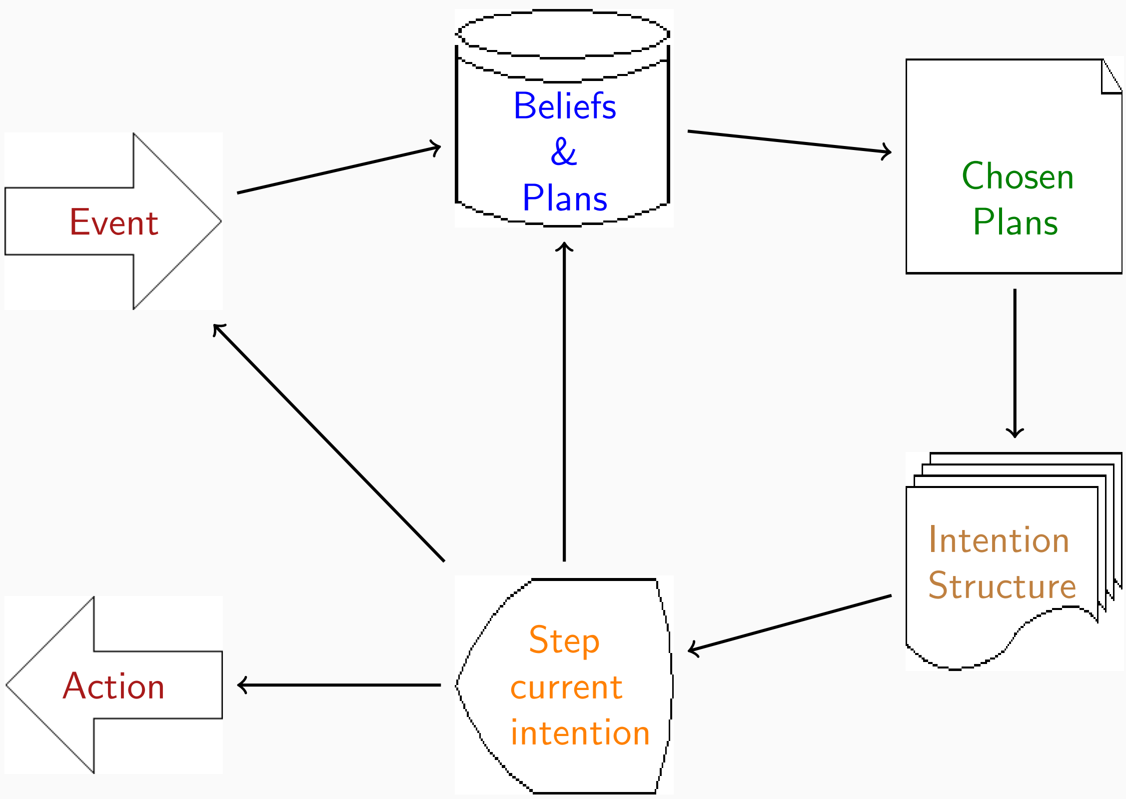
Plans in PRS: Clearing a Block

```
Plan: {
NAME: "Clear a block"
GOAL:
    ACHIEVE CLEAR $OBJ;
CONTEXT:
    FACT ON $OBJ2 $OBJ;
BODY:
    EXECUTE print "Clearing " $OBJ2 " from on top of " $OBJ "\n";
    EXECUTE print "Moving " $OBJ2 " to table.\n";
    ACHIEVE ON $OBJ2 "Table";
EFFECTS:
    EXECUTE print "CLEAR: Retracting ON " $OBJ2 " " $OBJ "\n";
    RETRACT ON $OBJ1 $OBJ;
FAILURE:
    EXECUTE print "\n\nClearing block " $OBJ " failed!\n\n";
}
```

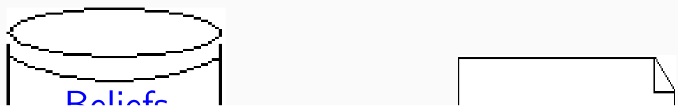
Intentions

1. Agent's intentions are determined dynamically by the agent at runtime based on its known facts, current goals, and available plans.
2. An intention is just a partially executed strategy:
   - comes from the plan library when resolving events.
3. An intention represent a focus of attention:
   - something the agent is currently working on;
   - actions/behavior arises as a consequence of executing intentions.
4. An agent may have several intentions active at one time.
   - different simultaneous focuses of attention;
5. A new intention is created when an external event is addressed.
6. An intention may create/post an internal event:
   - the intention will be updated when this event is addressed.

The BDI Execution Cycle [Rao&Georgeff 92]



The BDI Execution Cycle [Rao&Georgeff 92]



The BDI Execution Cycle: Detailed Version

1. Observe the environment for new *external* events.
2. Pick a pending event *e*.
3. Select relevant plans from library (match event).
4. Select applicable plans from relevant set (match context).
5. If event *e* is external, create new intention with selected plan.
6. If event *e* is internal, update intention with selected plan on top.
7. Partially execute some intention (may post internal events).
    - If execution fails, then perform failure recovery.
8. Repeat cycle.

Key Points of BDI Programming

- Flexible and responsible to the environment: "reactive planning."
    - Well suited for soft real-time reasoning and control.
- Relies on context sensitive subgoal expansion: "act as you go."
- Leave for as late as possible the choice of which plans to commit to as the chosen course of action to achieve (sub)goals.
- Modular and incremental programming.
- Nondeterminism on choosing plans and bindings.

