Appendix D

Built-in Methods

This Appendix describes the collection of all Noos built-in methods. The goal of this Appendix is to specify the inference involved in the evaluation of each built-in method. We will not provide examples of their use in Noos. Each built-in method is described in three parts:

1. First, we will describe its required features indicating the least required sort for each feature, and the least sort of the value yielded in the evaluation of the method. We will use the following format:

   $$\textbf{method-name}[par1 \doteq s1 \ \cdots \ parn \doteq sn] \longrightarrow s$$

   where **method-name** is the name of the built-in method; $par1, \ldots, parn$ are the names of the required features for the method; $s1, \ldots, sn$ are the least required sorts of values for each required feature; and s is the least sort of the value yielded in the evaluation of the method.

   When least required sorts for feature values are not preserved, an error is produced.

2. Then, a brief description of the evaluation of the method will be provided. We will talk about feature values using their *pari* names. For instance, *item1*<*item2* means that value of feature *item1* is lower to value of feature *item2*.

3. Finally, the corresponding inference rule, using DDL notation, will be introduced.

D.1 General

Conditional
$[\mathit{condition} \doteq \mathit{boolean} \quad \mathit{result} \doteq \{\mathit{any}\} \quad \mathit{otherwise} \doteq \{\mathit{any}\}] \longrightarrow \{\mathit{any}\}$

The **conditional** method performs first the subtask *condition* and depending on its result being *true* or *false*, either the *result* subtask or the *otherwise* subtask is performed, and its result is the value yielded by conditional method.

Inference rules δ_m^{cond} performing the evaluation of a **conditional** method m are the following:

$$\delta_m^{cond} = \frac{\begin{array}{c} condition \doteq true \\ result \doteq c' \end{array}}{result(m) \doteq c'}, \qquad \delta_m^{cond} = \frac{\begin{array}{c} condition \doteq false \\ otherwise \doteq c' \end{array}}{result(m) \doteq c'}$$

Sequence $[] \longrightarrow \{any\}$

The **sequence** method allows the definition of a sequential chaining of subtasks (relatively to the writing order) and its result is the value yielded by the last subtask. **Sequence** has no required features but, at least, one subtask has to be defined. The names of the subtasks can be any feature name.

Since the name of the subtasks are not predetermined, given a specific **sequence** method m with subtasks $t1, \ldots, tn$, the evaluation of m is formalized as the sequence of task programs $\pi_{m.t1}; \ldots; \pi_{m.tn}$ (see Section 5.13.4) followed by an inference rule δ_m^{seq} as follows:

$$\delta_m^{seq} = \frac{tn \doteq c}{result(m) \doteq c}$$

D.2 Comparison methods

Noos provides three different comparison methods for testing equality and inclusion relationships between of two feature terms. The **identity?** method is the most specific, the **subsumption** method is the most general, and the **equivalence** method is more general than **identity?** and more specific than **subsumption**.

Identity? $[item1 \doteq any \quad item2 \doteq any] \longrightarrow boolean$

The **identity?** method compares if the values of features *item1* and *item2* are the same yielding as result *true* when are the same and *false* otherwise.

Inference rules $\delta_m^{id?}$ performing the evaluation of an **identity?** method m are the following:

$$\delta_m^{id?} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_1 \end{array}}{result(m) \doteq true}, \qquad \delta_m^{id?} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_2 \end{array}}{result(m) \doteq false}$$

Subsumption $[source \doteq \{any\} \quad pattern \doteq any] \longrightarrow boolean$

The **subsumption** method checks if *pattern* subsumes *source* yielding as result *true* when *pattern* subsumes *source* and *false* otherwise (see Section 5.6).

Inference rules δ_m^{subsum} performing the evaluation of a **subsumption** method m are the following:

$$\delta_m^{subsum} = \frac{\begin{array}{c} source \doteq c_1 \\ pattern \doteq c_2 \\ \text{“}c_1 \sqsubseteq c_2\text{”} \end{array}}{result(m) \doteq true}, \qquad \delta_m^{subsum} = \frac{\begin{array}{c} source \doteq c_1 \\ pattern \doteq c_2 \\ \text{“}c_1 \not\sqsubseteq c_2\text{”} \end{array}}{result(m) \doteq false}$$

Equivalence $[item1 \doteq any \quad item2 \doteq any] \longrightarrow boolean$

The **equivalence** method checks if *item1* subsumes *item2* and *item2* subsumes *item1* yielding as result *true* when this is the case and *false* otherwise.

