Reactive Action Programs

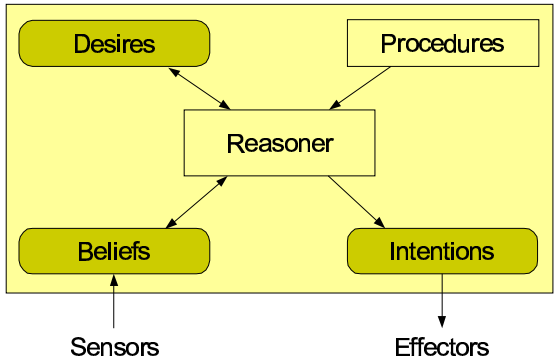
- BDI-Agents
- Programming in AgentSpeak
- AgentSpeak in Prolog
- The SPARK Programming Language

BDI-Agents

The state of an agent at any time is characterized by its

- Beliefs, which constitute the internal world model. They describe what the agent currently believes about the environment and its own tasks.
- Desires, which are derived from the beliefs and describe what the agent currently tries to achieve.
- Intentions, which are the behaviors (procedures) which the agent has adopted and is currently following in order to meet its desires.

Procedural Reasoning Systems (PRS)



Beliefs

- Beliefs are composed of fluents
- Beliefs are affected by actions and sensor information
- Beliefs may include static knowledge (aka domain constraints)
- Beliefs may also concern the "behavioral stance" of the agent

Desires

- Desires are properties which the agent currently wants to achieve
- Desires are (typically short-term) goals

Procedures

- Procedures constitute the behavioral knowledge of the agent
- Procedures correspond to (typically short) GOLOG procedures or ALP clausal definitions
- Each procedure has as condition a desire for which it is suitable

Intentions

- Intentions are instantiated procedures which the agent has selected to meet its desires
- Intentions can be active or passive
- Actions are selected from active intentions
- Intention ordering: Intentions on top are active, all others remain passive until the preceding intentions have been completed

Reasoner

- Controls the execution of a reactive action program
- Sense-Select-Act cycle
  - Sense the environment
  - Select procedures for desires; select an active intention
  - Act according to the selected intention

AgentSpeak

Syntax

Definition 6.2.1

An AgentSpeak domain signature includes a finite set of belief predicates and a finite set of action predicates.

A belief literal is an atom with a belief predicate or its negation.

Example: Gold Mining with Moving Obstacles

Symbol	Type
Adjacent	CELL × CELL
At	{*Agent*, *Gold*, *Obstacle*, *Depot*} × CELL
Carries	{*Agent*} × {*Gold*}

Adjacent(*A*, *B*)
Adjacent(*B*, *C*)
At(*Agent*, *A*)
At(*Gold*, *B*)
At(*Depot*, *C*)

Symbol	Type	Meaning
Pick	{*Gold*}	pick up a gold item
Drop	{*Gold*}	drop a gold item
Move	CELL × CELL	move to an adjacent cell

Triggering Events

Definition 6.2.2

If *f* is a belief atom, then !*f* and ?*f* are goals.

A triggering event is any of

- +*f*
- –*f*
- +g
- –*g*

where *f* belief atom, *g* goal.

Procedures

Definition 6.2.2 (cont'd)

A procedure is an expression

$$e : b_1, \ldots, b_m \leftarrow p_1, \ldots, p_n$$

where

- e triggering event
- context $b_1, \ldots, b_m$ belief literals ($m \geq 0$)
- body $p_1, \ldots, p_n$ action atoms or goals ($n \geq 0$)

Empty context or body denoted by *True*

Example

$+At(Gold, x) : At(Agent, x), At(Depot, y) \leftarrow Pick(Gold), !At(Agent, y), Drop(Gold)$

$+!At(Agent, x) : At(Agent, x) \leftarrow True$

$+!At(Agent, x) : At(Agent, y), x \neq y, Adjacent(y, z), \neg At(Obstacle, z)$
$\leftarrow Move(y, z), !At(Agent, x)$

