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

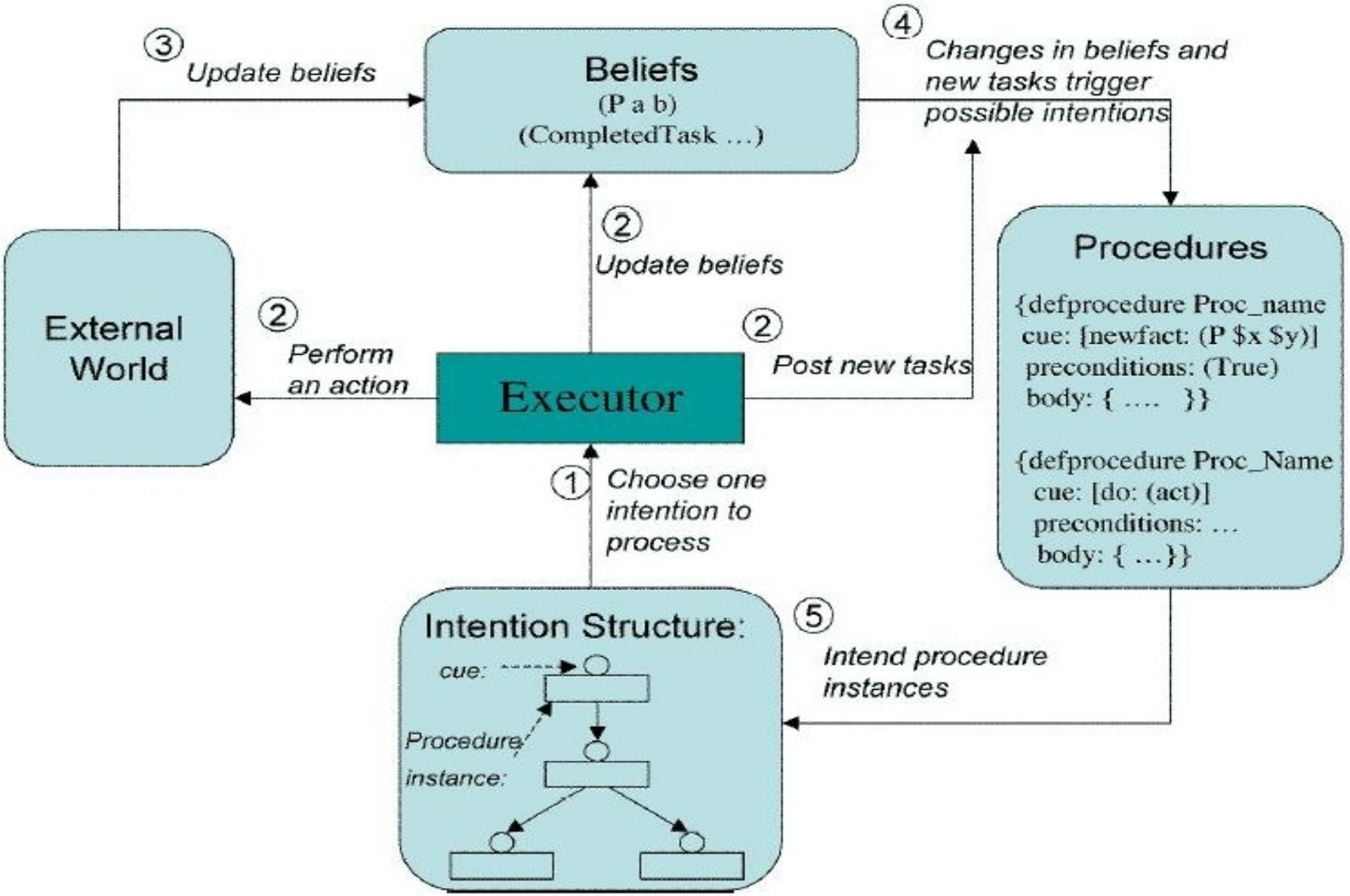
Task	Meaning
noop	do nothing
fail	fail
conclude φ	add the fact to the beliefs
retract φ	delete the fact from the beliefs
do a	perform the action
achieve φ	attempt to achieve φ
seq(τ_1, τ_2)	execute τ_1 then τ_2
if$(\varphi, \tau_1, \tau_2)$	if φ is true, execute τ_1, else τ_2
try(τ, τ_1, τ_2)	if τ succeeds, execute τ_1, else τ_2
wait(φ, τ)	wait until φ is true, then execute τ
while$(\varphi, \tau_1, \tau_2)$	repeat τ_1 until φ has no solution, then execute τ_2