Inference rules δ_m^{equiv} performing the evaluation of an **equivalence** method m are the following:

$$\delta_m^{equiv} = \frac{\begin{array}{c} source \doteq c_1 \\ pattern \doteq c_2 \\ \text{“}c_1 \sqsubseteq c_2 \wedge c_2 \sqsubseteq c_1\text{”} \end{array}}{result(m) \doteq true}, \qquad \delta_m^{equiv} = \frac{\begin{array}{c} source \doteq c_1 \\ pattern \doteq c_2 \\ \text{“}c_1 \not\sqsubseteq c_2 \vee c_2 \not\sqsubseteq c_1\text{”} \end{array}}{result(m) \doteq false}$$

D.3 Filter methods

Subsumption-matching $[pattern \doteq any \quad sources \doteq \{any\}] \longrightarrow any$

The **subsumption-matching** method yields an element of the set in feature *sources* such that is subsumed by *pattern*. Backtracking on a **subsumption-matching** method yields, consecutively, all other elements from *sources* also subsumed by *pattern* (see next).

Filter-by-subsumption $[pattern \doteq any \quad sources \doteq \{any\}] \longrightarrow \{any\}$

The **filter-by-subsumption** method yields the subset of elements of *sources* such that are subsumed by *pattern*.

The **filter-by-subsumption** method is defined using the **subsumption-matching** method and by refinement of the **all-eval** method as follows:

```
(define (All-Eval Filter-By-Subsumption)
  (pattern )
  (sources )
  (methods (define (Subsumption-Matching)
             (pattern (>> pattern))
             (sources (>> sources))))
```

D.4 Arithmetic methods

All the arithmetic methods require that the values of their required features be numbers. A non-number value in a feature produces an error. Moreover, each method works on all types of numbers and automatically performs any required coercions when parameters are of different types in the same way that CommonLisp.

Add $[item1 \doteq number \quad item2 \doteq number] \longrightarrow number$

The **add** method yields the sum of *item1* and *item2*.

The inference rule δ_m^{add} performing the evaluation of an **add** method m is the following:

$$\delta_m^{add} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_2 \\ \text{``}c' = c_1 + c_2\text{''} \end{array}}{result(m) \doteq c'}$$

Subtract $[amount \doteq number \quad minus \doteq number] \longrightarrow number$

The **subtract** method yields the subtraction of *amount* and *minus*.

The inference rule δ_m^{sub} performing the evaluation of a **subtract** method m is the following:

$$\delta_m^{sub} = \frac{\begin{array}{c} amount \doteq c_1 \\ minus \doteq c_2 \\ \text{``}c' = c_1 - c_2\text{''} \end{array}}{result(m) \doteq c'}$$

Mult $[item1 \doteq number \quad item2 \doteq number] \longrightarrow number$

The **mult** method yields the product of *item1* and *item2*.

The inference rule δ_m^{mult} performing the evaluation of a **mult** method m is the following:

$$\delta_m^{mult} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_2 \\ \text{``}c' = c_1 * c_2\text{''} \end{array}}{result(m) \doteq c'}$$

Div $[item1 \doteq number \quad item2 \doteq number] \longrightarrow number$

The **div** method yields the division of *item1* by *item2*.

The inference rule δ_m^{div} performing the evaluation of a **div** method m is the following:

$$\delta_m^{div} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_2 \\ \text{“}c' = c_1/c_2\text{”} \end{array}}{result(m) \doteq c'}$$

Max $[item1 \doteq number \quad item2 \doteq number] \longrightarrow number$

The **max** method yields the greatest number of *item1* and *item2*.

Inference rules δ_m^{max} performing the evaluation of a **max** method m are the following:

$$\delta_m^{max} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_2 \\ \text{“}c_1 > c_2\text{”} \end{array}}{result(m) \doteq c_1}, \qquad \delta_m^{max} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_2 \\ \text{“}c_1 \leq c_2\text{”} \end{array}}{result(m) \doteq c_2}$$

Min $[item1 \doteq number \quad item2 \doteq number] \longrightarrow number$

The **min** method yields the least number between *item1* and *item2*.

Inference rule δ_m^{min} performing the evaluation of a **min** method m are the following:

$$\delta_m^{max} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_2 \\ \text{“}c_1 < c_2\text{”} \end{array}}{result(m) \doteq c_1}, \qquad \delta_m^{max} = \frac{\begin{array}{c} item1 \doteq c_1 \\ item2 \doteq c_2 \\ \text{“}c_1 \geq c_2\text{”} \end{array}}{result(m) \doteq c_2}$$

D.4.1 Numeric comparisons

Higher-than? $[\textit{is-higher} \doteq number \quad than \doteq number] \longrightarrow boolean$

The **higher-than?** method yields *true* if *is-higher* is higher than *than*. Otherwise yields *false*.

