RoboTool Developer's Manual

Alvaro Miyazawa Pedro Ribeiro
Kangfeng Ye Ana Cavalcanti Wei Li
Jim Woodcock Jon Timmis

WWW.CS.YORK.AC.UK/ROBOSTAR

Contents

III Appendices

List of rules

1 Untimed semantics

1. Introduction

This report complements the *RoboChart Reference Manual* [7], the *RoboSim Reference Manual* citeRoboSim and the *RoboSim Physical Modelling Reference Manual* [5] with RoboTool implementation specific details. For instance, while the expression language of RoboChart is the expression language of the Z notation [1, 4], the tool uses a slightly different metamodel. Such differences are documented here.

RoboChart

2. Language Structure

In this chapter, we describe the implementation specific differences between the metamodel of RoboChart in RoboTool and the proposed metamodel of RoboChart. The complete ecore metamodel used in the implementation is shown in Appendix **??**.

2.0.1 Type Declaration

While in RoboChart [7], the domain of a variable is an expression, in RoboTool, this is implemented as a more restrictive system where only types can be used as the domain of variables. This requires a class of type constructs separate from the class of expressions as in Z. The metamodel of type constructs is described in this section.

The metamodel for Types is given in Figure 2.1. As indicated, they include references to type declarations (TypeRef), sets (SetType), and cartesian products (ProductType). A type reference refer to type declarations, which are PrimitiveTypes, given just by their names, Enumerations, records (DataType), named definitions (NamedDefinition). Table 2.1 gives the concrete syntax for the various type constructors.

Additional types, such as functions, relations and sequences, which are defined in the mathematical toolkit of Z [1], hard coded in the tool with the concrete syntax shown in Table 2.1.

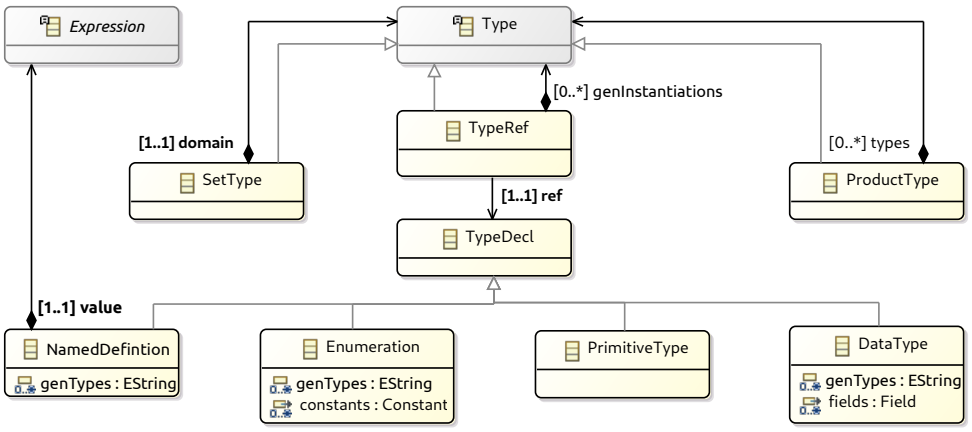

Figure 2.1: Metamodel of RoboChart types

Element	Concrete Syntax	Meaning
Type Reference	`N`	`N` is the name of a declared type.
Set Type	`Set(T)`	Type of sets of elements of `T`.
Sequence Type	`Seq(T)`	Type of sequences of elements of `T1` to `T2`.
Product Type	`T1 * T2`	Type of pairs whose first element has type `T1` and second element has type `T2`.
Function Type	`T1 -> T2`	Type of functions from `T1` to `T2`.
Relation Type	`T1 <-> T2`	Type of relations between `T1` and `T2`.
Vector Type	`vector(T1,N)`	Type of `N`-vectors of `T1` values.
Matrix Type	`matrix(T1,M,N)`	Type of `M` by `N` matrices of `T1` values.

Table 2.1 : **Concrete syntax of types.**

2.0.2 Expression

The expressions implemented in the tool are, in general, those of Z with a slightly different syntax. A number of additional expressions are defined as syntactic sugar to provide a more roboticist friendly syntax; these include arrays, vectors and matrices.