Operational Semantics

State of AgentSpeak program characterized by triple

- $\mathcal{B}$: a set of variable-free belief atoms
- $\mathcal{I}$: a set of intentions of the form
  $[P_1; \ldots; P_n]$
  where each P_i procedure body (possibly partially instantiated)
- $\mathcal{D}$: a set of desires of the form
  $\langle e; i \rangle$
  where e triggering event, i intention
  External desire: $\langle e; [\,] \rangle$

Selection Functions

- S_D selects an element from the current desires
- S_I selects an element from the current intentions
- S_P selects an applicable procedure for a triggering event

Relevance / Applicability

Definition 6.2.3

$\mathcal{B}$ – set of variable-free belief atoms

e – triggering event

P – procedure $d : b_1, ..., b_m \leftarrow p_1, ..., p_n$

P relevant for e :<=> $d\theta = e\theta$ for some θ

$P\theta\eta$ applicable to e wrt. $\mathcal{B}$:<=> $\mathcal{B} \vdash (b_1 \wedge ... \wedge b_m)\theta\eta$

Example

Adjacent(*A*, *B*)

Adjacent(*B*, *C*)

At(*Agent*, *A*)

At(*Gold*, *A*)

At(*Depot*, *C*)

+!*At*(*Agent*, *x*) : *At*(*Agent*, *x*) ← *True*

+!*At*(*Agent*, *x*) : *At*(*Agent*, *y*), *x* ≠ *y*, *Adjacent*(*y*, *z*), ¬*At*(*Obstacle*, *z*)
← *Move*(*y*, *z*), !*At*(*Agent*, *x*)

Both procedures are relevant for +!*At*(*Agent*, *C*).

The second procedure is also applicable with θ = {*x*/*C*} and η = {*y*/*A*, *z*/*B*}

Derivations: States

$\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \sigma \rangle$ state, where $\sigma \in \{\texttt{Sense}, \texttt{Select}, \texttt{Act}\}$

$\langle \mathcal{B}, \{\}, \{\}, \texttt{Sense} \rangle$ initial state

Derivation Rules (1)

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Sense} \rangle}{\langle \mathcal{B}', \mathcal{D}', \mathcal{I}, \texttt{Select} \rangle}$$

$\mathcal{B}'$:<=> $\mathcal{B}$ updated according to sensing result

$\mathcal{D}'$:<=> $\mathcal{D}$ plus external desires that have been sensed

Derivation Rules (2)

$$\frac{\langle \mathcal{B}, \{\}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \{\}, \mathcal{I}, \texttt{Act} \rangle}$$

If $S_D(\mathcal{D}) = \langle e; i \rangle$ and no relevant procedure exists:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \mathcal{D} \setminus \{\langle e; i \rangle\}, \mathcal{I}, \texttt{Select} \rangle}$$

If $S_D(\mathcal{D}) = \langle e; i \rangle$ and relevant but not applicable procedures exist:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}$$

Derivation Rules (3)

If $S_D(\mathcal{D}) = \langle e; i \rangle$ and $S_P(e) = P\theta\eta$:

(a) For external desires

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \mathcal{D} \setminus \{\langle e; [] \rangle\}, \mathcal{I} \cup \{[P\theta\eta]\}, \texttt{Act} \rangle}$$

(b) For internal desires

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \mathcal{D} \setminus \{\langle e; i \rangle\}, \mathcal{I} \cup \{[P\theta\eta; P_1; \ldots; P_k]\}, \texttt{Act} \rangle}$$

where $i = [P_1, \ldots, P_k]$

Derivation Rules (4)

$$\frac{\langle \mathcal{B}, \mathcal{D}, \{\}, \texttt{Act} \rangle}{\langle \mathcal{B}, \mathcal{D}, \{\}, \texttt{Sense} \rangle}$$

If $S_I(\mathcal{I}) = [a, P_1; \ldots; P_k]$:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}', \mathcal{D}, \mathcal{I} \setminus \{[a, P_1; \ldots; P_k]\} \cup \{[P_1; \ldots; P_k]\}, \texttt{Sense} \rangle}$$

$\mathcal{B}'$:<=> $\mathcal{B}$ updated according to effects of a

Derivation Rules (5)