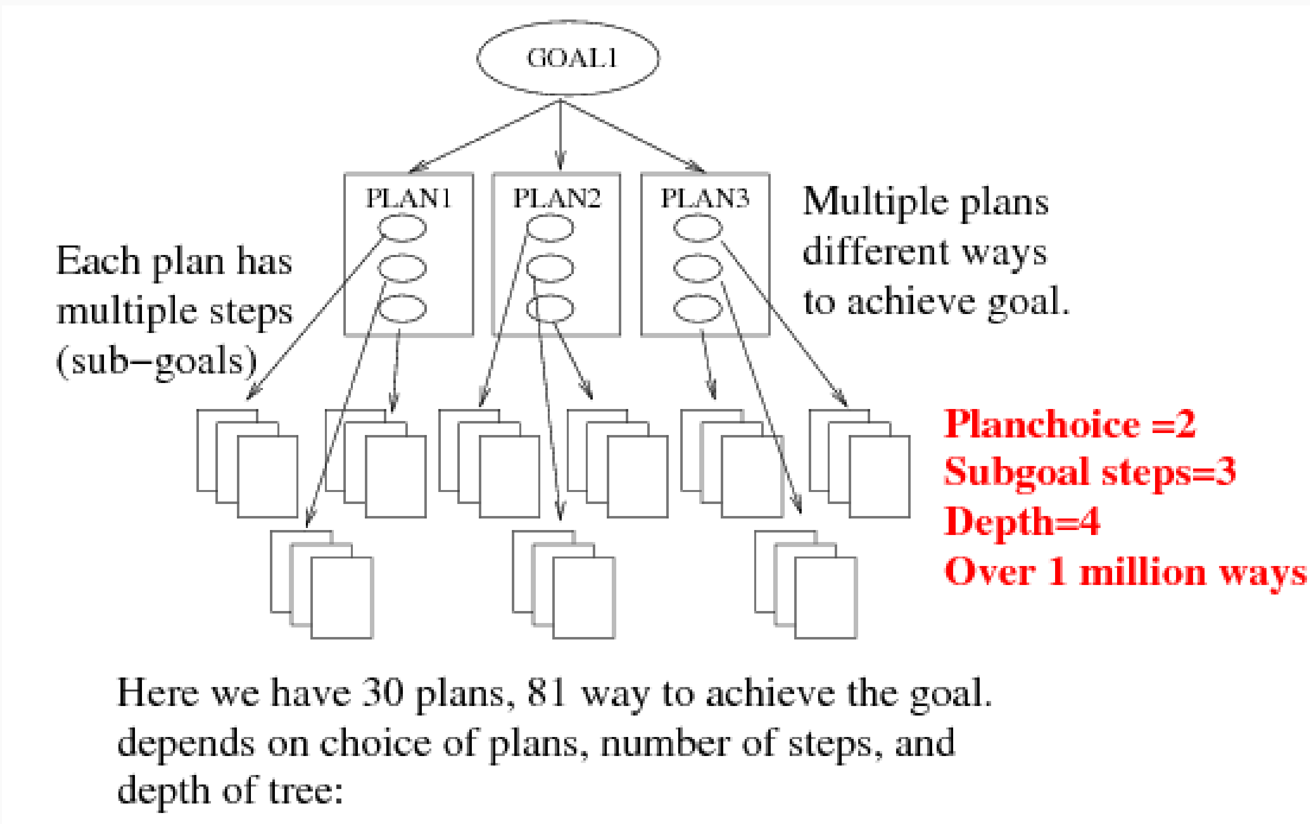
Key Points of BDI Programming

- Flexible and responsible to the environment: "reactive planning."
  - Well suited for soft real-time reasoning and control.
- Relies on context sensitive subgoal expansion: "act as you go."
- Leav
  the

> BDI Programming =
> Implicit Goal-based Programming + Rational Online Executor

- Modular and incremental programming.
- Nondeterminism on choosing plans and bindings.

