Goals in the Context of BDI Plan Failure and Planning

Sebastian Sardina and Lin Padgham

Department of Computer Science and Information Technology
RMIT University
Melbourne, AUSTRALIA
Motivation

BDI-style **agent-oriented programming** is a novel approach to programming complex systems in highly dynamic environments. These systems are very open to changes in the environment, flexible, and robust.
Motivation

BDI-style agent-oriented programming is a novel approach to programming complex systems in highly dynamic environments. These systems are very open to changes in the environment, flexible, and robust.

They are build around the concept of goals.
Motivation

BDI-style agent-oriented programming is a novel approach to programming complex systems in highly dynamic environments. These systems are very open to changes in the environment, flexible, and robust.

They are build around the concept of goals.

However, the level of support for goals has not been matched with their importance:

- mostly procedural view;
- weak representation and reasoning;
- not well linked with agent-theories.
Motivation

BDI-style agent-oriented programming is a novel approach to programming complex systems in highly dynamic environments. These systems are very open to changes in the environment, flexible, and robust.

They are build around the concept of goals. However, the level of support for goals has not been matched with their importance:

- mostly procedural view;
- weak representation and reasoning;
- not well linked with agent-theories.

What do we here? Improve the handling of goals in CANPlan.
BDI Agent-Oriented Programming

BDI programming ⇒ beliefs, events, plan library, intentions.
(e.g., AgentSpeak/Jason, PRS & dMars, 3APL, Jack, JAM)
BDI Agent-Oriented Programming

BDI programming \(\Rightarrow\) beliefs, events, plan library, intentions. (e.g., AgentSpeak/Jason, PRS & dMars, 3APL, Jack, JAM)

The BDI Execution Cycle [Rao&Georgeff 92]

1. Select an intention and execute a step.
   - Pick a pending event \(e\).
   - Select *relevant* plans from library (match event).
   - Select *applicable* plans from relevant set (match context).

2. Incorporate any pending external events.

3. Update the set of goals and intentions.

4. Repeat cycle.

We will mostly concentrate on step 3 in language CAN(Plan)...
The CAN & CANPlan Languages [Winikoff et al. 02 & Sardina 06]

CAN ≈ AgentSpeak + Goal($\phi_s, P, \phi_f$) + Failure Handling

CANPlan = CAN + HTN-style local planning
Limitations in CAN(Plan)

Declarative goals $\text{Goal}(\phi_s, P, \phi_f) \ldots$

1. are only adopted due to the execution of plans;
Limitations in CAN(Plan)

Declarative goals $\text{Goal}(\phi_s, P, \phi_f)$ ...

1. are only adopted due to the execution of plans;
2. are not forced to terminate when achieved or deemed impossible;
   
   ... depends on corresponding intention being selected.
Limitations in CAN(Plan)

Declarative goals $\text{Goal}(\phi_s, P, \phi_f)$ ...

1. are **only adopted** due to the the execution of plans;
2. are **not forced to terminate** when achieved or deemed impossible;
   ... *depends on corresponding intention being selected.*
3. are **too “fanatic”** on their procedural methods.
   ... $G_2$ is a **blocked sub-goal of $G_1$, but there is an alternative way for achieving $G_1$**.

**Reason:** focus was on the semantics of *individual intentions.*
Enhancing support for goals in CANPlan

1. Declarative goals can be generated proactively.
2. Achieved and "impossible" goals are dropped at every step.
3. Only "reactive" intentions can be dropped when blocked; differentiate event-goals from declarative-goals.
4. Non-working declarative sub-goals may be dropped for the sake of achieving (higher-level) motivating goals; differentiate goals from their sub-goals.

Restriction: Incremental & modular extension to CANPlan.
Enhancing support for goals in CANPlan

1. Declarative goals can be generated pro-actively.
Enhancing support for goals in CANPlan

1. Declarative goals can be generated pro-actively.
2. Achieved and “impossible” goals are dropped at every step.
Enhancing support for goals in CANPlan

1. Declarative goals can be generated pro-actively.
2. Achieved and “impossible” goals are dropped at every step.
3. Only “reactive” intentions can be dropped when blocked; ⇒ differentiate event-goals from declarative-goals.
Enhancing support for goals in CANPlan

1. Declarative goals can be generated pro-actively.
2. Achieved and “impossible” goals are dropped at every step.
3. Only “reactive” intentions can be dropped when blocked; 
   \[\Rightarrow\] differentiate event-goals from declarative-goals.
4. Non-working declarative sub-goals may be dropped for the sake of achieving (higher-level) motivating goals; 
   \[\Rightarrow\] differentiate goals from their sub-goals.
Enhancing support for goals in CANPlan

1. Declarative goals can be generated pro-actively.
2. Achieved and “impossible” goals are dropped at every step.
3. Only “reactive” intentions can be dropped when blocked; 
   \(\Rightarrow\) differentiate event-goals from declarative-goals.
4. Non-working declarative sub-goals may be dropped for the sake of achieving (higher-level) motivating goals; 
   \(\Rightarrow\) differentiate goals from their sub-goals.