If $S_I(\mathcal{I}) = [!f, P_1; \ldots; P_k]$:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}, \mathcal{D} \cup \{\langle +!f; [P_1; \ldots; P_k] \rangle\}, \mathcal{I} \setminus \{[!f, P_1; \ldots; P_k]\}, \texttt{Sense} \rangle}$$

If $S_I(\mathcal{I}) = [?f, P_1; \ldots; P_k]$ and $\mathcal{B} \vdash f\theta$ for some θ:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}, \mathcal{D}\mathcal{I} \setminus \{[?f, P_1; \ldots; P_k]\} \cup \{[P_1; \ldots; P_k]\theta\}, \texttt{Sense} \rangle}$$

If $S_I(\mathcal{I}) = [?f, P_1; \ldots; P_k]$ and $\mathcal{B} \nvdash f\theta$ for all θ:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}, \mathcal{D} \cup \{\langle +?f; [P_1; \ldots; P_k] \rangle\}, \mathcal{I} \setminus \{[?f, P_1; \ldots; P_k]\}, \texttt{Sense} \rangle}$$

Derivation Rules (6)

If $S_I(\mathcal{I}) = [True; P_2; ...; P_k]$:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}, \mathcal{D}, \mathcal{I} \setminus \{[True; P_2; ...; P_k]\} \cup \{[P_2; ...; P_k]\}, \texttt{Sense} \rangle}$$

If $S_I(\mathcal{I}) = [\,]$:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}, \mathcal{D}, \mathcal{I} \setminus \{[]\}, \texttt{Sense} \rangle}$$

Example

$\mathcal{B}$ = {*At*(*Agent*, *A*), *At*(*Gold*, *A*), *At*(*Depot*, *C*)}

$\mathcal{D}$ = {⟨+*At*(*Gold*, *A*); [ ]⟩}

$\mathcal{I}$ = { }

σ = `Select`

$\mathcal{B}$ = {*At*(*Agent*, *A*), *At*(*Gold*, *A*), *At*(*Depot*, *C*)}

$\mathcal{D}$ = { }

$\mathcal{I}$ = {[*Pick*(*Gold*), !*At*(*Agent*, *C*), *Drop*(*Gold*)]}

σ = `Act`

$\mathcal{B}$ = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*)}

$\mathcal{D}$ = { }

$\mathcal{I}$ = {[!*At*(*Agent*, *C*), *Drop*(*Gold*)]}

σ = `Sense`

Example (cont'd)

$\mathcal{B}$ = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*)}

$\mathcal{D}$ = { }

$\mathcal{I}$ = {[!*At*(*Agent*, *C*), *Drop*(*Gold*)]}

σ = `Select`

$\mathcal{B}$ = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*)}

$\mathcal{D}$ = { }

$\mathcal{I}$ = {[!*At*(*Agent*, *C*), *Drop*(*Gold*)]}

σ = `Act`

$\mathcal{B}$ = {*At*(*Agent*, *A*), *Carries*(*Agent*, *Gold*), *At*(*Depot*, *C*)}

$\mathcal{D}$ = {⟨+!*At*(*Agent*, *C*); [*Drop*(*Gold*)]⟩}

$\mathcal{I}$ = { }

σ = `Sense`

An AgentSpeak Interpreter

```
desire(E,I) :- procedure(E,P), intention([P|I]).
intention([[do(A)|P]|I]) :- do(A), intention([P|I]).
intention([[!(F)|P]|I]) :- desire(+(!(F)),[P|I]).
intention([[?(F)|P]|I]) :- F, intention([P|I]).
intention([[?(F)|P]|I]) :- desire(+(?(F)),[P|I]).
intention([[]|I]) :- intention(I).
intention([]).
```

Example AgentSpeak Program (1)

```
adjacent(a,b).
adjacent(b,c).
at(agent,a).
at(gold,a).
at(depot,c).

do(pick(gold)) :- at(agent,X), retract(at(gold,X)),
                  assert(carries(agent,gold)).
do(move(X,Y)) :-  at(agent,X), retract(at(agent,X)),
                  assert(at(agent,Y)).
do(drop(gold)) :- at(agent,X), retract(carries(agent,gold)).
```