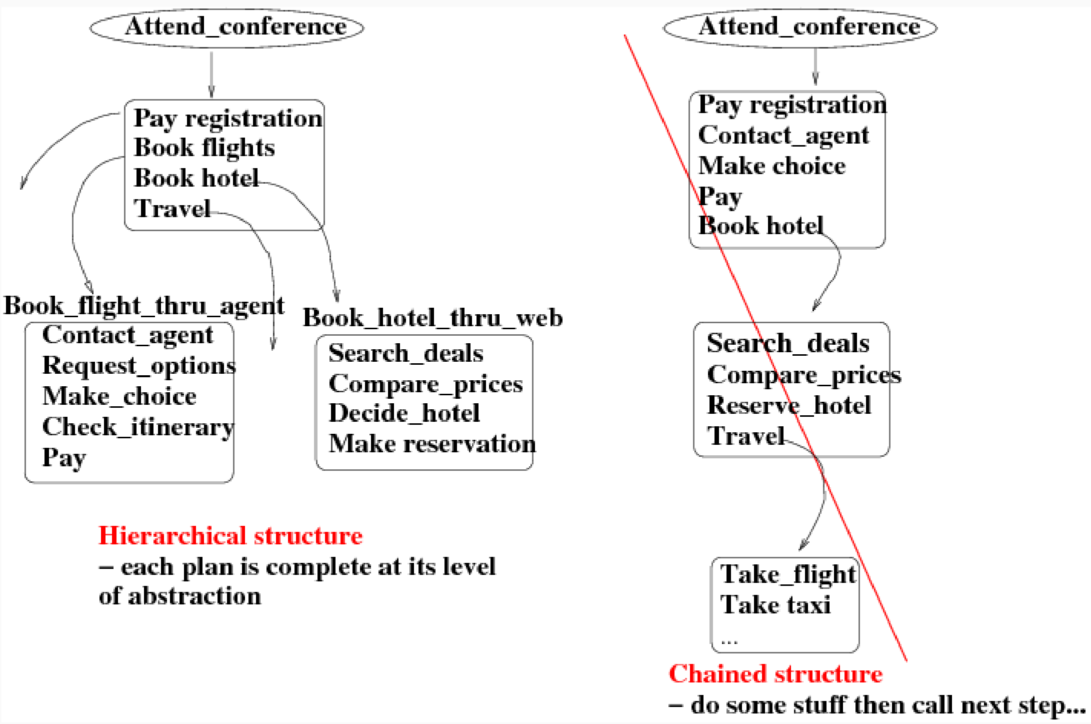
Possibility of Many Options



Making Use of the BDI Framework

1. Provide alternative plans where possible.
2. Break things down into subgoal steps.
3. Use subgoals and alternative plans rather than **if**... **then** in code.
4. Keep plans small and modular.
5. Plans are abstract modules - don't chain them together like a flowchart.

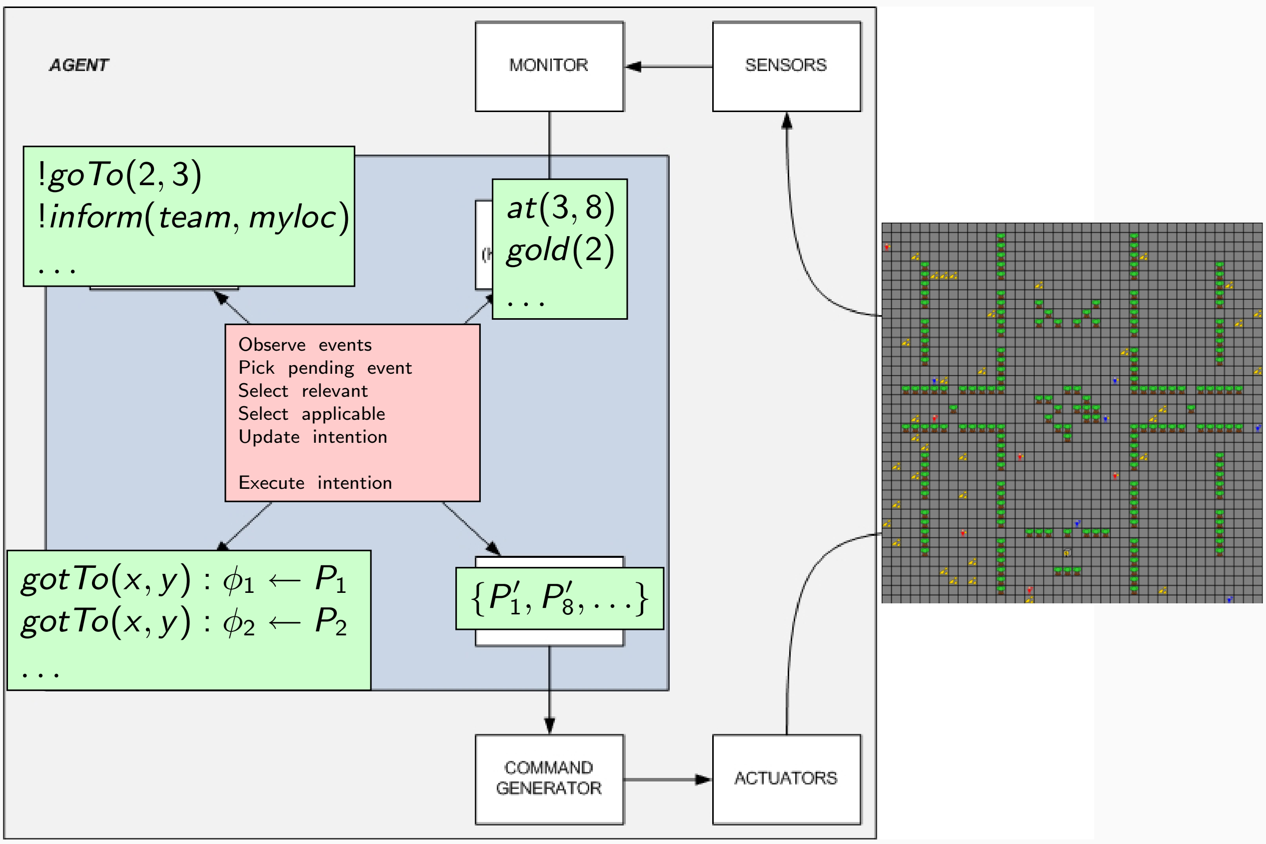
Plan Structure



Structuring Plans and Goals

1. Make each plan complete at a particular abstraction level.
   - A high-level but complete plan for *Attend_Conference*.
2. Use a subgoal - even if only one plan choice for now.
   - Decouple a goal from its plans.
3. Modular and easy to add other plan choices later.
   - Booking a flight can now be done with the Internet, if available!
4. Think in terms of subgoals, not function calls.
   - What way-points do we need to achieve so as to realize a goal?
5. Learn to pass information between subgoals.
   - How are these way-points inter-related w.r.t. data?