Inference rules $\delta_m^{>}$ performing the evaluation of a **higher-than?** method m are the following:

$$\delta_m^{>} = \frac{\begin{array}{c} \textit{is-higher} \doteq c_1 \\ than \doteq c_2 \\ \text{“}c_1 > c_2\text{”} \end{array}}{result(m) \doteq true}, \qquad \delta_m^{>} = \frac{\begin{array}{c} \textit{is-higher} \doteq c_1 \\ than \doteq c_2 \\ \text{“}c_1 \leq c_2\text{”} \end{array}}{result(m) \doteq false}$$

Higher-equal-than? $[\textit{is-higher} \doteq number \quad than \doteq number] \longrightarrow boolean$

The **higher-equal-than?** method yields *true* if *is-higher* is higher or equal than *than*. Otherwise yields *false*.

Inference rules $\delta_m^{\geq}$ performing the evaluation of a **higher-equal-than?** method m are the following:

$$\delta_m^{\geq} = \frac{\begin{array}{c} \textit{is-higher} \doteq c_1 \\ \textit{than} \doteq c_2 \\ \text{“}c_1 \geq c_2\text{”} \end{array}}{result(m) \doteq true}, \quad \delta_m^{\geq} = \frac{\begin{array}{c} \textit{is-higher} \doteq c_1 \\ \textit{than} \doteq c_2 \\ \text{“}c_1 < c_2\text{”} \end{array}}{result(m) \doteq false}$$

Lower-than? [*is-lower* $\doteq$ *number* *than* $\doteq$ *number*] $\longrightarrow$ *boolean*

The **lower-than?** method yields *true* if *is-lower* is lower than *than*. Otherwise yields *false*.

Inference rules $\delta_m^{<}$ performing the evaluation of a **lower-than?** method m are the following:

$$\delta_m^{<} = \frac{\begin{array}{c} \textit{is-lower} \doteq c_1 \\ \textit{than} \doteq c_2 \\ \text{“}c_1 < c_2\text{”} \end{array}}{result(m) \doteq true}, \quad \delta_m^{<} = \frac{\begin{array}{c} \textit{is-lower} \doteq c_1 \\ \textit{than} \doteq c_2 \\ \text{“}c_1 \geq c_2\text{”} \end{array}}{result(m) \doteq false}$$

Lower-equal-than? [*is-lower* $\doteq$ *number* *than* $\doteq$ *number*] $\longrightarrow$ *boolean*

The **lower-equal-than?** method yields *true* if *is-lower* is lower or equal than *than*. Otherwise yields *false*.

Inference rules $\delta_m^{\leq}$ performing the evaluation of a **lower-equal-than?** method m are the following:

$$\delta_m^{\leq} = \frac{\begin{array}{c} \textit{is-lower} \doteq c_1 \\ \textit{than} \doteq c_2 \\ \text{“}c_1 \leq c_2\text{”} \end{array}}{result(m) \doteq true}, \quad \delta_m^{\leq} = \frac{\begin{array}{c} \textit{is-lower} \doteq c_1 \\ \textit{than} \doteq c_2 \\ \text{“}c_1 > c_2\text{”} \end{array}}{result(m) \doteq false}$$

D.5 Methods on sets

All the methods on sets require that the values of their required features be sets of elements. If one of them is a singleton is considered as a set with one element.

Empty-set? [*set* $\doteq$ $\{any\}$] $\longrightarrow$ *boolean*

The **empty-set?** method yields *true* if *set* is *empty-set* and *false* otherwise.

Inference rules $\delta_m^{emp?}$ performing the evaluation of an **empty-set?** method m are the following:

$$\delta_m^{emp?} = \frac{\textit{set} \doteq \textit{empty-set}}{result(m) \doteq true}, \quad \delta_m^{emp?} = \frac{\textit{set} \doteq \{c_1 \cdots c_n\}}{result(m) \doteq false}$$

Cardinal $[set \doteq \{any\}] \longrightarrow number$

The **cardinal** method yields the number of elements in *set*.

The inference rule δ_m^{card} performing the evaluation of a **cardinal** method m is the following:

$$\delta_m^{card} = \frac{set \doteq \{c_1 \cdots c_n\}}{result(m) \doteq n}$$

Union $[set1 \doteq \{any\} \quad set2 \doteq \{any\}] \longrightarrow \{any\}$

The **union** method yields the union of sets *set1* and *set2*.