Expressions include logical expressions of a first-order predicate calculus, and usual arithmetic and relational expressions. We also have a LetExpression, an IFExpression, and expressions to deal with tuples, enumerated types, function calls, and so on. The syntax boxes 2.0.1 and 2.0.2 give the concrete syntax for expressions. We omit the simple metamodel diagrams here.

Syntax 2.0.1 — Expressions.

```
Expr  ::=  ('0'..'9')+                                  – integer
       |   ('0'..'9')+.('0'..'9')+                      – float
       |   '"'[^'"']+'"'                                – string
       |   'true' | 'false'                             – boolean
       |   N                                            – reference
       |   '<'(Expr (',' Expr)*)?'>'                     – sequence
       |   '{'(Expr (',' Expr)*)?'}'                     – set
       |   '{' N ':' Type '|' Expr '@' Expr '}'          – set comprehension
       |   '['Expr ',' Expr ']'                         – closed interval
       |   '(' Expr ',' Expr ')'                        – open interval
       |   N '::' N                                     – enumeration constant
       |   '(|' (Expr (',' Expr)*)? '|)'                – tuple
       |   Expr '['  Expr ']'                           – array access
       |   Expr '(' (Expr (',' Expr)*)? ')'             – function application
       |   Expr '.' N                                   – field access
       |   '-' Expr                                     – negation
       |   Expr '+' Expr                                – sum
       |   Expr '-' Expr                                – subtraction
       |   Expr '*' Expr                                – multiplication
       |   Expr '/' Expr                                – division
       |   Expr '%' Expr                                – remainder
       |   Expr '==' Expr                               – equality
       |   Expr '=' Expr!                               – difference
       |   Expr '>' Expr                                – greater
       |   Expr '>=' Expr                               – greater or equal
       |   Expr '<' Expr                                – less
       |   Expr '<=' Expr                               – less or equal
       |   'not' Expr                                   – not
       |   Expr '/\' Expr                               – and
       |   Expr '\/' Expr                               – or
       |   Expr '=> Expr                                – implies
       |   Expr 'iff' Expr                              – if and only if
       |   ...                                          – continues on Syntax 2.0.2
```

Syntax 2.0.2 — Expressions (cont.).

```
Expr  ::=  ...
      |    'forall' N ':' Type '|' Expr '@' Expr        – universal quantification
      |    'exists' N ':' Type '|' Expr '@' Expr        – existential quantification
      |    'exists1' N ':' Type '|' Expr '@' Expr       – uniqueness quantification
      |    Expr '^' Expr                                – concatenation
      |    'if' Expr 'then' Expr 'else' Expr 'end'      – conditional
      |    'let' N '==' Expr '@' Expr                   – local definition
      |    'the' N ':' Type '|' Expr '@' Expr           – definite description
      |    'lambda' N ':' Type '|' Expr '@' Expr        – lambda expression
      |    'since' '(' N ')'                            – clock expression
      |    'sinceEntry' '(' N ')'                       – state clock expression
```

Table 2.2 summarises primitive expression in RoboChart. Table 2.3 describes the arithmetic expressions, Table 2.4 list the expressions used to compare values, Table 2.5 covers the logical expressions, and Table 2.6 explains the remaining expressions.

Expression	Concrete Syntax	Comment
Integer	`(0..9)+`	
Float	`(0..9)+.(0..9)+`	
String	`"..."`	Quoted values
Boolean	`true | false`	
Reference	`N`	`N` is the name of a variable or constant.
Sequence	`<e1,e2,...>`	Sequence with values `ei`.
Set	`{e1,e2,...}`	Set with values `ei`.
Set Comprehension	`{x:T | P @ e}`	Set containing values `e`, calculated from elements of type `T`, for which the predicate `P` holds.
Interval	`[e1,e2]` or `(e3,e4)`	Closed interval between `e1` and `e2`, and open interval between `e3` and `e4`, or a combination of both.
Enumeration	`E::c`	Constant `c` of enumeration `E`.
Tuple	`(|e1,e2,...|)`	Tuple containing elements `ei`.
Array	`e[i]`	The i-th element of array `e`.
Function application	`f(e1,e2,...)`	Apply function `f` to parameters `ei`.
Selection	`e.n`	The `n` field of record `e`.
Vector	`[|v1,...,vn|]`	An n-vector.
Matrix	`[|v11,...,v1n;...;vm1,...,vmn|]`	An m-by-n matrix.
Inverse	`inverse(M)`	The inverse of a matrix.
Transpose	`transpose(M)`	The transpose of a matrix.