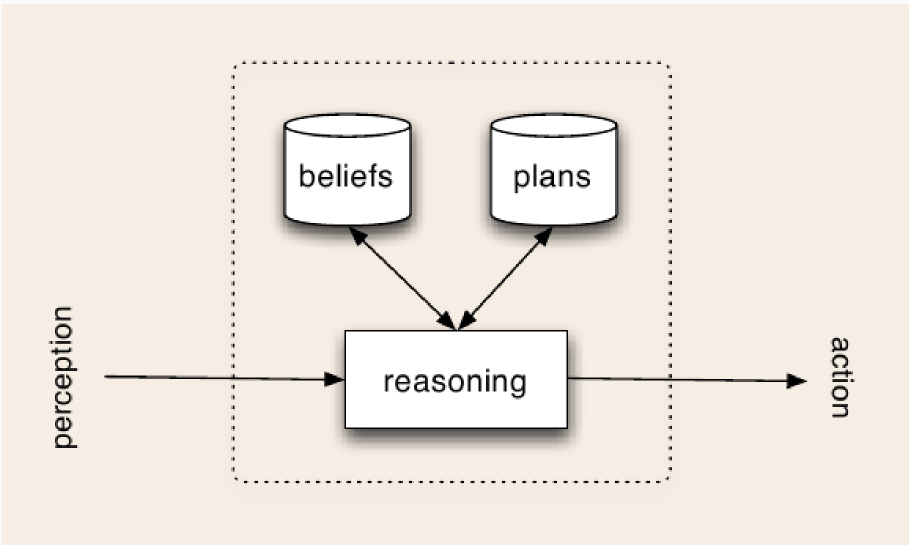
Typical BDI-style System



Historical Evolution

1. A few actual BDI systems-frameworks were developed (late 80s'):
   - IRMA, PRS, dMars, etc.
2. Anand Rao tried to formally capture the common features (1996):
   - AgentSpeak appeared: events, beliefs, intentions, rational cycle, etc.
   - Can be seen as an elegant extension of logic programming for the implementation of BDI agents (reactive planning systems).
3. Many other languages appeared in the BDI-tradition (late 90s'):
   - 3APL
   - JACK
   - JADEX
   - JASON
   - ...
4. CAN(Plan) is similar to AgentSpeak but with (2002-2007):
   - built-in semantics for failure handling (as in real BDI systems);
   - declarative goals (as in agent theories);
   - planning capabilities (as in HTN-planning & IndiGolog).
   - interleaved concurrency in plans (as in ConGolog);

Basic Architecture of CAN



Basic Syntax of CAN

- Beliefs: *predicate(terms)* (e.g., $curLoc(2,3)$)
- Actions: *action(terms)* (e.g., $pickGold$, $move(left)$).
- Events: *event(terms)* (e.g., $goTo(10,20)$, $block(agnt3)$).
- Plans: $event : context \leftarrow body$.
- Body:

act	*primitive action*
$+b$	*belief addition*
$-b$	*belief deletion*
$?\phi$	*tests goal*
$!e$	*posting of achievement event goal*
$P_1; P_2$	*sequence*
$P_1 \| P_2$	*interleaved concurrency*

A Simple Example

```
friend(john).
friend(anne).
time(3pm).
phone(john,93812298).
....

meet(X) : friend(X) <- !greet_friend(X).
meet(X) : not friend(X) <- !greet(X).

greet(X) : time(T) and T<=12pm <- say('Good morning').
greet(X) : time(T) and T>12pm <- say('Good afternoon').
greet(X) : true <- say('Hello!').

greet_friend(X) : true <- say('Hi '+X).

told(X,phone,N) : not friend(X) <- +friend(X), +phone(X,N).
```

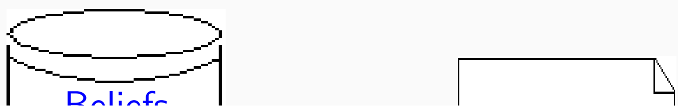
A Simple Example II

```
currLoc(10,12).
goldAt(10,13).
goldAt(2,3).
carrying(2).
depotLoc(20,14).
...

goTo(X,Y) : grid_known >50 <- !planPath(X,Y, Path), !follow(Path).
goTo(X,Y) : not grid_known>50 <- !getTowards(X,Y,X2,Y2),
                                 !moveTo(X2,Y2).
...

act : fullLoaded <- ?depotLoc(X,Y), +toDepot, !goTo(X,Y).
act : currLoc(X,Y) and goldAt(X,Y) <- pick.
act : goldAt(X,Y) <- !goTo(X,Y).
act : not goldAt(X,Y) <- ?team(Agnt), !askForGold(Agnt).
...
```