Restriction: incremental & modular extension to CANPlan.

\[
\text{CANPlan2} = \text{CANPlan} + \text{Improved handling of goals.}
\]
Summary of changes to CANPlan

- Introduced a new "motivational" library $\mathcal{M}$ to generate goals pro-actively: $\psi \leadsto \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M}$

  \[
  \text{RoomDirty} \land \neg\text{Busy} \leadsto \text{Goal}(\neg\text{RoomDirty}, !\text{clean}, \text{HasWork}).
  \]
Summary of changes to CANPlan

- Introduced a new “motivational” library $\mathcal{M}$ to generate goals pro-actively: $\psi \leadsto \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M}$

  $$\text{RoomDirty} \land \neg \text{Busy} \leadsto \text{Goal}(\neg \text{RoomDirty}, !\text{clean}, \text{HasWork}).$$

- Divided the agent-level semantics into three-phases:
  1. intention execution;
  2. handling of external events & sensor information;
  3. goal update.
Summary of changes to CANPlan

- Introduced a new “motivational” library $\mathcal{M}$ to generate goals pro-actively: $\psi \rightsquigarrow \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M}$

  \begin{align*}
  \text{RoomDirty} & \land \neg \text{Busy} \rightsquigarrow \text{Goal}(\neg \text{RoomDirty}, !\text{clean}, \text{HasWork}).
  \end{align*}

- Divided the agent-level semantics into three-phases:
  1. intention execution;
  2. handling of external events & sensor information;
  3. goal update.

- Introduced 9 new and modified 4 derivation rules.
  $\Rightarrow$ implement Rao’s rational execution cycle.
Theorem 1: Achieved and impossible goals

\[ G(\phi_s, \phi_f) \text{ is active if some intention is executing some } \text{Goal}(\phi_s, P, \phi_f). \]

**Definition (Single-Minded Configuration)**

An agent configuration \( C \) is “single-minded” if for every active declarative goal \( G(\phi_s, \phi_f) \), it is the case that \( B \nmid \phi_s \) and \( B \nmid \phi_f \).

**Theorem**

*The single-minded property is propagated throughout BDI executions.*
Theorem 2: Failure Handling and Goals Hierarchy

Let $G_k$ be an active (sub)goal such that its current strategy is blocked. Then, for every subgoal $G_{k'}$ of $G_k$, either:

1. the current strategy for $G_{k'}$ is blocked and $G_{k'}$ does not have an alternative strategy; or

2. there is a planning (sub)goal $G_p$ “between” $G_k$ and $G_{k'}$ with no solution.
Theorem 2: Failure Handling and Goals Hierarchy (cont.)

\[ G_k \rightarrow G_k' \]
Theorem 2: Failure Handling and Goals Hierarchy (cont.)

current strategy blocked \( G_k \)

\( G_k' \)
Theorem 2: Failure Handling and Goals Hierarchy (cont.)

\[ G_k \rightarrow \text{current strategy blocked} \]

\[ \neg G_{k'} \rightarrow \text{no alternative strategy} \]
Theorem 2: Failure Handling and Goals Hierarchy (cont.)
Theorem 2: Failure Handling and Goals Hierarchy (cont.)

Current strategy blocked: $G_k 

Planning

No solution

$G_k' 

Motivation

The CAN(Plan) Language

CANPlan2 = CANPlan + Goal handling

Conclusions
Theorem 3: Why a goal may be dropped

Assume $C \xrightarrow{\text{agent}} C'$ and that $G_k = G(\phi_s, \phi_f)$ is active in $C$ but not in $C'$. Then, one of the following cases must apply:

1. $B' \models \phi_s$, i.e., the goal has been achieved;
2. $B' \models \phi_f$, i.e., the goal is believed to be impossible; or
3. there is a motivating goal $G_{k'}$ of $G_k$, such that either:
   a. $G_{k'} = \text{Goal}(\phi'_s, P', \phi'_f)$ and $B' \models \phi'_s \lor \phi'_f$;
   b. $G_{k'}$ is a planning goal with no solution; or
   c. $G_k$ is blocked, but $G_{k'}$ has an alternative applicable strategy.
Theorem 3: Why a goal may be dropped

Assume $C \xRightarrow{\text{agent}} C'$ and that $G_k = G(\phi_s, \phi_f)$ is active in $C$ but not in $C'$. Then, one of the following cases must apply:

1. $B' \models \phi_s$, i.e., the goal has been achieved;
2. $B' \models \phi_f$, i.e., the goal is believed to be impossible; or
3. there is a motivating goal $G_{k'}$ of $G_k$, such that either:
   - $G_{k'} = \text{Goal}(\phi_s', P', \phi_f')$ and $B' \models \phi_s' \lor \phi_f'$;
   - $G_{k'}$ is a planning goal with no solution; or
   - $G_k$ is blocked, but $G_{k'}$ has an alternative applicable strategy.
Theorem 3: Why a goal may be dropped (cont.)

\[ \text{dropped} \implies G_k = \text{Goal}(\phi_s, P, \phi_f) \]
Theorem 3: Why a goal may be dropped (cont.)

There is a motivating goal $G_{k'}$ of $G_k$, such that:

\[
\text{dropped} \implies G_k = \text{Goal}(\phi_s, P, \phi_f)
\]
Theorem 3: Why a goal may be dropped (cont.)

There is a motivating goal $G_{k'}$ of $G_k$, such that:

1. $G_{k'} = \text{Goal}(\phi'_s, P', \phi'_f)$ and $\mathcal{B}' \models \phi'_s \lor \phi'_f$
There is a motivating goal $G_{k'}$ of $G_k$, such that:

2. $G_{k'}$ is a planning goal ...
**Theorem 3: Why a goal may be dropped (cont.)**

There is a motivating goal $G_{k'}$ of $G_k$, such that:

2. $G_{k'}$ is a planning goal with no solution.
Theorem 3: Why a goal may be dropped (cont.)

There is a motivating goal $G_{k'}$ of $G_k$, such that:

3. $G_k$ is blocked, but ...

\[ G_k = \text{Goal}(\phi_s, P, \phi_f) \]

\[ G_k' = \text{Goal}(\phi_s', P', \phi_f') \]

\[ \text{dropped} \implies G_k = \text{Goal}(\phi_s, P, \phi_f) \]

\[ B| = \phi_s' \lor \phi_f' \]

\[ G_k' = \text{planning} \]
Theorem 3: Why a goal may be dropped (cont.)

