Appendix C

The Noos Syntax

This Appendix describes the syntax of Noos using BNF notation. We write predefined terminal symbols, that are part of the Noos language, in `typewriter` font. We write user-defined identifiers in *italic* font. We write non-terminal symbols in normal type face. Symbols ::=, [,] |, *, + are part of the BNF formalism as follows:

L ::= R	defines the syntax of L as R
[X]	defines an optional item
X | Y	defines two alternative options X and Y
X+	defines one or more occurrences of X
X*	defines zero or more occurrences of X

top-level-expression	::=	description
	|	query-expression
description	::=	single-description
	|	named-description
	|	anonymous-description
	|	set-description
	|	named-metalevel-description
	|	anonymous-metalevel-description
	|	named-default-description
	|	anonymous-default-description
single-description	::=	`(define` *name* [`:ephemeral`] feature-description*`)`
named-description	::=	`(define (`*constituent [*`:id`*] name*`)` [`:ephemeral`] feature-description*`)`

```
anonymous-description            ::= (define (constituent) [:ephemeral]
                                        feature-description*)

set-description                  ::= (define (set name)
                                        [name | anonymous-description]*)

named-metalevel-description      ::= (define (metalevel name) [:ephemeral]
                                        feature-description*)

anonymous-metalevel-description  ::= (define (metalevel (meta+ of name))
                                        [:ephemeral]
                                        feature-description*)

named-default-description        ::= (define (default name)
                                        [name | anonymous-description]*)

anonymous-default-description    ::= (define (default (default meta* of name))
                                        [name | anonymous-description]*)

feature-description              ::= (feature-name v-expression*)
                                  |  ((feature-name v-expression+))

v-expression                     ::= name
                                  |  'symbol | string | number
                                  |  anonymous-description
                                  |  path-reference
                                  |  eval-expression
                                  |  compact-method
                                  |  reification
                                  |  metalevel-operation

query-expression                 ::= path-reference
                                  |  eval-expression
                                  |  metalevel-operation

path-reference                   ::= (>> v-expression* [of v-expression])
                                  |  (!>> v-expression* [of v-expression])
                                  |  (?>> v-expression* [of v-expression])
                                  |  (*>> v-expression* [of v-expression])

eval-expression                  ::= (noos-eval [v-expression])
                                  |  (known-eval [v-expression])
                                  |  (exists-eval [v-expression])
                                  |  (all-eval [v-expression])
```

compact-method ::= (*name* v-expression*)

reification ::= (`reify` v-expression)

metalevel-operation ::= (`meta` [v-expression])
| (`default` [v-expression])
| (`referent` [v-expression])
| (`current-task` [v-expression])
| (`task` v-expression [`of` v-expression])

C.1 Compact syntax for closed methods

Compact methods allows the defintion of closed methods by position instead of by name. Since definition by position assumes that required features of a built-in method are specified in a particular order, we introduce here, for each built-in method, the name of the method in compact syntax (using typewriter font) and the assumed order of parameters (inside angles). Parameters can be any v-expression such as a feature value.

When we use the compact syntax we have to specify a value for all the required features and only the required features can be specified. Otherwise an error is produced.

Compact syntax for built-in methods such as `conjunction`, `disjunction`, `union`, `intersection`, `add`, `mult`, `max`, and `min` allows the definition of n arguments. In fact, these definitions are translated as a composition of compact expressions. For instance, the following compact expression:

```
(max 1 3 5 4 2)
```

is translated to the following compact expression:

```
(max 1 (max 3 (max 5 (max 4 2))))
```

Compact syntax for Noos built-in methods is the following:

conditional ::= (`if` <condition> <result> [<otherwise>])

not ::= (`not` <item>)

conjunction ::= (`and` <item1> <item2> $\cdots$ <itemn>)

disjunction ::= (`or` <item1> <item2> $\cdots$ <itemn>)

difference ::= (`difference` <set1> <set2>)

member ::= (`member` <element> <set> [<test>])

union ::= (`union` <set1> <set2> $\cdots$ <setn>)

intersection	::=	(**intersection** <set1> <set2> ··· <setn>)
empty-set?	::=	(**empty-set?** <set>)
cardinal	::=	(**cardinal** <set>)
add	::=	(+ <item1> <item2> ··· <itemn>)
subtract	::=	(- <amount> <minus>)
mult	::=	(* <item1> <item2> ··· <itemn>)
div	::=	(/ <item1> <item2>)
max	::=	(**max** <item1> <item2> ··· <itemn>)
min	::=	(**min** <item1> <item2> ··· <itemn>)
higher-than?	::=	(> <is-higher> <than>)
higher-equal-than?	::=	(>= <is-higher> <than>)
lower-than?	::=	(< <is-lower> <than>)
lower-equal-than?	::=	(<= <is-lower> <than>)
identity?	::=	(**identity?** <item1> <item2>)
subsumption	::=	(**subsumes?** <pattern> <source>)
equivalence	::=	(**equivalent?** <item1> <item2>)