The inference rule δ_m^{union} performing the evaluation of a **union** method m is the following:

$$\delta_m^{union} = \frac{\begin{array}{c} set1 \doteq \{c_1 \cdots c_n\} \\ set2 \doteq \{c'_1 \cdots c'_m\} \\ \text{“}S = \{c_1 \cdots c_n\} \cup \{c'_1 \cdots c'_m\}\text{”} \end{array}}{result(m) \doteq S}$$

Intersection $[set1 \doteq \{any\} \quad set2 \doteq \{any\}] \longrightarrow \{any\}$

The **intersection** method yields the intersection of sets *set1* and *set2*.

The inference rule δ_m^{inter} performing the evaluation of an **intersection** method m is the following:

$$\delta_m^{inter} = \frac{\begin{array}{c} set1 \doteq \{c_1 \cdots c_n\} \\ set2 \doteq \{c'_1 \cdots c'_m\} \\ \text{“}S = \{c_1 \cdots c_n\} \cap \{c'_1 \cdots c'_m\}\text{”} \end{array}}{result(m) \doteq S}$$

Difference $[set1 \doteq \{any\} \quad set2 \doteq \{any\}] \longrightarrow \{any\}$

The **difference** method yields the set difference of *set1* and *set2*.

The inference rule δ_m^{diff} performing the evaluation of a **difference** method m is the following:

$$\delta_m^{diff} = \frac{\begin{array}{c} set1 \doteq \{c_1 \cdots c_n\} \\ set2 \doteq \{c'_1 \cdots c'_m\} \\ \text{“}S = \{c_1 \cdots c_n\} \setminus \{c'_1 \cdots c'_m\}\text{”} \end{array}}{result(m) \doteq S}$$

Member $[set \doteq \{any\} \quad item \doteq any \quad test \doteq symbol] \longrightarrow boolean$

The **member** method yields *true* if *item* is found in *set* using the comparison criterion *test*. Otherwise yields *false*. Three comparison criteria can be used: *identity*, *subsumption*, and *equivalence* corresponding to the comparison methods described before (see Section D.2).

Inference rules δ_m^{member} performing the evaluation of a **member** method m with *identity* comparison criterion are the following:

$$\delta_m^{member} = \frac{\begin{array}{c} set \doteq \{c_1 \cdots c_n\} \\ element \doteq c_i \\ test \doteq \text{'}identity \\ \text{“}c_i \in \{c_1 \cdots c_n\}\text{”} \end{array}}{result(m) \doteq true} \quad \delta_m^{member} = \frac{\begin{array}{c} set \doteq \{c_1 \cdots c_n\} \\ element \doteq c_i \\ test \doteq \text{'}identity \\ \text{“}c_i \notin \{c_1 \cdots c_n\}\text{”} \end{array}}{result(m) \doteq true}$$

Inference rules δ_m^{member} performing the evaluation of a **member** method m with *subsumption* comparison criterion are the following:

$$\delta_m^{member} = \frac{\begin{array}{c} set \doteq S \\ element \doteq c_i \\ test \doteq \text{'}subsumption \\ \text{“}\exists c_j \in S : c_i \sqsubseteq c_j\text{”} \end{array}}{result(m) \doteq true}, \qquad \delta_m^{member} = \frac{\begin{array}{c} set \doteq S \\ element \doteq c_i \\ test \doteq \text{'}subsumption \\ \text{“}\nexists c_j \in S : c_i \sqsubseteq c_j\text{”} \end{array}}{result(m) \doteq true}$$

Inference rules δ_m^{member} performing the evaluation of a **member** method m with *equivalence* comparison criterion are the following:

$$\delta_m^{member} = \frac{\begin{array}{c} set \doteq S \\ element \doteq c_i \\ test \doteq \text{'}equivalence \\ \text{“}\exists c_j \in S : c_j \sqsubseteq c_i \sqsubseteq c_j\text{”} \end{array}}{result(m) \doteq true}, \delta_m^{member} = \frac{\begin{array}{c} set \doteq S \\ element \doteq c_i \\ test \doteq \text{'}equivalence \\ \text{“}\nexists c_j \in S : c_j \sqsubseteq c_i \sqsubseteq c_j\text{”} \end{array}}{result(m) \doteq true}$$

D.6 Logic methods

All the logic methods require that the values of their required features be boolean. A non-boolean value in a feature produces an error.

Not $[item \doteq boolean] \longrightarrow boolean$

The **not** method yields the negation of *item*.