There is a motivating goal $G_k'$ of $G_k$, such that:

3. $G_k$ is blocked, but $G_k'$ has an alternative applicable strategy.
Conclusions

We proved the following results for CANPlan2:

- Agents would not pursue goals that are achieved or deemed unachievable (Theorem 1).
- Agents always respect the hierarchical structure of active goals; subgoals as mere instruments for higher level goals (Theorem 2).
- Characterised the reasons why a declarative goal may be abandoned by an agent (Theorem 3).
Conclusions

We proved the following results for CANPlan2:

- Agents would not pursue goals that are achieved or deemed unachievable (Theorem 1).
- Agents always respect the hierarchical structure of active goals; subgoals as mere instruments for higher level goals (Theorem 2).
- Characterised the reasons why a declarative goal may be abandoned by an agent (Theorem 3).

“Flexible single-minded” =

single-minded + opt. goal reconsideration when blocked

... lays between the simple *single-minded* and the sophisticated *open-minded* commitment strategies.
Limitations and Future Work

1. Goals are adopted without checking for negative or positive interactions with current active goals.

2. Goals and motivations are restricted to achievement goals only.

3. No account for “suspended” goals or intentions.

4. Planning goals are not repaired upon failure.

5. ...

...
Motivation

The CAN(Plan) Language

CONCLUSIONS

For Further Reading: BDI Theory

Michael Bratman.  
*Intentions, Plans, and Practical Reason.*  

Daniel Dennett.  
*The Intentional Stance.*  

Philip R. Cohen and Hector J. Levesque.  
Intention is choice with commitment.  

Modeling rational agents within a BDI-architecture.  
For Further Reading: BDI Formal Languages


For Further Reading: BDI Systems/Architectures

J. Dix, H. Munoz-Avila, and D. Nau
IMPACTing SHOP: Planning in a Multi-Agent Environment.

J. Ambros-Ingerson and S. Steel.
Integrating planning, execution and monitoring.

Olivier Despouys and Francois Felix Ingrand.
Propice-plan: Toward a unified framework for planning and exec.

A Planning Component for RETSINA Agents.

David E. Wilkins and Karen L. Myers.
A Multiagent Planning Architecture.
Agent-level goal derivation rules: \( \text{goal} \)

\[
\psi \rightsquigarrow \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M} \quad \mathcal{B} \models \psi \theta \quad \text{Goal}(\phi_s, P, \phi_f)\theta \notin \Gamma
\]

\[
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \{\text{Goal}(\phi_s, !e, \phi_f)\theta\} \rangle
\]

\[
P \in \Gamma \quad \mathcal{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_s \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\text{bdi}} \langle \mathcal{B}, \mathcal{A}, P' \rangle
\]

\[
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\} \rangle
\]

\[
P \in \Gamma \quad \mathcal{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\text{bdi}} \langle \mathcal{B}, \mathcal{A}, P' \rangle
\]

\[
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\} \rangle
\]
Agent-level goal derivation rules:

\[ \psi \rightsquigarrow \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M} \quad \mathcal{B} \models \psi \theta \quad \text{Goal}(\phi_s, P, \phi_f) \theta \notin \Gamma \]

\[ \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \{ \text{Goal}(\phi_s, !e, \phi_f) \theta \} \rangle \]

\[ P \in \Gamma \quad \text{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_s \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\text{bdi}} \langle \mathcal{B}, \mathcal{A}, P' \rangle \]

\[ \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{ P \}) \cup \{ P' \} \rangle \]

\[ P \in \Gamma \quad \text{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\text{bdi}} \langle \mathcal{B}, \mathcal{A}, P' \rangle \]

\[ \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{ P \}) \cup \{ P' \} \rangle \]
Agent-level goal derivation rules: \( \rightarrow \)

\[
\begin{align*}
\psi \leadsto & \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M} \quad \mathcal{B} \models \psi \theta \quad \text{Goal}(\phi_s, P, \phi_f) \theta \notin \Gamma \\
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\text{goal}}{\Rightarrow} & \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \{ \text{Goal}(\phi_s, !e, \phi_f) \theta \} \rangle \\
A^{1}_{\text{goal}}
\end{align*}
\]

\[
\begin{align*}
P & \in \Gamma \quad G(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_s \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \overset{\text{bdi}}{\rightarrow} \langle \mathcal{B}, \mathcal{A}, P' \rangle \\
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\text{goal}}{\Rightarrow} & \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\} \rangle \\
A^{2}_{\text{goal}}
\end{align*}
\]

\[
\begin{align*}
P & \in \Gamma \quad G(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \overset{\text{bdi}}{\rightarrow} \langle \mathcal{B}, \mathcal{A}, P' \rangle \\
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \overset{\text{goal}}{\Rightarrow} & \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\} \rangle \\
A^{3}_{\text{goal}}
\end{align*}
\]
Agent-level goal derivation rules:

\[ \psi \rightsquigarrow \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M} \quad \mathcal{B} \models \psi \theta \quad \text{Goal}(\phi_s, P, \phi_f) \theta \notin \Gamma \]

1. \[
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \{\text{Goal}(\phi_s, !e, \phi_f) \theta\} \rangle
\]

2. \[
P \in \Gamma \quad \text{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_s \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\text{bd}} \langle \mathcal{B}, \mathcal{A}, P' \rangle + \{G_s\}
\]