Example AgentSpeak Program (2)

```
procedure(+(at(gold,X)),P) :-
     at(agent,X), at(depot,Y),
     P = [do(pick(gold)), !(at(agent,Y)), do(drop(gold))].

procedure(+(!(at(agent,X))),P) :- at(agent,X), P=[].

procedure(+(!(at(agent,X))),P) :-
     at(agent,Y), not X=Y, adjacent(Y,Z), not at(obstacle,Z),
     P = [do(move(Y,Z)), !(at(agent,X))].
```


Example: Query `desire(+(at(gold,a)),[])`

```
[[do(pick(gold)), !(at(agent,c)), do(drop(gold))]]
[[!(at(agent,c)), do(drop(gold))]]
[[do(move(a,b)), !(at(agent,c))], [do(drop(gold))]]
[[!(at(agent(c))], [do(drop(gold))]]
[[do(move(b,c)), !(at(agent,c))], [], [do(drop(gold))]]
[[!(at(agent,c))], [], [do(drop(gold))]]
[[], [], [], [do(drop(gold))]]
[[], [], [do(drop(gold))]]
[[], [do(drop(gold))]]
[[do(drop(gold))]]
[[]]
[]
```


Programming in SPARK

The SPARK System

SPARK – Stanford Research Institute Procedural Agent Realization Kit

- Large-scale, practical applications of reactive agent programs
- BDI-model, PRS-architecture (like AgentSpeak)
- More expressive means to control agents in rich and dynamic domains

SPARK Syntax

- Belief literals :<=> belief atoms or their negation
- Actions :<=> primitive actions or names for complex behaviors
- Pre-defined belief atoms are:
  - *Desire*(a) where a action
  - *Success*(a) where a action
  - *Fail*(a) where a action
  - *Desire*(φ) where φ belief literal
  - *Success*(φ) where φ belief literal
  - *Fail*(φ) where φ belief literal

Triggers and Procedures

- Triggers are of the form
  - *Do*(a) where a action
  - *Achieve*(φ) where φ belief literal
  - $+\varphi$ where φ belief literal
- Procedures are of the form

$$e : \phi \leftarrow \tau$$

where e trigger, ϕ formula using belief atoms, τ task description

Programming Language for Task Descriptions

Task	Meaning
noop	do nothing
fail	fail
conclude φ	add the fact to the beliefs
retract φ	delete the fact from the beliefs
do a	perform the action
achieve φ	attempt to achieve φ
seq(τ_1, τ_2)	execute τ_1 then τ_2
if(φ, τ_1, τ_2)	if φ is true, execute τ_1, else τ_2
try(τ, τ_1, τ_2)	if τ succeeds, execute τ_1, else τ_2
wait(φ, τ)	wait until φ is true, then execute τ
while(φ, τ_1, τ_2)	repeat τ_1 until φ has no solution, then execute τ_2

Example Procedures

Do(*ForwardMessage*(*x*)) : ¬*IsSpam*(*x*) ← **try** (**achieve** *ClassifyMessage*(*x*,*y*),
do *AddToFolder*(*x*,*y*),
do *Forward*(*x*)
)

Do(*ForwardMessage*(*x*)) : *IsSpam*(*x*) ← **do** *Delete*(*x*)

Operational Semantics: State Transitions

- Semantics of a task description given by a finite automaton
- Distinguished states in this automaton:
  - s_0 initial state
  - s^+ success state
  - s^- failure state
- Labeled state transitions
  $s \xrightarrow{\vec{c}|\vec{e}} s'$

Conditions and Effects

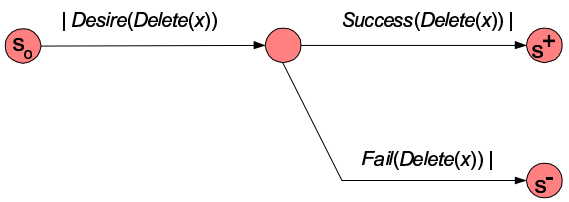
In a labeled state transition $s \xrightarrow{\vec{c}|\vec{e}} s'$, the