Inference rules δ_m^{not} performing the evaluation of a **not** method m are the following:

$$\delta_m^{not} = \frac{item \doteq false}{result(m) \doteq true}, \qquad \delta_m^{not} = \frac{item \doteq true}{result(m) \doteq false}$$

Conjunction $[item1 \doteq boolean \quad item2 \doteq boolean] \longrightarrow boolean$

The **conjunction** method yields *true* if *item1* and *item2* are both *true* and *false* otherwise.

The inference rule δ_m^{conj} performing the evaluation of a **conjunction** method m is the following:

$$\delta_m^{conj} = \frac{\begin{array}{c} item1 \doteq b_1 \\ item2 \doteq b_2 \\ \text{“}b = b_1 \wedge b_2\text{”} \end{array}}{result(m) \doteq b}$$

Disjunction $[item1 \doteq boolean \quad item2 \doteq boolean] \longrightarrow boolean$

The **disjunction** method yields *true* if either *item1* or *item2* is *true* and *false* if both are *false*.

The inference rule δ_m^{disj} performing the evaluation of a **disjunction** method m is the following:

$$\delta_m^{disj} = \frac{\begin{array}{c} item1 \doteq false \\ item2 \doteq false \\ \text{“}b = b_1 \vee b_2\text{”} \end{array}}{result(m) \doteq b}$$

D.7 Retrieval methods

Retrieval methods provide a powerful mechanism for accessing to the episodic memory contents. There are three built-in methods for retrieval defined in Noos: **retrieve-by-pattern**, **retrieve-by-task**, and **retrieve-by-feature-value**.

Retrieve-by-pattern $[pattern \doteq any] \longrightarrow \{any\}$

The **retrieve-by-pattern** method yields the collection of terms, from the episodic memory, subsuming the *pattern*.

The inference rule δ_m^{rbp} performing the evaluation of a **retrieve-by-pattern** method m on the episodic memory U is the following:

$$\delta_m^{rbp} = \frac{\begin{array}{c} pattern \doteq c \\ S = \{c_i \in U | c \sqsubseteq c_i\} \end{array}}{result(m) \doteq S}$$

Retrieve-by-task $[task\text{-}name \doteq symbol] \longrightarrow \{any\}$

The **retrieve-by-task** method yields the collection of terms, from the episodic memory, that have solved the task *task-name*.

The inference rule δ_m^{rbt} performing the evaluation of a **retrieve-by-task** method m on the episodic memory U is the following:

$$\delta_m^{rbt} = \frac{\begin{array}{c} \textit{task-name} \doteq \text{'}f \\ S = \{c_i \in U | c_i.f \text{ is defined}\} \end{array}}{result(m) \doteq S}$$

Retrieve-by-feature-value [*task-name* $\doteq$ *symbol* *value* $\doteq$ *any*] $\longrightarrow \{any\}$

The **retrieve-by-feature-value** method yields the collection of terms, from the episodic memory, that have *value* in *task-name*.

The inference rule δ_m^{rbfv} performing the evaluation of a **retrieve-by-feature-value** method m on the episodic memory U is the following:

$$\delta_m^{rbfv} = \frac{\begin{array}{c} \textit{task-name} \doteq \text{'}f \\ value \doteq c \\ S = \{c_i \in U | c_i.f \doteq c\} \end{array}}{result(m) \doteq S}$$

D.8 Preferences

We have described preferences in Section 3.4. Now we will briefly describe preference methods indicating their required features and their corresponding inference rules using the definitions of preference operations given in Section 5.10. The reader can consult Section 3.4 for examples of the use of preferences in Noos.

We have explained that a preference method takes a set of source elements and builds a preference taking into account an ordering criterion. In fact, built-in preference methods are more powerful: when the set of source elements is already a partially ordered set, the new preference is added using the hierarchical union operator. Since all preference methods performs a hierarchical union, for the sake of clarity we will first describe a preference method without the hierarchical union and then, we will present its inference rule incorporating the hierarchical union.

Increasing-order-preference [*poset* $\doteq$ *poset* *feature* $\doteq$ *symbol*] $\longrightarrow$ *poset*

The *increasing-order-preference* method takes the set of elements in *poset*, the feature name *feature* of a feature with numeric value, and yields a new preference where the most preferred elements are those with a higher value in the specified feature. If any value for *feature* is not numeric an error is produced.