3. \[
P \in \Gamma \quad \text{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_f \quad \langle \mathcal{B}, \mathcal{A}, P \rangle \xrightarrow{\text{bd}} \langle \mathcal{B}, \mathcal{A}, P' \rangle + \{G_f\}
\]

Goals in the Context of BDI Plan Failure and Planning, AAMAS-07, Hawaii, USA

Sebastian Sardina
Agent-level goal derivation rules: \( \Rightarrow \)

\[ \begin{align*}
\psi \rightsquigarrow \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M} & \quad \mathcal{B} \models \psi \theta \quad \text{Goal}(\phi_s, P, \phi_f)\theta \notin \Gamma \\
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \{\text{Goal}(\phi_s, !e, \phi_f)\theta\} \rangle
\end{align*} \]

\[ \begin{align*}
P \in \Gamma & \quad \text{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_s \\
\langle \mathcal{B}, \mathcal{A}, P \rangle & \xrightarrow{\text{bdi}} \langle \mathcal{B}, \mathcal{A}, P' \rangle \\
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\} \rangle
\end{align*} \]

\[ \begin{align*}
P \in \Gamma & \quad \text{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_f \\
\langle \mathcal{B}, \mathcal{A}, P \rangle & \xrightarrow{\text{bdi}} \langle \mathcal{B}, \mathcal{A}, P' \rangle \\
\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, (\Gamma \setminus \{P\}) \cup \{P'\} \rangle
\end{align*} \]
Agent-level goal derivation rules: \( \Rightarrow \)

\[ \psi \rightsquigarrow \text{Goal}(\phi_s, !e, \phi_f) \in \mathcal{M} \quad \mathcal{B} \models \psi \theta \quad \text{Goal}(\phi_s, P, \phi_f) \theta \notin \Gamma \]

\[ \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, A, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, A, \Gamma \cup \{ \text{Goal}(\phi_s, !e, \phi_f) \theta \} \rangle \]

\[ P \in \Gamma \quad \text{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_s \quad \langle \mathcal{B}, A, P \rangle \xrightarrow{\text{bdi}} \langle \mathcal{B}, A, P' \rangle \]

\[ \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, A, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, A, (\Gamma \setminus \{ P \}) \cup \{ P' \} \rangle \]

\[ P \in \Gamma \quad \text{G}(\phi_s, \phi_f) \in \mathcal{G}(P) \quad \mathcal{B} \models \phi_f \quad \langle \mathcal{B}, A, P \rangle \xrightarrow{\text{bdi}} \langle \mathcal{B}, A, P' \rangle \]

\[ \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, A, \Gamma \rangle \xrightarrow{\text{goal}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, A, (\Gamma \setminus \{ P \}) \cup \{ P' \} \rangle \]
Agent-level event derivation rules:

Definition (Environment)

An environment is a total function $E : Act^* \rightarrow 2^{Event}$ satisfying the following consistency property: for every sequence of action $A \in Act^*$ and ground belief atom $b$, if $+b \in E(A)$, then $-b \notin E(A)$.

\[ \Gamma' = \{ !e : !e \in E(A) \} \]

$A_{event}^1$

\[ \langle \Lambda, \Pi, M, B, A, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, M, B, A, \Gamma \cup \Gamma' \rangle \]

$A_{event}^2$

\[ +b \in E(A) \quad B \not\models b \]

\[ \langle \Lambda, \Pi, M, B, A, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, M, B \cup \{b\}, A, \Gamma \rangle \]

$A_{event}^3$

\[ -b \in E(A) \quad B \models b \]

\[ \langle \Lambda, \Pi, M, B, A, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, M, B \setminus \{b\}, A, \Gamma \rangle \]
Agent-level event derivation rules:

**Definition (Environment)**

An **environment** is a total function $E : \text{Act}^* \rightarrow 2^{\text{Event}}$ satisfying the following consistency property: for every sequence of action $A \in \text{Act}^*$ and ground belief atom $b$, if $+b \in E(A)$, then $-b \notin E(A)$.

\[ \Gamma' = \{ !e : !e \in E(A) \} \]

\[
\langle \Lambda, \Pi, M, B, A, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, M, B, A, \Gamma \cup \Gamma' \rangle
\]

\[ +b \in E(A) \quad B \not\models b \]

\[
\langle \Lambda, \Pi, M, B, A, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, M, B \cup \{ b \}, A, \Gamma \rangle
\]

\[ -b \in E(A) \quad B \models b \]

\[
\langle \Lambda, \Pi, M, B, A, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, M, B \setminus \{ b \}, A, \Gamma \rangle
\]
Agent-level event derivation rules:

**Definition (Environment)**

An *environment* is a total function $\mathcal{E} : \text{Act}^* \rightarrow 2^{\text{Event}}$ satisfying the following consistency property: for every sequence of action $\mathcal{A} \in \text{Act}^*$ and ground belief atom $b$, if $+b \in \mathcal{E}(\mathcal{A})$, then $-b \notin \mathcal{E}(\mathcal{A})$.