Example Procedures

$Do(ForwardMessage(x)) : \neg IsSpam(x) \leftarrow$ **try** (**achieve** $ClassifyMessage(x,y)$,
do $AddToFolder(x,y)$,
do $Forward(x)$
)

$Do(ForwardMessage(x)) : IsSpam(x) \leftarrow$ **do** $Delete(x)$

SPARK Interpreter Loop



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

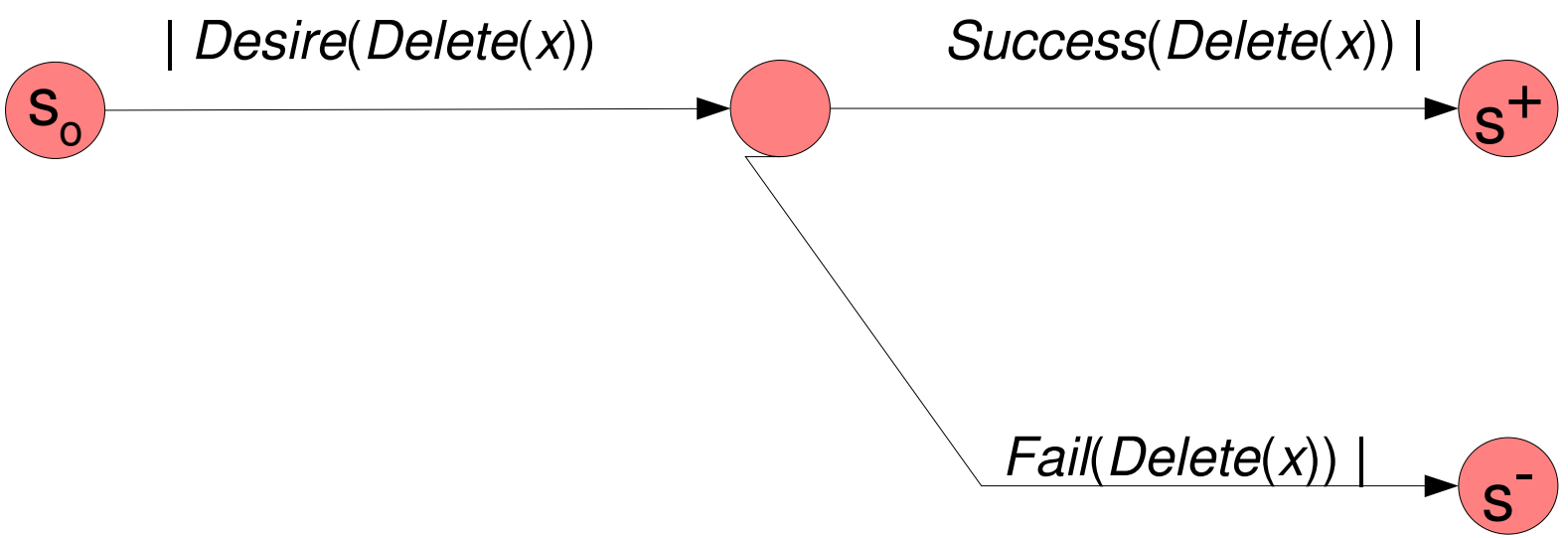
- conditions $\vec{c} = c_1, \ldots, c_n$ are
  - $c_i = \varphi$ presence of belief literal φ
  - $c_i = \overline{\varphi}$ absence of belief literal φ