The inference rule δ_m^{incp} performing the evaluation of a **increasing-order-preference** method m is the following:

$$\delta_m^{incp} = \frac{\begin{array}{c} poset \doteq \langle S, \prec \rangle \\ feature \doteq \text{'}f\text{'} \\ \text{“} \prec' = \{(c_i, c_j) | c_i, c_j \in S, c_i.f > c_j.f\} \\ \text{“}p' = \langle S, \prec \rangle \bullet \langle S, \prec' \rangle\text{”} \end{array}}{result(m) \doteq p'}$$

Decreasing-order-preference $[poset \doteq poset \quad feature \doteq symbol] \longrightarrow poset$

The *decreasing-order-preference* method takes the set of elements in *poset*, the feature name *feature* of a feature with numeric value, and yields a new preference where the most preferred elements are those with a lesser value in the specified feature. If any value for *feature* is not numeric an error is produced.

The inference rule δ_m^{decp} performing the evaluation of a **decreasing-order-preference** method m is the following:

$$\delta_m^{decp} = \frac{\begin{array}{c} poset \doteq \langle S, \prec \rangle \\ feature \doteq \text{'}f \\ \text{“} \prec' = \{(c_i, c_j) | c_i, c_j \in S, c_i.f < c_j.f\}\text{”} \\ \text{“}p' = \langle S, \prec \rangle \bullet \langle S, \prec' \rangle\text{”} \end{array}}{result(m) \doteq p'}$$

Higher-threshold-preference
$[poset \doteq poset \quad feature \doteq symbol \quad threshold \doteq number] \longrightarrow poset$

The *higher-threshold-preference* method takes the set of elements in *poset*, the feature name *feature* of a feature with numeric value, a *threshold*, and yields a new preference where the elements with a higher value than *value* in *feature* are preferred to elements with lower or equal value than *value*. If any value for *feature* is not numeric an error is produced.

The inference rule δ_m^{htp} performing the evaluation of a **higher-threshold-preference** method m is the following:

$$\delta_m^{htp} = \frac{\begin{array}{c} poset \doteq \langle S, \prec \rangle \\ feature \doteq \text{'}f \\ threshold \doteq c \\ \text{“} \prec' = \{(c_i, c_j) | c_i, c_j \in S, c_i.f > c, c_j.f \leq c\}\text{”} \\ \text{“}p' = \langle S, \prec \rangle \bullet \langle S, \prec' \rangle\text{”} \end{array}}{result(m) \doteq p'}$$

Lower-threshold-preference
$[poset \doteq poset \quad feature \doteq symbol \quad threshold \doteq number] \longrightarrow poset$

The *lower-threshold-preference* method takes the set of elements in *poset*, the feature name *feature* of a feature with numeric value, a *threshold*, and yields a new preference where the elements with a lower value than *value* in *feature* are preferred to elements with higher or equal value than *value*. If any value for *feature* is not numeric an error is produced.

The inference rule δ_m^{ltp} performing the evaluation of a **lower-threshold-preference** method m is the following:

$$\delta_m^{ltp} = \frac{\begin{array}{c} poset \doteq \langle S, \prec \rangle \\ feature \doteq {}'f \\ threshold \doteq c \\ \text{``} \prec' = \{(c_i, c_j) | c_i, c_j \in S, c_i.f < c, c_j.f \geq c\}\text{''} \\ \text{``}p' = \langle S, \prec \rangle \bullet \langle S, \prec' \rangle\text{''} \end{array}}{result(m) \doteq p'}$$

Subsumption-preference $[poset \doteq poset \quad pattern \doteq any] \longrightarrow poset$

The *subsumption-preference* method takes the set of elements in *poset*, the term *pattern*, and yields a new preference where the preferred elements are those *pattern* subsumes.

The inference rule δ_m^{subp} performing the evaluation of a **subsumption-preference** method m is the following:

$$\delta_m^{subp} = \frac{\begin{array}{c} poset \doteq \langle S, \prec \rangle \\ pattern \doteq c \\ \text{``} \prec' = \{(c_i, c_j) | c_i, c_j \in S, c_i.f \sqsubseteq c, c_j.f \not\sqsubseteq c\}\text{''} \\ \text{``}p' = \langle S, \prec \rangle \bullet \langle S, \prec' \rangle\text{''} \end{array}}{result(m) \doteq p'}$$

Equal-value-preference

$[poset \doteq poset \quad feature \doteq symbol \quad value \doteq number] \longrightarrow poset$

The *equal-value-preference* method takes the set of elements in *poset*, the feature name *feature* of a feature with numeric value, a *value*, and yields a new preference where the elements with *value* in *feature* are preferred to others.