\[\Gamma' = \{ !e : !e \in \mathcal{E}(\mathcal{A}) \} \]

\[\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \cup \Gamma' \rangle\]

\[+b \in \mathcal{E}(\mathcal{A}) \quad \mathcal{B} \not| b\]

\[\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B} \cup \{b\}, \mathcal{A}, \Gamma \rangle\]

\[-b \in \mathcal{E}(\mathcal{A}) \quad \mathcal{B} \models b\]

\[\langle \Lambda, \Pi, \mathcal{M}, \mathcal{B}, \mathcal{A}, \Gamma \rangle \xrightarrow{\text{event}} \langle \Lambda, \Pi, \mathcal{M}, \mathcal{B} \setminus \{b\}, \mathcal{A}, \Gamma \rangle\]
Agent-level intention execution derivation rules:

\[
P \in \Gamma \quad \langle B, A, P \rangle \xrightarrow{\text{bdi}} \langle B', A', P' \rangle \\
\langle \Lambda, \Pi, \mathcal{M}, B, A, \Gamma \rangle \xrightarrow{\text{int}} \langle \Lambda, \Pi, \mathcal{M}, B', A', (\Gamma \setminus \{P\}) \cup \{P'\} \rangle
\]

\[
P \in \Gamma \quad \mathcal{G}(P) = \emptyset \quad \langle B, A, P \rangle \xrightarrow{\text{bdi}} \\
\langle \Lambda, \Pi, \mathcal{M}, B, A, \Gamma \rangle \xrightarrow{\text{int}} \langle \Lambda, \Pi, \mathcal{M}, B, A, \Gamma \setminus \{P\} \rangle
\]

\[A_{\text{int}}^1\]

\[A_{\text{int}}^2\]
Agent-level main derivation rules: 

\[
\frac{\text{Agent}^1}{\frac{C \xrightarrow{\text{int}} C' \quad C' \xrightarrow{\text{event}} \big| C'' \quad C'' \xrightarrow{\text{goal}} \big| C'''}\quad C \xrightarrow{\text{agent}} C'''}
\]

\[
\frac{\text{Agent}^2}{\frac{C \xrightarrow{\text{int}} C \xrightarrow{\text{event}} \big| C' \quad C' \xrightarrow{\text{goal}} \big| C'' \quad C' \xrightarrow{\text{goal}} \big| C'' \quad C \xrightarrow{\text{agent}} C''}
\]
Agents are not objects!

1. An object does not have control over its behavior.
   \[\implies\] *Objects do it for free; agents do it because they want to*

2. Object model says nothing about flexible autonomous behavior.
   \[\implies\] *Reactive, pro-active, flexible, robust.*

3. Standard object model is single-threaded.
   \[\implies\] *Multiagent systems are inherently multi-threaded.*
1. Pick a pending event $e$.
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event $e$ is *external*, create new intention with selected plan.
5. If event $e$ is *internal*, update the corresponding intention.
6. Partially execute some intention (may post internal events). If execution fails, then perform *failure recovery*.
7. Observe the environment for new *external* events.
8. Update the set of goals and intentions.
9. Repeat cycle.
The BDI Execution Cycle [Rao&Georgeff 92]

1. Pick a pending event $e$.
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event $e$ is *external*, create new intention with selected plan.
5. If event $e$ is *internal*, update the corresponding intention.
6. Partially *execute* some intention (may post internal events).
   - If execution fails, then perform *failure recovery*.
7. *Observe* the environment for new *external* events.
8. Update the set of goals and intentions.
9. Repeat cycle.
The BDI Execution Cycle [Rao&Georgeff 92]

1. Pick a pending event $e$.
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event $e$ is *external*, create new intention with selected plan.
5. If event $e$ is *internal*, update the corresponding intention.
6. Partially execute some intention (may post internal events).
   - If execution fails, then perform *failure recovery*.
7. *Observe* the environment for new *external* events.
8. Update the set of goals and intentions.
9. Repeat cycle.
The BDI Execution Cycle [Rao&Georgeff 92]

1. Pick a pending event $e$.
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event $e$ is *external*, create new intention with selected plan.
5. If event $e$ is *internal*, update the corresponding intention.
6. Partially *execute* some intention (may post internal events).
   - If execution fails, then perform *failure recovery*.
7. *Observe* the environment for new *external* events.
8. Update the set of goals and intentions.
9. Repeat cycle.
The BDI Execution Cycle [Rao&Georgeff 92]

1. Pick a pending event $e$.
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event $e$ is *external*, create new intention with selected plan.
5. If event $e$ is *internal*, update the corresponding intention.
6. Partially *execute* some intention (may post internal events).
   - If execution fails, then perform *failure recovery*.
7. *Observe* the environment for new *external* events.
8. Update the set of goals and intentions.
9. Repeat cycle.
The BDI Execution Cycle [Rao&Georgeff 92]

1. Pick a pending event \( e \).
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event \( e \) is *external*, create new intention with selected plan.
5. If event \( e \) is *internal*, update the corresponding intention.
6. Partially *execute* some intention (may post internal events).
   If execution fails, then perform *failure recovery*.
7. *Observe* the environment for new *external* events.
8. Update the set of goals and intentions.
9. Repeat cycle.
The BDI Execution Cycle [Rao&Georgeff 92]

1. Pick a pending event $e$.
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event $e$ is *external*, create new intention with selected plan.
5. If event $e$ is *internal*, update the corresponding intention.
6. Partially *execute* some intention (may post internal events).
   - If execution fails, then perform *failure recovery*.
7. *Observe* the environment for new *external* events.
8. Update the set of goals and intentions.
9. Repeat cycle.
The BDI Execution Cycle [Rao&Georgeff 92]

1. Pick a pending event $e$.
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event $e$ is *external*, create new intention with selected plan.
5. If event $e$ is *internal*, update the corresponding intention.
6. Partially *execute* some intention (may post internal events).
   If execution fails, then perform *failure recovery*.
7. *Observe* the environment for new *external* events.
8. Update the set of goals and intentions.
9. Repeat cycle.
The BDI Execution Cycle [Rao&Georgeff 92]