Table 2.2 : **Primitive Expressions.**

Expression	Concrete Syntax	Comment
Negation	`-e`	Arithmetical negation of expression `e`.
Sum	`e1 + e2`	Sum of `e1` and `e2`.
Subtraction	`e1 - e2`	Subtraction of `e1` and `e2`.
Multiplication	`e1 * e2`	Multiplication of `e1` by `e2`.
Division	`e1 / e2`	Division of `e1` by `e2`.
Modulo	`e1 % e2`	Remainder of dividing `e1` by `e2`.

Table 2.3 : **Arithmetic Expressions.**

Expression	Concrete Syntax	Comment
Equality	`e1 == e2`	True if both expressions are equal.
Different	`e1 != e2`	True if both expressions are different.
Greater than	`e1 > e2`	True if `e1` is greater than `e2`.
Greater than or equal to	`e1 >= e2`	True if `e1` is greater than or equal to `e2`.
Less than	`e1 < e2`	True if `e1` is less than `e2`.
Less than or equal to	`e1 <= e2`	True if `e1` is less than or equal to `e2`.

Table 2.4 : **Comparison Expressions.**

Expression	Concrete Syntax	Comment
Logical not	`not e`	True if and only if `e` is false.
Logical and	`e1 /\ e2`	True if and only if `e1` and `e2` are true.
Logical or	`e1 \/ e2`	True if and only if at least one of the expressions is true.
Logical implies	`e1 => e2`	Equivalent to `not e1 \/ e2`.
Logical iff	`e1 iff e2`	Equivalent to `e1=>e2 /\ e2=>e1`
Universal quantification	`forall x: T | P @ Q`	True if and only if for all elements of `T`, if `P` is true, then `Q` is true.
Existential quantification	`exists x: T | P @ Q`	True if and only if there is an element of `T`, for which `P` is true and `Q` is true.
Uniqueness quantification	`exists1 x: T | P @ Q`	True if and only if there is a unique element of `T`, for which `P` and `Q` are true.

Table 2.5 : **Logical Expressions.**

Expression	Concrete Syntax	Comment
Concatenation	`e1^e2`	Concatenate sequences `e1` and `e2`.
Conditional	`if c then e else f end`	If condition `c` is true, `e1` else `e2`.
Local definition	`let n == e @ f`	Define locally `n` and use it to calculate `f`.
Definite description	`the x: T | P @ e`	The value `e` calculated based on the unique `x` for which `P` holds.
Lambda expression	`lambda x: T | P @ e`	The anonymous function that takes values of type `T` for which `P` holds, to values `e` calculated based on `x`.

Table 2.6 : **Other Expressions.**

3. Semantics

In this chapter, we specify the semantics of expressions as function of the RoboTool specific metamodel.

For the purpose of this semantics, the functions *vid*, *eventId*, *tid* and *id* calculate unique identifiers for their parameters, which are, respectively, variables, events, transitions and node containers (states and state machines). One possible implementation of such functions is to calculate the qualified name, and this is the implementation realised by RoboTool.

Finally, we assume the existence of a function that takes an expression and returns the set of variables used in that expression.

3.1 Detailed Semantics: Core Language

3.1.1 Expressions

Rule 1. Semantics of and expression

$$[\![s : And]\!]_{\mathcal{E}xpr} : CSPExpression = [\![s.left]\!]_{\mathcal{E}xpr^1} \wedge [\![s.right]\!]_{\mathcal{E}xpr^2}$$

Rule 2. Semantics of array expression

$$[\![s : ArrayExp]\!]_{\mathcal{E}xpr} : CSPExpression = [\![s.value]\!]_{\mathcal{E}xpr^3} (\{p : s.parameters \bullet [\![p]\!]_{\mathcal{E}xpr^4}\})$$