The inference rule δ_m^{evp} performing the evaluation of a **equal-value-preference** method m is the following:

$$\delta_m^{evp} = \frac{\begin{array}{c} poset \doteq \langle S, \prec \rangle \\ feature \doteq {}'f \\ value \doteq c \\ \text{``} \prec' = \{(c_i, c_j) | c_i, c_j \in S, c_i.f = c, c_j.f \neq c\}\text{''} \\ \text{``}p' = \langle S, \prec \rangle \bullet \langle S, \prec' \rangle\text{''} \end{array}}{result(m) \doteq p'}$$

User-preference $[poset \doteq poset] \longrightarrow poset$

The *user-preference* method takes the set of elements in *poset* and yields a new preference according to the answer provided by the user using a window interface.

Inversion $[poset \doteq poset] \longrightarrow poset$

The *inversion* method yields a new preference, inversion of preference *poset*.

The inference rule δ_m^{inver} performing the evaluation of a **inversion** method m is the following:

$$\delta_m^{inver} = \frac{\begin{array}{c} poset \doteq p \\ \text{“}p' = p^{-1}\text{”} \end{array}}{result(m) \doteq p'}$$

T-intersection $[poset1 \doteq poset \quad poset2 \doteq poset] \longrightarrow poset$

The *t-intersection* method yields a new preference by restricting preferences *poset1* and *poset2* to the elements of their intersection and, then, performing a transitive union on the resulting preferences (see Definition 5.25 in Section 5.10).

The inference rule $\delta_m^{\overline{\cap}}$ performing the evaluation of a *t-intersection* method m is the following:

$$\delta_m^{\overline{\cap}} = \frac{\begin{array}{c} poset1 \doteq p_1 \\ poset2 \doteq p_2 \\ \text{“}p = p_1 \overline{\cap} p_2\text{”} \end{array}}{result(m) \doteq p}$$

C-intersection $[poset1 \doteq poset \quad poset2 \doteq poset] \longrightarrow poset$

The *c-intersection* method yields a new preference by restricting preferences *poset1* and *poset2* to the elements of their intersection and, then, performing an intersection on the resulting preferences (see Definition 5.26 in Section 5.10).

The inference rule $\delta_m^{\underline{\cap}}$ performing the evaluation of a *c-intersection* method m is the following:

$$\delta_m^{\underline{\cap}} = \frac{\begin{array}{c} poset1 \doteq p_1 \\ poset2 \doteq p_2 \\ \text{“}p = p_1 \underline{\cap} p_2\text{”} \end{array}}{result(m) \doteq p}$$

T-union $[poset1 \doteq poset \quad poset2 \doteq poset] \longrightarrow poset$

The *t-union* method yields a new preference by extending preferences *poset1* and *poset2* to the elements of their union and, then, performing a transitive union on the resulting preferences (see Definition 5.27 in Section 5.10).

The inference rule $\delta_m^{\overline{\uplus}}$ performing the evaluation of a *t-union* method m is the following:

$$\delta_m^{\overline{\uplus}} = \frac{\begin{array}{c} poset1 \doteq p_1 \\ poset2 \doteq p_2 \\ \text{“}p = p_1 \overline{\uplus} p_2\text{”} \end{array}}{result(m) \doteq p}$$

C-union $[poset1 \doteq poset \quad poset2 \doteq poset] \longrightarrow poset$

The *c-union* method yields a new preference by extending preferences *poset1* and *poset2* to the elements of their union and, then, performing an intersection on the resulting preferences (see Definition 5.28 in Section 5.10).

The inference rule $\delta_m^{\underline{\uplus}}$ performing the evaluation of a *c-union* method m is the following:

$$\delta_m^{\underline{\uplus}} = \frac{\begin{array}{c} poset1 \doteq p_1 \\ poset2 \doteq p_2 \\ \text{“}p = p_1 \underline{\uplus} p_2\text{”} \end{array}}{result(m) \doteq p}$$

H-union $[higher - poset \doteq poset \quad lower - poset \doteq poset] \longrightarrow poset$

The *h-union* method yields a new preference by extending preferences *higher-poset* and *lower-poset* to the elements of their union and, then, performing a hierarchical union on the resulting preferences (see Definition 5.30 in Section 5.10).

The inference rule $\delta_m^{\bullet}$ performing the evaluation of a *h-union* method m is the following:

$$\delta_m^{\bullet} = \frac{\begin{array}{c} higher - poset \doteq p_1 \\ poset2 \doteq p_2 \\ \text{“}p = p_1 \bullet p_2\text{”} \end{array}}{result(m) \doteq p}$$

D.9 Methods for interaction

These kind of methods allow user interaction within the inference process. These methods are commonly used for dynamically choosing among alternatives and for introducing new values for features.