1. Pick a pending event $e$.
2. Select *relevant* plans from library (match event).
3. Select *applicable* plans from relevant set (match context).
4. If event $e$ is *external*, create new intention with selected plan.
5. If event $e$ is *internal*, update the corresponding intention.
6. Partially *execute* some intention (may post internal events). 
   If execution fails, then perform *failure recovery*.
7. *Observe* the environment for new *external* events.
8. Update the set of goals and intentions.
9. 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 programming.
- Nondeterminism on choosing plans and bindings.
- **No mechanism for doing lookahead for solving choices**
  Generally programmed *explicitly* by the BDI programmer.
Goal-programs: \( \text{Goal}(\phi_s, P, \phi_f) \)

**Example**

\[ \text{Goal}(\neg\text{Hungry}, \neg\text{satisfyHunger}, \neg\text{foodAvailable}). \]

**Goal(\phi_s, P, \phi_f) Properties**

1. If \( \phi_s \) becomes true, terminate successfully.
2. If \( \phi_f \) becomes true, terminate with failure.
3. If \( P \) terminates successfully and \( \phi_s \) is still not true, \( P \) is re-tried.
4. If \( P \) fails and \( \phi_s \) is still not true, \( P \) is re-tried.
The CAN Language: Operational Semantics

Defined in two levels:

1. Basic configurations: \( \langle \mathcal{B}, \mathcal{A}, P \rangle \rightarrow \langle \mathcal{B}', \mathcal{A}', P' \rangle \)

2. Agent level execution: \( \langle \mathcal{N}, \Lambda, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Rightarrow \langle \mathcal{N}, \Lambda, \Pi, \mathcal{B}', \mathcal{A}', \Gamma' \rangle \)

\[
\frac{P \in \Gamma}{\langle \mathcal{N}, \Lambda, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Rightarrow \langle \mathcal{N}, \Lambda, \Pi, \mathcal{B}', \mathcal{A}', \Gamma' \rangle}
\]

\( A_{\text{step}} \)

\[
\frac{\text{e is a new external event}}{\langle \mathcal{N}, \Lambda, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Rightarrow \langle \mathcal{N}, \Lambda, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \cup \{!e\} \rangle}
\]

\( A_{\text{event}} \)

\[
\frac{P \in \Gamma}{\langle \mathcal{N}, \Lambda, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \rangle \Rightarrow \langle \mathcal{N}, \Lambda, \Pi, \mathcal{B}, \mathcal{A}, \Gamma \setminus \{P\} \rangle}
\]

\( A_{\text{clean}} \)
The CAN Language: Operational Semantics

Defined in two levels:

1. Basic configurations: \( \langle B, A, P \rangle \rightarrow \langle B', A', P' \rangle \)

2. Agent level execution: \( \langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B', A', \Gamma' \rangle \)

\[
P \in \Gamma \quad \frac{\langle B, A, P \rangle \rightarrow \langle B', A', P' \rangle}{\langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B', A', (\Gamma \setminus \{P\}) \cup \{P'\} \rangle} \quad A_{step}
\]

\[
e \text{ is a new external event} \\
\frac{\langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B, A, \Gamma \cup \{!e\} \rangle}{A_{event}}
\]

\[
P \in \Gamma \quad \frac{\langle B, A, P \rangle \not\rightarrow}{\langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B, A, \Gamma \setminus \{P\} \rangle} \quad A_{clean}
\]
The CAN Language: Operational Semantics

Defined in two levels:

1. Basic configurations: \( \langle B, A, P \rangle \rightarrow \langle B', A', P' \rangle \)
2. Agent level execution: \( \langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B', A', \Gamma' \rangle \)

\[
P \in \Gamma \quad \langle B, A, P \rangle \rightarrow \langle B', A', P' \rangle
\]
\[
\langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B', A', (\Gamma \setminus \{P\}) \cup \{P'\} \rangle
\]

\( A_{\text{step}} \)

\( e \) is a new external event
\[
\langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B, A, \Gamma \cup \{!e\} \rangle
\]

\( A_{\text{event}} \)

\[
P \in \Gamma \quad \langle B, A, P \rangle \not\rightarrow
\]
\[
\langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B, A, \Gamma \setminus \{P\} \rangle
\]

\( A_{\text{clean}} \)
The CAN Language: Operational Semantics

Defined in two levels:

1. Basic configurations: \( \langle B, A, P \rangle \rightarrow \langle B', A', P' \rangle \)

2. Agent level execution: \( \langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B', A', \Gamma' \rangle \)

\[
P \in \Gamma \quad \langle B, A, P \rangle \rightarrow \langle B', A', P' \rangle \quad A_{step}
\]

\[
\langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B', A', (\Gamma \setminus \{P\}) \cup \{P'\} \rangle
\]

A new external event \( e \):

\[
\langle N, \Lambda, \Pi, B, A, \Gamma \rangle \Rightarrow \langle N, \Lambda, \Pi, B, A, \Gamma \cup \{!e\} \rangle \quad A_{event}
\]

\[
P \in \Gamma \quad \langle B, A, P \rangle \not\rightarrow \quad P \in \Gamma \quad \langle N, \Lambda, \Pi, B, A, \Gamma \setminus \{P\} \rangle \quad A_{clean}
\]
The CAN Language: Operational Semantics

Defined in two levels:

1. Basic configurations: \( \langle B, A, P \rangle \rightarrow \langle B', A', P' \rangle \)

2. Agent level execution: \( \langle N, \Lambda, \Pi, B, A, \Gamma \rangle = \Rightarrow \langle N, \Lambda, \Pi, B', A', \Gamma' \rangle \)

\[
P \in \Gamma \quad \langle B, A, P \rangle \rightarrow \langle B', A', P' \rangle \quad A_{\text{step}}
\]

\[
\langle N, \Lambda, \Pi, B, A, \Gamma \rangle = \Rightarrow \langle N, \Lambda, \Pi, B', A', (\Gamma \setminus \{P\}) \cup \{P'\} \rangle
\]