- 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

$$\mathrm{M}(\textbf{noop}) :\!<\!=\!> \{s_0 \rightarrow s^+\}$$

$$\mathrm{M}(\textbf{fail}) :\!<\!=\!> \{s_0 \rightarrow s^-\}$$

$$\mathrm{M}(\textbf{conclude}\ \varphi) :\!<\!=\!> \{s_0 \overset{|\varphi}{\rightarrow} s^+\}$$

$$\mathrm{M}(\textbf{retract}\ \varphi) :\!<\!=\!> \{s_0 \overset{|\overline{\varphi}}{\rightarrow} s^+\}$$

$$\mathrm{M}(\textbf{do}\ a) :\!<\!=\!> \{s_0 \overset{|\mathit{Desire}(a)}{\rightarrow} s,\ s \overset{\mathit{Success}(a)|}{\rightarrow} s^+,\ s \overset{\mathit{Fail}(a)|}{\rightarrow} s^-\}$$

$$\mathrm{M}(\textbf{achieve}\ \varphi) :\!<\!=\!> \{\ s_0 \overset{\varphi|}{\rightarrow} s^+,\ s_0 \overset{\overline{\varphi}|\mathit{Desire}(\varphi)}{\rightarrow} s,\ s \overset{\mathit{Success}(\varphi)|}{\rightarrow} s^+,\ s \overset{\mathit{Fail}(\varphi)|}{\rightarrow} s^-\}$$

Construction of State Machine (cont'd)

$$M(\textbf{seq}(\tau_1, \tau_2)) \;:\!<\!=\!>\; M(\tau_1)\{s^+/s\} \cup M(\tau_2)\{s_0/s\}$$

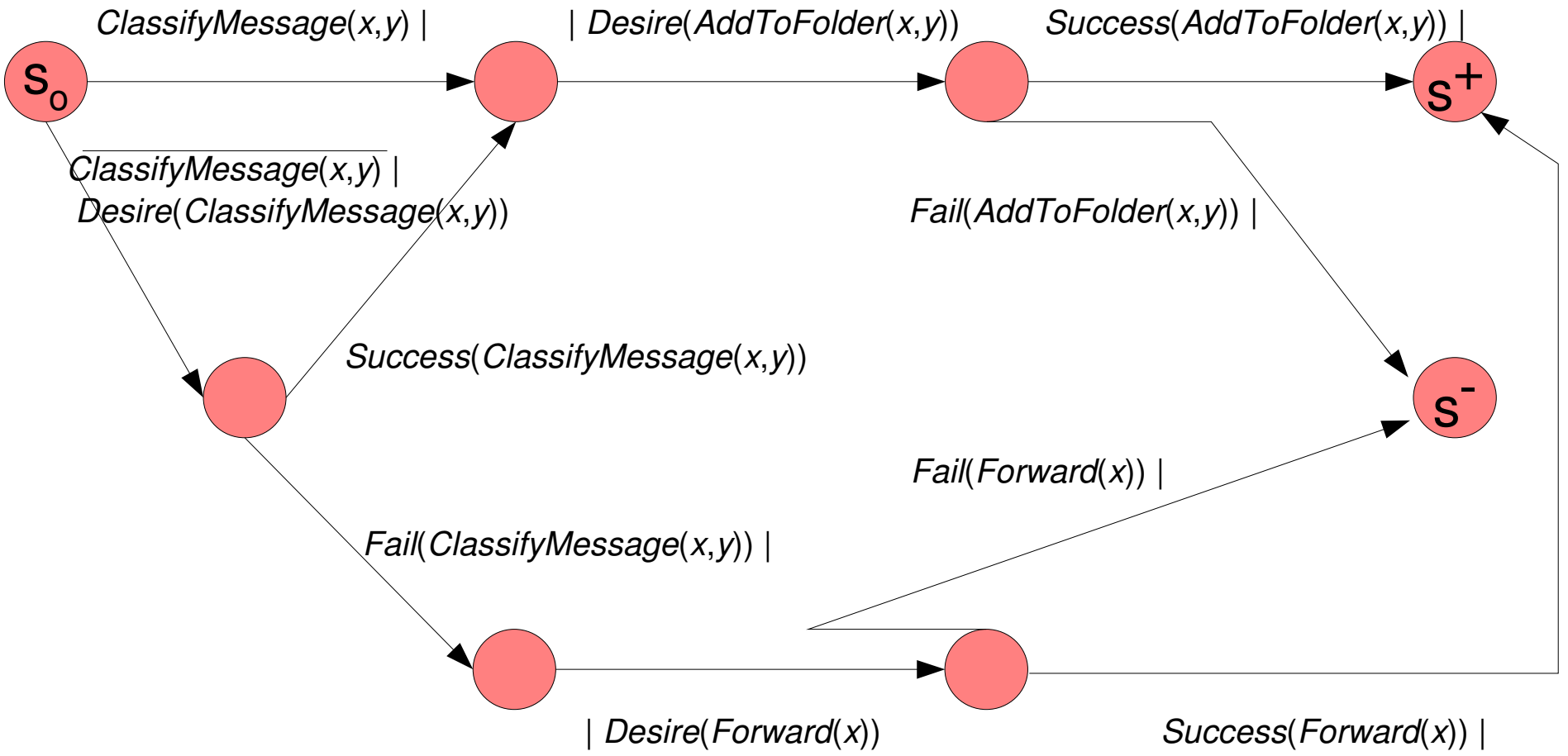
$$M(\textbf{if}\ (\varphi, \tau_1, \tau_2)) \;:\!<\!=\!>\; \varphi \Rightarrow M(\tau_1) \cup \overline{\varphi} \Rightarrow M(\tau_2)$$