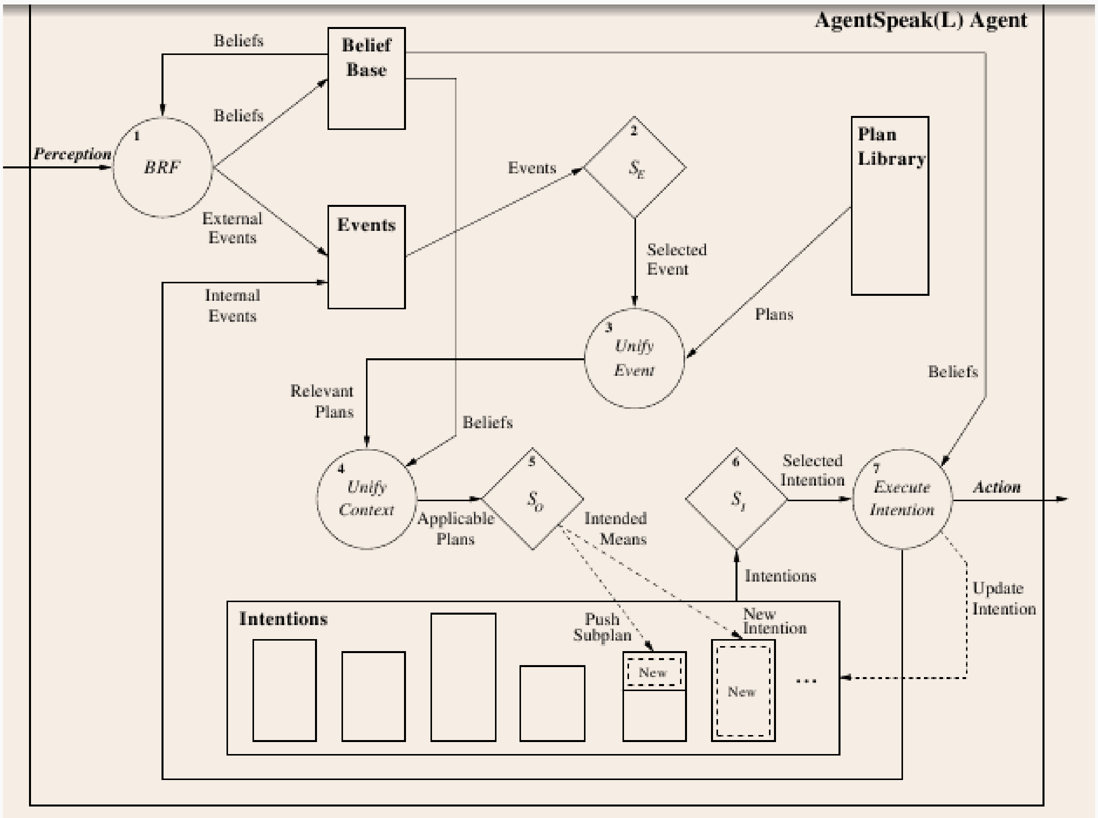

The BDI Execution Cycle [Rao&Georgeff 92]



The BDI Execution Cycle: Detailed Version

1. Observe the environment for new *external* events.
2. Pick a pending event *e*.
3. Select relevant plans from library (match event).
4. Select applicable plans from relevant set (match context).
5. If event *e* is external, create new intention with selected plan.
6. If event *e* is internal, update intention with selected plan on top.
7. Partially execute some intention (may post internal events).
   - If execution fails, then perform failure recovery.
8. Repeat cycle.

Detailed Architecture of AgentSpeak



The CAN Language [Winikioff et al. 2002]

CAN: **C**onceptual **A**gent **N**otation

Can be seen as an extension of Rao's AgentSpeak.

A CAN agent is defined as $Agt = \langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle$, where:

- $\mathcal{N}$ is the agent name.
- $\mathcal{B}$ is the belief base: current agent's knowledge.
- $\mathcal{A}$ is the sequence of actions executed so far.
- Π is a plan library containing plan rules $e : \psi \leftarrow P$:
  - e is the triggering event
  - ψ is the context condition
  - P is the plan-body
- Γ is the intention base: partially uninstantiated plan-bodies.

The CAN Language: Beliefs, Actions & Goals

- In principle, $\mathcal{B}$ is any KR formalism that allows queries and updates:
  - $\mathcal{B} \models \phi$, $B \cup \{b\}$ and $\mathcal{B} \setminus \{b\}$.
- In practice: a database of facts.
- If b is a predicate symbol, and $t_1, \ldots, t_n$ are (first-order) terms, $b(t_1, \ldots, t_n)$ is a belief atom.
  - Ground belief atoms are base beliefs
  - If ϕ is a belief atom, ϕ and $\neg\phi$ are belief literals.
- If a is an action symbol and $t_1, \ldots, t_n$ are first-order terms, then $a(t_1, \ldots, t_n)$ is an action.
- If g is a predicate symbol, and $t_1, \ldots, t_n$ are terms, $!g(t_1, \ldots, t_n)$ and $?g(t1, \ldots, t_n)$ are goals
  - '!' denotes posting of achievement goals.
  - '?' denotes test goals.