- \( e \) is a new external event

\[
\langle N, \Lambda, \Pi, B, A, \Gamma \rangle = \Rightarrow \langle N, \Lambda, \Pi, B, A, \Gamma \cup \{!e\} \rangle \quad A_{\text{event}}
\]

\[
P \in \Gamma \quad \langle B, A, P \rangle \not\rightarrow \quad A_{\text{clean}}
\]

\[
\langle N, \Lambda, \Pi, B, A, \Gamma \rangle = \Rightarrow \langle N, \Lambda, \Pi, B, A, \Gamma \setminus \{P\} \rangle
\]
\[ \begin{align*}
!e & \rightarrow (\psi_1 : P_1, \ldots, \psi_n : P_n) \\
& \quad \rightarrow P_i \triangleright (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \xrightarrow{*} \\
& \quad (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n)
\end{align*} \]

\[ \Delta = \{ \psi_i \theta : P_i \theta \mid e' : \psi_i \leftarrow P_i \in \Pi \land \theta = \text{mgu}(e, e') \} \]

\[ \langle B, A, !e \rangle \rightarrow \langle B, A, (\Delta) \rangle \quad \text{Event} \]

\[ \psi_i : P_i \in \Delta \quad B \models \psi_i \theta \]

\[ \langle B, A, (\Delta) \rangle \rightarrow \langle B, A, P_i \theta \triangleright (\Delta \setminus \psi_i : P_i) \rangle \quad \text{Sel} \]

\[ \langle B, A, (\text{nil} \triangleright P_2) \rangle \rightarrow \langle B, A, \text{nil} \rangle \quad \triangleright_t \quad \text{P}_1 \neq \text{nil} \quad \langle B, A, \text{P}_1 \rangle \rightarrow \langle B, A, \text{P}_2 \rangle \quad \triangleright_f \]

\[ \langle B, A, (\text{P}_1 \triangleright P_2) \rangle \rightarrow \langle B, A, P_2 \rangle \quad \triangleright_f \]
The CAN Language: Operational Semantics (cont.)

\[ !e \rightarrow (\psi_1 : P_1, \ldots, \psi_n : P_n) \]

\[ P_i \triangleright (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \rightarrow^* (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \]

\[ \Delta = \{ \psi_i \theta : P_i \theta \mid e' : \psi_i \leftarrow P_i \in \Pi \land \theta = \text{mgu}(e, e') \} \]

\[ \langle B, A, !e \rangle \rightarrow \langle B, A, (\Delta) \rangle \]

\[ \psi_i : P_i \in \Delta \quad B \models \psi_i \theta \]

\[ \langle B, A, (\Delta) \rangle \rightarrow \langle B, A, P_i \theta \triangleright (\Delta \setminus \psi_i : P_i) \rangle \]

\[ \langle B, A, (\text{nil} \triangleright P_2) \rangle \rightarrow \langle B, A, \text{nil} \rangle \]

\[ P_1 \neq \text{nil} \quad \langle B, A, P_1 \rangle \not\rightarrow \langle B, A, (P_1 \triangleright P_2) \rangle \]

\[ \langle B, A, (P_1 \triangleright P_2) \rangle \rightarrow \langle B, A, P_2 \rangle \]
The CAN Language: Operational Semantics (cont.)

\[ !e \rightarrow (\psi_1 : P_1, \ldots, \psi_n : P_n) \rightarrow \]

\[ P_i \triangleright (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \stackrel{\ast}{\rightarrow} \]

\[ (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \]

\[ \Delta = \{ \psi_i \theta : P_i \theta \mid e' : \psi_i \leftarrow P_i \in \Pi \land \theta = \text{mgu}(e, e') \} \]

\[ \langle B, A, !e \rangle \rightarrow \langle B, A, (\Delta) \rangle \]

Event

\[ \psi_i : P_i \in \Delta \quad B \models \psi_i \theta \]

\[ \langle B, A, (\Delta) \rangle \rightarrow \langle B, A, P_i \theta \triangleright (\Delta \setminus \psi_i : P_i) \rangle \]

Sel

\[ \langle B, A, (\text{nil} \triangleright P_2) \rangle \rightarrow \langle B, A, \text{nil} \rangle \]

\[ P_1 \neq \text{nil} \quad \langle B, A, P_1 \rangle \not\rightarrow \]

\[ \langle B, A, (P_1 \triangleright P_2) \rangle \rightarrow \langle B, A, P_2 \rangle \]

\[ \triangleright_f \]
The CAN Language: Operational Semantics (cont.)

\[ !e \rightarrow (\psi_1 : P_1, \ldots, \psi_n : P_n) \]

\[ P_i \triangleright (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \]

\[ \Delta = \{ \psi_i \theta : P_i \theta \mid e' : \psi_i \leftarrow P_i \in \Pi \land \theta = \text{mgu}(e, e') \} \]

\[ \langle B, A, !e \rangle \rightarrow \langle B, A, (\Delta) \rangle \]

**Event**

\[ \psi_i : P_i \in \Delta \quad B \models \psi_i \theta \]

\[ \langle B, A, (\Delta) \rangle \rightarrow \langle B, A, P_i \theta \triangleright (\Delta \setminus \psi_i : P_i) \rangle \]