We will describe the two built-in methods provided by Noos without introducing inference rules.

Ask-option-to-user $[question \doteq string \quad options \doteq set] \longrightarrow set$

The **ask-option-to-user** method shows *question* and *options* to the user and yields the subset of elements of *options* chosed by the user.

Ask-value-to-user $[question \doteq string] \longrightarrow \{any\}$

The **ask-value-to-user** method shows *question* and request a value to the user. The value can be any feature term already defined. Otherwise an error is produced.

D.10 Query methods

We have described query-methods in Section 3.3.6. Now we will briefly describe query-methods indicating their required features. Inference rules and DDL programs formalizing inference of query-methods are given in Section 5.13. The reader can consult Section 3.3.6 for examples of the use of query-methods in Noos.

Infer-value $[\mathit{feature} \doteq symbol \quad domain \doteq \{any\}] \longrightarrow \{any\}$

The **infer-value** method takes a feature name f in *feature*; a unit or a set of units S in *domain*; performs tasks $f(s_i)$ for all units in S; and yields as result the union of the values inferred in those tasks.

Known-value $[\mathit{feature} \doteq symbol \quad domain \doteq \{any\}] \longrightarrow \{any\}$

The **known-value** method takes a feature name f in *feature*; a unit or a set of units S in *domain*; and determines if a solution to a task f for the set of units in S have already been computed, yielding a boolean accordingly.

Exists-value $[\mathit{feature} \doteq symbol \quad domain \doteq \{any\}] \longrightarrow \{any\}$

The **exists-value** method takes a feature name f in *feature*; a unit or a set of units S in *domain*; and determines if any solution to a task f for the set of units in S exists, yielding a boolean accordingly.

All-values $[\mathit{feature} \doteq symbol \quad domain \doteq \{any\}] \longrightarrow \{any\}$

The **all-values** method takes a feature name f in *feature*; a unit or a set of units S in *domain*; determines the set of all inferrable values of tasks $f(s_i)$ for all units in S; and yields as result the union of all those values.

D.11 Eval methods

Eval-methods have been described in Section 3.3.6. Now we will briefly describe eval-methods indicating their required features. Inference rules and DDL programs formalizing inference of eval-methods are given in Section 5.13.

Noos-eval $[methods \doteq \{method\}] \longrightarrow \{any\}$

The **noos-eval** method takes a method or a set of methods M in *methods*; performs their evaluation; and yields as result the union of the values inferred in the evaluation of those methods.

Known-eval $[methods \doteq \{method\}] \longrightarrow \{any\}$

The **known-eval** method takes a method or a set of methods M in *methods* and determines if a solution for methods in M have already been computed, yielding a boolean accordingly.

Exists-eval $[methods \doteq \{method\}] \longrightarrow \{any\}$

The **exists-eval** method takes a method or a set of methods M in *methods* and determines if exists a solution for methods in M, yielding a boolean accordingly.

All-eval $[methods \doteq \{method\}] \longrightarrow \{any\}$

The **all-eval** method takes a method or a set of methods M in *methods*; determines the set of all inferrable values in their evaluations; and yields as result the union of all those values.

D.12 Reflective operations

We have described reflective operations in Section 3.3.5. Reflective operations are also reified as methods. Now we will briefly describe these methods indicating their required features and no inference rules will be provided. The reader can consult Section 3.3.5 for examples of the use of reflective operations.

Meta $[object \doteq any] \longrightarrow any$

The **meta** method performs the subtask *object* and then, yields the meta-level of *object* as result.

Default $[object \doteq any] \longrightarrow any$

The **default** method performs the subtask *object* and then, yields the default metalevel of *object* as result.

Referent $[object \doteq any] \longrightarrow any$

The **referent** method performs the subtask *object* and then, yields the referent of *object* as result.

Constituent $[object \doteq any] \longrightarrow any$

The **constituent** method performs the subtask *object* and then, yields the constituent of *object* as result.

Task $[task\text{-}name \doteq symbol \quad domain \doteq any] \longrightarrow task$

The **task** method performs subtasks *task-name* and *domain* obtaining respectively a task name `F` and a term `D`. Then, yields the task term that reifies the inference `F(D)` as result.

Current-task $[method \doteq method] \longrightarrow task$

The **current-task** method performs the subtask *method*, obtaining a method `M`, and then yields as result the task term that reifies the task `M` is solving.