Intro. BDI Framework CAN Conclusions Overview Syntax Semantics Other Languages

The CAN Language: Plans & Intentions

- Π is a plan library containing plan rules $e : \psi \leftarrow P$:

act	*primitive action*
$+b$	*belief addition*
$-b$	*belief deletion*
$?\phi$	*tests goal*
$!e$	*achievement event goal*
$P_1; P_2$	*sequence*
$P_1 \| P_2$	*interleaved concurrency*

  Plus, the following system-auxiliary constructs:

$nil(\theta)$	*empty program with bindings*
$P_1 \rhd P_2$	*try P_1; else P_2*
$(\!\|\psi_1 : P_1, \ldots, \psi_n : P_n\|\!)$	*guarded plans*

- Γ is the intention base: set of partially uninstantiated plan-bodies.
  - E.g.: `(?phone(john,N);call(N);!talk) || !cook_dinner`

Semantics of CAN

CAN has an single-step operational semantics (Plotkin 1981):

- Give meaning to computer programs in a mathematically rigorous way.
- System is interpreted as sequences of computational steps. These sequences then are the meaning of the program.
- Set of rules defining the transitions between system configurations.
- Contrast with denotational semantics & axiomatic semantics.

We will use two types of configurations:

1. Agent configuration: $\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle$.
   - Γ is a set of partially-instantiated plan bodies – the intention base.
2. Intention configuration: $\langle \Pi, \mathcal{B}, \mathcal{A}, P \rangle$.
   - P is just one partially-instantiated plan body – the selected intention.

What we need are rules to state how configurations may evolve (one step).

Semantics of CAN (cont.)

The semantics of CAN is modularly defined in two levels:

1. Agent-level semantics: $\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}', \mathcal{A}', \Gamma' \rangle$.
   - State that agent configuration $\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle$ may legally evolve to configuration $\langle \mathcal{N}, \Pi, \mathcal{B}', \mathcal{A}', \Gamma' \rangle$.
2. Intention-level semantics: $\langle \Pi, \mathcal{B}, \mathcal{A}, P \rangle \longrightarrow \langle \Pi, \mathcal{B}', \mathcal{A}', P' \rangle$.
   - State that intention configuration $\langle \Pi, \mathcal{B}, \mathcal{A}, P \rangle$ may legally evolve to configuration $\langle \Pi, \mathcal{B}', \mathcal{A}', P' \rangle$.

Legal transitions are characterized by a set of rules of the form:

$$\frac{\text{Set of conditions}}{C \longrightarrow C'}\ \textit{RuleName}$$

Definition (BDI Agent Execution)

A BDI *execution* E of an agent $C_0 = \langle \mathcal{N}, \Pi, \mathcal{B}_0, \mathcal{A}_0, \Gamma_0 \rangle$ is a, possibly infinite, sequence of agent configurations $C_0 \cdot C_1 \cdot \ldots$ such that $C_i \Longrightarrow C_{i+1}$, for every $i \geq 0$. A *terminating* execution is a finite execution $C_0 \cdot \ldots \cdot C_n$ with $\Gamma_n = \{\}$.

Agent-Level Semantics

$$\frac{P \in \Gamma \quad \langle \mathcal{B}, \mathcal{A}, P\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P'\rangle}{\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma\rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}', \mathcal{A}', (\Gamma \setminus \{P\}) \cup \{P'\}\rangle} \; Agt_{step}$$

$$\frac{e \text{ is a new external event}}{\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma\rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \cup \{!e\}\rangle} \; Agt_{event}$$

$$\frac{P \in \Gamma \quad \langle \mathcal{B}, \mathcal{A}, P\rangle \not\longrightarrow}{\langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma\rangle \Longrightarrow \langle \mathcal{N}, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \setminus \{P\}\rangle} \; Agt_{clean}$$

Execute an active intention P. Assimilate an external event e. Remove an active intention P that is blocked.

Intention-Level Semantics: Basic Programs

$$\frac{\mathcal{B} \models \phi\theta}{\langle \mathcal{B}, \mathcal{A}, ?\phi\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, nil(\theta)\rangle} \; ?$$

$$\frac{}{\langle \mathcal{B}, \mathcal{A}, act\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A} \cdot act, nil(\emptyset)\rangle} \; do$$

$$\frac{}{\langle \mathcal{B}, \mathcal{A}, +b\rangle \longrightarrow \langle \mathcal{B} \cup \{b\}, \mathcal{A}, nil(\emptyset)\rangle} \; +b$$

$$\frac{}{\langle \mathcal{B}, \mathcal{A}, -b\rangle \longrightarrow \langle \mathcal{B} \setminus \{b\}, \mathcal{A}, nil(\emptyset)\rangle} \; -b$$

Goal test condition – propagate corresponding bindings.
Primitive action execution – actions just execute.
Addition & deletion of a belief atom.

Intention-Level Semantics: Complex Programs

$$\frac{\langle \mathcal{B}, \mathcal{A}, P_1\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_1'\rangle}{\langle \mathcal{B}, \mathcal{A}, P_1; P_2\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_1'; P_2\rangle}\ Seq \qquad \frac{}{\langle \mathcal{B}, \mathcal{A}, nil(\theta)\ ;P\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, P\theta\rangle}\ Seq_t$$

$$\frac{\langle \mathcal{B}, \mathcal{A}, P_1\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P'\rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \parallel P_2\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \parallel P_2\rangle}\ \|_1 \qquad \frac{}{\langle \mathcal{B}, \mathcal{A}, nil \parallel P_2\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, P_2\rangle}\ \|_{t_1}$$

$$\frac{\langle \mathcal{B}, \mathcal{A}, P_1\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P'\rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rhd P_2\rangle}\ \rhd \qquad \frac{}{\langle \mathcal{B}, \mathcal{A}, nil(\theta) \rhd P'\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, nil(\theta)\rangle}\ \rhd_t$$

$$\frac{P_1 \neq nil \quad \langle \mathcal{B}, \mathcal{A}, P_1\rangle \not\longrightarrow \quad \langle \mathcal{B}, \mathcal{A}, P_2\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_2'\rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_2'\rangle}\ \rhd_f$$

Sequence of programs – propagate corresponding bindings.
Interleaved concurrency – two analogous rules for P_2.
Try execution – jump to P_2 if P_1 is not working.