$$M(\textbf{try}\ (\tau, \tau_1, \tau_2)) \;:\!<\!=\!>\; M(\tau)\{s^+/s_1, s^-/s_2\} \cup M(\tau_1)\{s_0/s_1\} \cup M(\tau_2)\{s_0/s_2\}$$

$$M(\textbf{wait}\ (\varphi, \tau)) \;:\!<\!=\!>\; \{s_0 \rightarrow s\} \cup \varphi \Rightarrow M(\tau)\{s_0/s\}$$

$$M(\textbf{while}\ (\varphi, \tau_1, \tau_2)) \;:\!<\!=\!>\; \{s_0 \rightarrow s\} \cup \varphi \Rightarrow M(\tau_1)\{s_0/s, s^+/s\} \cup \overline{\varphi} \Rightarrow M(\tau_2)\{s_0/s\}$$

Example (2)


ClassifyMessage(x,y) |
| Desire(AddToFolder(x,y))
Success(AddToFolder(x,y)) |
s_o
s^+
$\overline{ClassifyMessage(x,y)}$ |
Desire(ClassifyMessage(x,y))
Fail(AddToFolder(x,y)) |
Success(ClassifyMessage(x,y))
s^-
Fail(Forward(x)) |
Fail(ClassifyMessage(x,y)) |
| Desire(Forward(x))
Success(Forward(x)) |

Derivations

⟨B, D, I, σ⟩ state

- B set of belief literals
- D set of *Desire*(*a*), *Desire*(φ), or +φ
- I set of triples ⟨*d*, *S*, *s*⟩
  - *d* desire
  - *S* state machine instance
  - *s* current state in *S*
- $\sigma \in$ {`Sense`, `Select`, `Act`}

⟨B, {}, {}, `Sense`⟩ 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 $\langle d, S, s_1 \rangle \in \mathbf{I}$ with an allowed state transition $s_1 \xrightarrow{\vec{c}|\vec{e}} s_2$

Derivation Rules (1/5)

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

B ' :<=> B updated according to sensing result

D ' :<=> D plus all sensed desires

Derivation Rules (2/5)

$$\frac{\langle \mathsf{B}, \{\}, \mathsf{I}, \mathtt{Select} \rangle}{\langle \mathsf{B}, \{\}, \mathsf{I}, \mathtt{Act} \rangle}$$

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

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

where S state machine from $S_P(\mathit{Desire}(x))$

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

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

where $S = \{s_0 \overset{|\mathit{Fail}(x)}{\rightarrow} s^+\}$

Derivation Rules (3/5)