- conditions $\vec{c}=c_1,\ldots,c_n$ are
  - $c_i=\varphi$ presence of belief literal φ
  - $c_i=\overline{\varphi}$ absence of belief literal φ
- effects $\vec{e}=e_1,\ldots,e_m$ are
  - $e_i=\varphi$ addition of belief atom φ
  - $e_i=\overline{\varphi}$ removal of belief atom φ

Empty conditions and/or effects are simply omitted

Example (1)

Do(*ForwardMessage*(*x*)) : *IsSpam*(*x*) ← do *Delete*(*x*)



Recursive Construction of State Machine

$\mathcal{M}$(**noop**) :<=> $\{s_0 \rightarrow s^+\}$

$\mathcal{M}$(**fail**) :<=> $\{s_0 \rightarrow s^-\}$

$\mathcal{M}$(**conclude** φ) :<=> $\{s_0 \xrightarrow{|\varphi} s^+\}$

$\mathcal{M}$(**retract** φ) :<=> $\{s_0 \xrightarrow{|\overline{\varphi}} s^+\}$

$\mathcal{M}$(**do** a) :<=> $\{s_0 \xrightarrow{|Desire(a)} s, s \xrightarrow{Success(a)|} s^+, s \xrightarrow{Fail(a)|} s^-\}$

$\mathcal{M}$(**achieve** φ) :<=> $\{ s_0 \xrightarrow{\varphi|} s^+, s_0 \xrightarrow{\overline{\varphi}|Desire(\varphi)} s, s \xrightarrow{Success(\varphi)|} s^+, s \xrightarrow{Fail(\varphi)|} s^-\}$

Construction of State Machine (cont'd)

$\mathcal{M}$(**seq**(τ_1, τ_2)) :<=> $\mathcal{M}(\tau_1)\{s^+/s\} \cup \mathcal{M}(\tau_2)\{s_0/s\}$

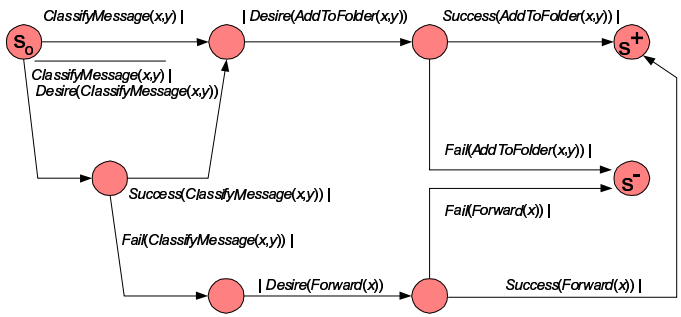
$\mathcal{M}$(**if** (φ, τ_1, τ_2)) :<=> $\varphi \Rightarrow \mathcal{M}(\tau_1) \cup \overline{\varphi} \Rightarrow \mathcal{M}(\tau_2)$

$\mathcal{M}$(**try** (τ, τ_1, τ_2)) :<=> $\mathcal{M}(\tau)\{s^+/s_1, s^-/s_2\} \cup \mathcal{M}(\tau_1)\{s_0/s_1\} \cup \mathcal{M}(\tau_2)\{s_0/s_2\}$

$\mathcal{M}$(**wait** (φ, τ)) :<=> $\{s_0 \longrightarrow s\} \cup \varphi \Rightarrow \mathcal{M}(\tau)\{s_0/s\}$

$\mathcal{M}$(**while** (φ, τ_1, τ_2)) :<=> $\{s_0 \longrightarrow s\} \cup \varphi \Rightarrow \mathcal{M}(\tau_1)\{s_0/s, s^+/s\} \cup \overline{\varphi} \Rightarrow \mathcal{M}(\tau_2)\{s_0/s\}$

Example (2)



Derivations

$\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \sigma \rangle$ state

- $\mathcal{B}$ set of belief literals
- $\mathcal{D}$ set of *Desire*(a), *Desire*(φ), or $+\varphi$
- $\mathcal{I}$ set of triples $\langle d, S, s \rangle$