Intention-Level Semantics: Selection & Failure

When an unresolved event is addressed:

1. inspect plan-library for potential *relevant* plans;
2. select one of those plans that is *applicable*;
3. start executing.

When the plan P being pursued for an event e has problems executing:

1. is there an alternative applicable strategy P' we can follow?
   - maybe same plan-strategy but with different variable bindings.
2. if so, switch to it!
3. otherwise, go up in the hierarchy of goals and do the same reasoning.

All this is achieved by means of the following two special constructs:

$P_1 \rhd P_2$	*try P_1 if possible; else P_2*
$(\!|\psi_1 : P_1, \ldots, \psi_n : P_n|\!)$	*guarded (relevant) plans*

Intention-Level Semantics: Selection & Failure (cont.)

$$!e \longrightarrow (\!|\psi_1 : P_1, \ldots, \psi_n : P_n|\!) \longrightarrow$$

$$P_i\theta_i \rhd (\!|\psi_1 : P_1, \ldots, \underline{\psi_i \wedge \vec{x} \neq \theta_i : P_i}, \ldots, \psi_n : P_n|\!) \xrightarrow{*}$$

$$nil \rhd (\!|\psi_1 : P_1, \ldots, \psi_i \wedge \vec{x} \neq \theta_i : P_i, \ldots, \psi_n : P_n|\!) \longrightarrow nil$$

$$P_i'\theta_i \rhd (\!|\psi_1 : P_1, \ldots, \psi_i \wedge \vec{x} \neq \theta_i : P_i, \ldots, \psi_n : P_n|\!) \longrightarrow$$

$$P_j\theta_j \rhd (\!|\psi_1 : P_1, \ldots, \psi_i \wedge \vec{x} \neq \theta_i : P_i, \ldots, \underline{\psi_j \wedge \vec{x} \neq \theta_j : P_j}, \ldots, \psi_n : P_n|\!)$$

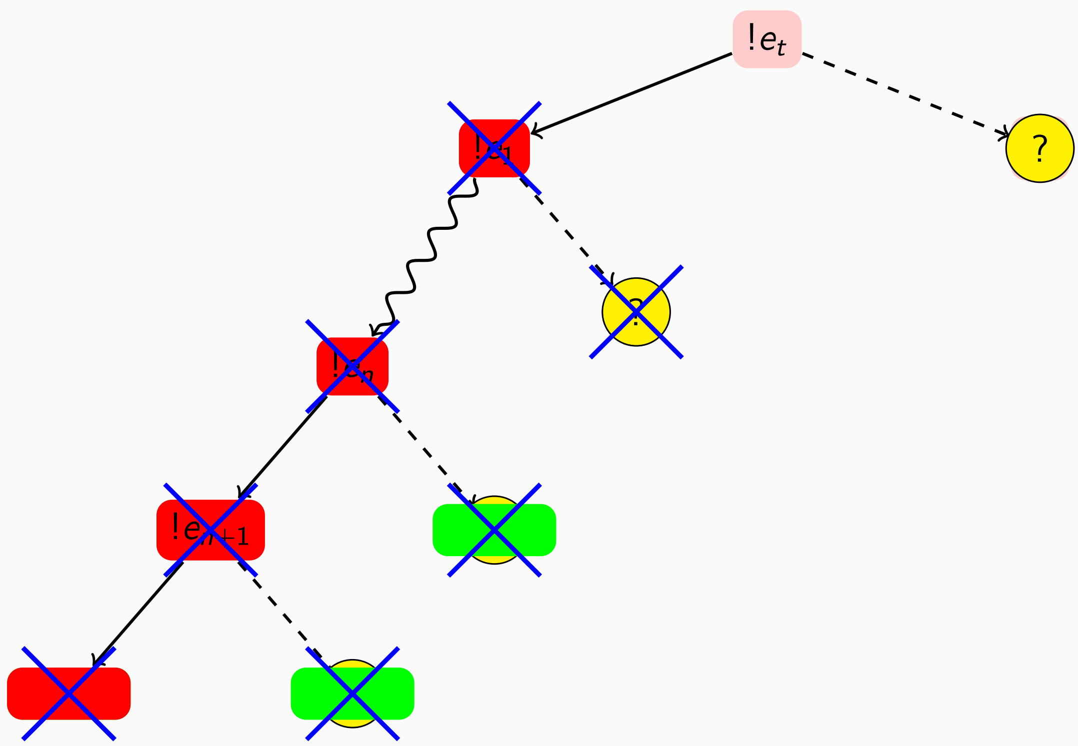
$$\frac{\Delta = \{\psi_i\theta : P_i\theta \mid e' : \psi_i \leftarrow P_i \in \Pi \wedge \theta = \mathrm{mgu}(e, e')\}}{\langle \mathcal{B}, \mathcal{A}, !e\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, (\!|\Delta|\!)\rangle}\ Event$$