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

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

where I′ :<=> all $\langle +\varphi, S, s_0 \rangle$ for applicable procedures

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

$$\frac{\langle \mathrm{B}, \mathrm{D}, \mathrm{I}, \mathit{Select} \rangle}{\langle \mathrm{B}, \mathrm{D} \setminus \{+\varphi\}, \mathrm{I}, \mathit{Select} \rangle}$$

Derivation Rules (4/5)

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

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

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

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

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

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

Derivation Rules (5/5)

c) If $d = Desire(x)$ and $s_2 = s^+$

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

d) If $d = Desire(x)$ and $s_2 = s^-$

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

Programming in SPARK: Packages

- Package statement `package: sri.foo`
- Declarations
- Definitions
- Additional facts/beliefs for the knowledge base
- Import statements
```
importfrom: spark.lang.list Member
importall: spark.lang.list
```
- Export statements `export: my_sym`
- File requirements `requires: foo.bar`
- Comments `# comment`

Programming in SPARK: Declarations

- Variables `$foo`
- Strings `“this is a string“`
- Lists `[item1 item2 item3]`
- Actions

```
{defaction (_myPrivateAction +$var1 $var2)
   doc: “_myPrivateAction is local to this file“}
```

- Predicates

```
{defpredicate (MyPred $var1 $var2)
   doc: “tests whether $var1 and $var2 are related“}
```

- Functions

```
{deffunction (myFun $var1 $var2)
   doc: “applies myFun to $var1 and $var2“}
```

- Constants `{defconstant person23}`

Procedures (Example)

```
{defprocedure paintHouse_havePaint
   cue: [do: (paintHouse $color)]
   precondition: (HavePaint $color)
   body: [seq: [do: (placeDropCloths)]
               [do: (applyPaintCoat $color)]
               [do: (cleanUp)]
         ]
}
```

Procedures: Cue

A procedure is relevant when its cue event occurs.
There are 3 types of cues:

- Direct action request `[do: (action_identifier variable*)]`
- New fact `[newfact: (Happy $person)]`
- Achievement `[achieve: (Happy $person)]`

- Indicating argument mode for calling a procedure:

```
[do: (paintHouse +$color)]
[do: (paintHouse +“blue“)]

[do: (paintHouse –“blue“)]
```

Procedures: Precondition

A procedure is applicable when its precondition logical expression is true.
Logical expressions are

- any declared predicate
- `(True)` or `(False)`
- Logically connected expressions

```
(and (> 4 $x) (!= $x 2) $person))
(or (AtHome $person) (AtWork $person) (AtSchool $person))
```

- Existential quantifier `(exists [$person] (AtHome $person))`

Procedures: Body (1)

- Actions `[do: (applyPaint wall red)]`
- Achievements `[achieve: (ColorOf wall red)]`
- Noop `[succeed:]` or `[]`
- Failure `[fail: outOfPaintError]`
- Variable binding
```
[context: (HasParent $x $y)]
[set: $x (+ 3 4)]
```
- Sequence
```
[seq: [context: (and (MyFavoriteColor $col1)
                     (YourFavoriteColor $col2))]
      [do: (applyPaint wall $col1)]
      [do: (applyPaint door $col2)]
```
- Parallel
```
[parallel: [do: (rubStomach Bob)]
           [do: (patHead Bob)]]
```

Procedures: Body (2)

- Select
```
[select: (InCar Bob) [do: (drive Bob home)]
         (True)      [do: (talkOnCellPhone Bob)]]
```
- Wait
```
[wait: (AtHome Bob) [do: (talkOnCellPhone Bob)]]
```
- Loop
```
[while: [$x] (and (MyPet $x) (Hungry $x))
             [do: (feed $x)]]
```
- Find all solutions
```
[forall: [$p] (InCar $p)
              [do: (drive $p home)]]
```

Procedures: Body (3) – Special Effects

- Add to knowledge base `[conclude: (Happy Bill)]`
- Remove from knowledge base
```
[retract: (Scared Bill)]
[retractall: [$p] (Scared $p)]
```
- Remove
```
[while: [$x] (and (MyPet $x) (Hungry $x))
             [do: (feed $x)]]
```