  - d desire
  - S state machine instance
  - s current state in S
- $\sigma \in \{$`Sense`, `Select`, `Act`$\}$

$\langle \mathcal{B}, \{\}, \{\},$ `Sense`$\rangle$ initial state

Selection Functions

- S_D selects an element from the current desires
- S_P selects an applicable procedure for a desire
- S_I selects an element $\langle d, S, s_1 \rangle \in \mathcal{I}$ with an allowed state transition $s_1 \xrightarrow{d[\vec{e}]} s_2$

Derivation Rules (1)

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Sense} \rangle}{\langle \mathcal{B}', \mathcal{D}', \mathcal{I}, \texttt{Select} \rangle}$$

$\mathcal{B}'$:<=> $\mathcal{B}$ updated according to sensing result

$\mathcal{D}'$:<=> $\mathcal{D}$ plus all sensed desires

Derivation Rules (2)

$$\frac{\langle \mathcal{B}, \{\}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \{\}, \mathcal{I}, \texttt{Act} \rangle}$$

If $S_D(\mathcal{D}) = Desire(x)$ and there is applicable procedure for $Do(a)$ (if $x = a$) or $Achieve(\varphi)$ (if $x = \varphi$)

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \mathcal{D} \setminus \{Desire(x)\}, \mathcal{I} \cup \{\langle Desire(x), S, s_0 \rangle\}, \texttt{Act} \rangle}$$

where S state machine from $S_P(Desire(x))$

If $S_D(\mathcal{D}) = Desire(x)$ and no procedure applicable:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \mathcal{D} \setminus \{Desire(x)\}, \mathcal{I} \cup \{\langle Desire(x), S, s_0 \rangle\}, \texttt{Act} \rangle}$$

where $S = \{s_0 \xrightarrow{!Fail(x)} s^+\}$

Derivation Rules (3)

If $S_D(\mathcal{D}) = +\varphi$ and there is applicable procedure:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \mathcal{D} \setminus \{+\varphi\}, \mathcal{I} \cup \mathcal{I}', \texttt{Act} \rangle}$$

where $\mathcal{I}'$:<=> all $\langle +\varphi, S, s_0 \rangle$ for applicable procedures

If $S_D(\mathcal{D}) = +\varphi$ and no applicable procedure:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Select} \rangle}{\langle \mathcal{B}, \mathcal{D} \setminus \{+\varphi\}, \mathcal{I}, \texttt{Select} \rangle}$$

Derivation Rules (4)

$$\frac{\langle \mathcal{B}, \mathcal{D}, \{\}, \texttt{Act} \rangle}{\langle \mathcal{B}, \mathcal{D}, \{\}, \texttt{Sense} \rangle}$$

If $S_i(\mathcal{I})$ is $\langle d, S, s_1 \rangle$ along with $s_1 \xrightarrow{\bar{c}|\vec{e}} s_2$:

a) If $s_2 \notin \{s^+, s^-\}$

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}', \mathcal{D}', \mathcal{I} \setminus \{\langle d, S, s_1 \rangle\} \cup \{\langle d, S, s_2 \rangle\}, \texttt{Sense} \rangle}$$

b) If $d = +\varphi$ and $s_2 \in \{s^+, s^-\}$:

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}', \mathcal{D}', \mathcal{I} \setminus \{\langle d, S, s_1 \rangle\}, \texttt{Sense} \rangle}$$

Derivation Rules (5)

c) If $d = \mathit{Desire}(x)$ and $s_2 = s^+$

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}' \cup \{\mathit{Success}(x)\}, \mathcal{D}', \mathcal{I} \setminus \{\langle d, S, s_1 \rangle\}, \texttt{Sense} \rangle}$$

d) If $d = \mathit{Desire}(x)$ and $s_2 = s^-$

$$\frac{\langle \mathcal{B}, \mathcal{D}, \mathcal{I}, \texttt{Act} \rangle}{\langle \mathcal{B}' \cup \{\mathit{Fail}(x)\}, \mathcal{D}', \mathcal{I} \setminus \{\langle d, S, s_1 \rangle\}, \texttt{Sense} \rangle}$$