**Sel**

\[ \langle B, A, (\text{nil} \triangleright P_2) \rangle \rightarrow \langle B, A, \text{nil} \rangle \]

\[ P_1 \neq \text{nil} \]

\[ \langle B, A, P_1 \rangle \rightarrow \langle B, A, (P_1 \triangleright P_2) \rangle \]

\[ \langle B, A, P_2 \rangle \]

**Sel**
The CAN Language: Operational Semantics (cont.)

\[ \text{!}e \rightarrow (\psi_1 : P_1, \ldots, \psi_n : P_n) \rightarrow \]
\[ P_i \triangleright (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \]
\[ (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \]
\[ \Delta = \{ \psi_i \theta : P_i \theta \mid e' : \psi_i \leftarrow P_i \in \Pi \land \theta = \text{mgu}(e, e') \} \]
\[ \langle B, A, !e \rangle \rightarrow \langle B, A, (\Delta) \rangle \]
\[ \psi_i : P_i \in \Delta \quad B \models \psi_i \theta \]
\[ \langle B, A, (\Delta) \rangle \rightarrow \langle B, A, P_i \theta \triangleright (\Delta \setminus \psi_i : P_i) \rangle \]
\[ \langle B, A, (\text{nil} \triangleright P_2) \rangle \rightarrow \langle B, A, \text{nil} \rangle \]
\[ P_1 \neq \text{nil} \quad \langle B, A, P_1 \rangle \rightarrow \]
\[ \langle B, A, (P_1 \triangleright P_2) \rangle \rightarrow \langle B, A, P_2 \rangle \]

Goals in the Context of BDI Plan Failure and Planning, AAMAS-07, Hawaii, USA

Sebastian Sardina
The CAN Language: Operational Semantics (cont.)

\[ \text{!}e \rightarrow ([\psi_1 : P_1, \ldots, \psi_n : P_n]) \rightarrow \]

\[ P_i \triangleright ([\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n]) \rightarrow^* \]

\[ \Delta = \{ \psi_i \theta : P_i \theta \mid e' : \psi_i \leftarrow P_i \in \Pi \land \theta = \text{mgu}(e, e') \} \]

\[ \triangleright \text{Event} \]

\[ \langle B, A, !e \rangle \rightarrow \langle B, A, ([\Delta]) \rangle \]

\[ \psi_i : P_i \in \Delta \quad B \models \psi_i \theta \]

\[ \langle B, A, ([\Delta]) \rangle \rightarrow \langle B, A, P_i \theta \triangleright ([\Delta \setminus \psi_i : P_i]) \rangle \]

\[ \langle B, A, (\text{nil} \triangleright P_2) \rangle \rightarrow \langle B, A, \text{nil} \rangle \]

\[ \triangleright \text{Sel} \]

\[ P_1 \neq \text{nil} \quad \langle B, A, P_1 \rangle \rightarrow \]

\[ \langle B, A, (P_1 \triangleright P_2) \rangle \rightarrow \langle B, A, P_2 \rangle \]

\[ \triangleright \text{f} \]
The CAN Language: Operational Semantics (cont.)

\[
\begin{align*}
!e & \rightarrow (\langle \psi_1 : P_1, \ldots, \psi_n : P_n \rangle) \\
P_i \triangleright (\langle \psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n \rangle) & \rightarrow^* \\
(\langle \psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n \rangle) \\
\Delta = \{ \psi_i \theta : P_i \theta \mid e' : \psi_i \leftarrow P_i \in \Pi \land \theta = \text{mgu}(e, e') \} & \\
\langle B, A, !e \rangle & \rightarrow \langle B, A, \langle \Delta \rangle \rangle \\
\psi_i : P_i \in \Delta & \quad B \models \psi_i \theta \\
\langle B, A, \langle \Delta \rangle \rangle & \rightarrow \langle B, A, P_i \theta \triangleright \langle \Delta \setminus \psi_i : P_i \rangle \rangle \\
\langle B, A, (\text{nil} \triangleright P_2) \rangle & \rightarrow \langle B, A, \text{nil} \rangle \\
P_1 \neq \text{nil} & \quad \langle B, A, P_1 \rangle \nleftrightarrow \rightarrow \langle B, A, (P_1 \triangleright P_2) \rangle \\
\langle B, A, (P_1 \triangleright P_2) \rangle & \rightarrow \langle B, A, P_2 \rangle
\end{align*}
\]
The CAN Language: Operational Semantics (cont.)

\[ e \rightarrow (\psi_1 : P_1, \ldots, \psi_n : P_n) \rightarrow\]

\[ P_i \triangleright (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \rightarrow^*\]

\[ (\psi_1 : P_1, \ldots, \psi_{i-1} : P_{i-1}, \psi_{i+1} : P_{i+1}, \ldots, \psi_n : P_n) \]

\[ \Delta = \{ \psi_i \theta : P_i \theta \mid e' : \psi_i \leftarrow P_i \in \Pi \wedge \theta = \text{mgu}(e, e') \} \]

\[ \langle B, A, !e \rangle \rightarrow \langle B, A, (\Delta) \rangle \]

\[ \psi_i : P_i \in \Delta \quad B \models \psi_i \theta \]

\[ \langle B, A, (\Delta) \rangle \rightarrow \langle B, A, P_i \theta \triangleright (\Delta \setminus \psi_i : P_i) \rangle \]

\[ (\psi_1 : P_1, \ldots, \psi_n : P_n) \rightarrow \langle B, A, nil \triangleright P_2 \rangle \rightarrow \langle B, A, P_1 \triangleright P_2 \rangle \rightarrow \langle B, A, P_2 \rangle \]

Event

Sel

Goals in the Context of BDI Plan Failure and Planning, AAMAS-07, Hawaii, USA  Sebastian Sardina  28 / 28