$$\frac{\psi_i(\vec{x}) : P_i \in \Delta \quad \mathcal{B} \models \psi_i(\vec{x})\theta}{\langle \mathcal{B}, \mathcal{A}, (\!|\Delta|\!)\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, P_i\theta \rhd (\!|\ (\Delta \setminus \{\psi_i(\vec{x}) : P_i\}) \cup \{\psi_i(\vec{x}) \wedge \vec{x} \neq \theta : P_i\}\ |\!)\rangle}\ Sel$$

$$\frac{\langle \mathcal{B}, \mathcal{A}, P_1\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P'\rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P' \rhd P_2\rangle}\ \rhd \qquad \frac{}{\langle \mathcal{B}, \mathcal{A}, (nil \rhd P_2)\rangle \longrightarrow \langle \mathcal{B}, \mathcal{A}, nil\rangle}\ \rhd_t$$

$$\frac{P_1 \neq nil \quad \langle \mathcal{B}, \mathcal{A}, P_1\rangle \not\longrightarrow \quad \langle \mathcal{B}, \mathcal{A}, P_2\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_2'\rangle}{\langle \mathcal{B}, \mathcal{A}, P_1 \rhd P_2\rangle \longrightarrow \langle \mathcal{B}', \mathcal{A}', P_2'\rangle}\ \rhd_f$$

Failure at Work within the Goal Hierarchy



Some Other Formal BDI Languages

AgentSpeak [LNCS 1996]

- no concurrency in plans; no built-in failure handling.
- special belief updates events $!+b$ and $!-b$.

3APL [Autonomous Agents and Multi-Agent Systems 1999]

- no events, but uses a goal base $\mathcal{G}$;
- different type of (practical) rules in priority: failure rules, reactive rules, plan rules, optimization rules.

$$\pi_h \leftarrow \varphi \mid \pi_b$$

GOAL [Journal of Applied Logic 2007]

- no events, but uses goal base Π.
- two special goal adoption actions: adopt(ϕ) and drop(ϕ).
- plan rules are called *conditional actions*: $\varphi \rhd do(a)$.

$$\mathbf{B}(in\_cart(book) \wedge \mathbf{G}(bought(book)) \rhd do(pay\_cart)$$

Review

In this lecture we have seen:

1. Basic concepts of BDI programming:
   - programming using mentalistic concepts such as beliefs, desires, capabilities, etc.
   - goal-oriented programming – via events;
   - implicit programming – via plan library & context conditions;
   - rational execution cycle: on-the-fly recombination of plans.
2. CAN formal BDI programming language:
   - captures the basic notions of BDI programming: rational executor;
   - formal operational semantics;
   - includes built-in failure handling.

Next Lecture

Declarative Goals in CAN

&

Hierarchical planning in CAN

Next Lecture

In the next lecture we will:

1. Show how to accommodate hierarchical HTN-style planning into CAN.
   - to perform some "offline" look-ahead reasoning within the whole online "reactive" execution scheme.
2. Review the Java-based JACK agent programming language:
   - go over a basic gold-mining agent team implementation.

BDI Formal Languages

Anand S. Rao and Michael P. Georgeff.
An abstract architecture for rational agents.
In *Proceedings of KR-92*, pages 438–449, 1992.

Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer.
Agent programming in 3APL.
Autonomous Agents and Multi-Agent Systems, 2:357–401, 1999.

Frank S. de Boer, Koen V. Hindriks, Wiebe van der Hoek, and John-Jules Ch. Meyer.
A Verification Framework for Agent Programming with Dec. Goals.
Journal of Applied Logic, 5(2):277–302, 2007.

Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah.
Declarative & procedural goals in intelligent agent systems.
In *Proceedings of the KR-02*, 2002.

BDI Systems I

Georgeff and Lansky.
Reactive reasoning and planning.
In *Proceedings of AAAI-87*, pages 677-682, 1987.

Georgeff, M. P. and Ingrand, F. F.
Decision-making in an embedded reasoning system.
In *Proceedings of IJCAI-89*, 972-978, 1989.

F. Ingrand, M. Georgeff, and A Rao.
An architecture for real-time reasoning and system control.
IEEE Expert, 7(6), 1992.

Mark d'Inverno, Michael Luck, Michael Georgeff, , David Kinny, and Michael Wooldridge.
The dMARS architechure: A specification of the distributed multi-agent reasoning system.
Autonomous Agents and Multi-Agent Systems, 9(1–2):5–53, 2004.

BDI Systems II

Huber, Marcus J.
JAM: A BDI-theoretic mobile agent architecture.
In *Proceedings of AGENTS-99*, pages 236–243, 1999.

P. Busetta, Ralph Rönnquist, A. Hodgson, and A. Lucas.
JACK Intelligent Agents: Components for intelligent agents in Java.
AgentLink News Letter, Jan 1999.

R. H. Bordini and J. F. H ubner.
BDI agent programming in AgentSpeak using Jason.
In *Proceedings of the International Workshop on Computational Logic in Multi-Agent Systems (CLIMA)*, pages 143–164, 2006.

Alexander Pokahr, Lars Braubach, Winfried Lamersdorf
Jadex: Implementing a BDI-Infrastructure for JADE Agents.
In *Search of Innovation (Special Issue on JADE)*, 3(5):76–85, Telecom Italia Lab, Turin, Italy, September 2003, 76-85, 2003.