Rule 3. Semantics of boolean expression

$[\![s : \text{BooleanExp}]\!]_{\mathcal{E}xpr} : \text{CSPExpression} =$

if (s.value = TRUE) then *true* else *false*

Rule 4. Semantics of call expression

$[\![s : \text{CallExp}]\!]_{\mathcal{E}xpr} : \text{CSPExpression} =$

if (name = '*size*' ∧ (head s.args **has type** SetType)) then
  card($[\![\text{head s.args}]\!]_{\mathcal{E}xpr^5}$)
else if (name = '*size*' ∧ (head s.args **has type** SeqType)) then
  length($[\![\text{head s.args}]\!]_{\mathcal{E}xpr^6}$)
else
  name($\{a : s.\text{args} \bullet [\![a]\!]_{\mathcal{E}xpr^7}\}$)

where

name = s.function.name

Rule 5. Semantics of concatenation expression

$[\![s : \text{Cat}]\!]_{\mathcal{E}xpr} : \text{CSPExpression} =$

$[\![\text{s.left}]\!]_{\mathcal{E}xpr^8} \frown [\![\text{s.right}]\!]_{\mathcal{E}xpr^9}$

Rule 6. Semantics of not equal expression

$[\![s : \text{Different}]\!]_{\mathcal{E}xpr} : \text{CSPExpression} =$

$[\![\text{s.left}]\!]_{\mathcal{E}xpr^{10}} \neq [\![\text{s.right}]\!]_{\mathcal{E}xpr^{11}}$

Rule 7. Semantics of division

$[\![s : \text{Div}]\!]_{\mathcal{E}xpr} : \text{CSPExpression} =$

$[\![\text{s.left}]\!]_{\mathcal{E}xpr^{12}} / [\![\text{s.right}]\!]_{\mathcal{E}xpr^{13}}$

Rule 8. Semantics of equality

$[\![s : \text{Equals}]\!]_{\mathcal{E}xpr} : \text{CSPExpression} =$

$[\![\text{s.left}]\!]_{\mathcal{E}xpr^{14}} = [\![\text{s.right}]\!]_{\mathcal{E}xpr^{15}}$

Rule 9. Semantics of greater or equal expression

$[\![s : \text{GreaterOrEqual}]\!]_{\mathcal{E}xpr} : \text{CSPExpression} =$

$[\![\text{s.left}]\!]_{\mathcal{E}xpr^{16}} \geq [\![\text{s.right}]\!]_{\mathcal{E}xpr^{17}}$

Rule 10. Semantics of greater than

$[\![s : \text{GreaterThan}]\!]_{\mathcal{E}expr} : \text{CSPExpression} =$

$$[\![s.\text{left}]\!]_{\mathcal{E}expr^{18}} > [\![s.\text{right}]\!]_{\mathcal{E}expr^{19}}$$

Rule 11. Semantics of if and only if expression

$[\![s : \text{Iff}]\!]_{\mathcal{E}expr} : \text{CSPExpression} =$

$$[\![s.\text{left}]\!]_{\mathcal{E}expr^{20}} \Leftrightarrow [\![s.\text{right}]\!]_{\mathcal{E}expr^{21}}$$

Rule 12. Semantics of implication

$[\![s : \text{Implies}]\!]_{\mathcal{E}expr} : \text{CSPExpression} =$

$$[\![s.\text{left}]\!]_{\mathcal{E}expr^{22}} \Rightarrow [\![s.\text{right}]\!]_{\mathcal{E}expr^{23}}$$

Rule 13. Semantics of integer expression

$[\![s : \text{IntegerExp}]\!]_{\mathcal{E}expr} : \text{CSPExpression} =$

$$s.\text{value}$$

Rule 14. Semantics of less or equal expression

$[\![s : \text{LessOrEqual}]\!]_{\mathcal{E}expr} : \text{CSPExpression} =$

$$[\![s.\text{left}]\!]_{\mathcal{E}expr^{24}} \leq [\![s.\text{right}]\!]_{\mathcal{E}expr^{25}}$$

Rule 15. Semantics of less than

$[\![s : \text{LessThan}]\!]_{\mathcal{E}expr} : \text{CSPExpression} =$

$$[\![s.\text{left}]\!]_{\mathcal{E}expr^{26}} < [\![s.\text{right}]\!]_{\mathcal{E}expr^{27}}$$

Rule 16. Semantics of minus

$[\![s : \text{Minus}]\!]_{\mathcal{E}expr} : \text{CSPExpression} =$

$$[\![s.\text{left}]\!]_{\mathcal{E}expr^{28}} - [\![s.\text{right}]\!]_{\mathcal{E}expr^{29}}$$

Rule 17. Semantics of modulus

$[\![s : \text{Modulus}]\!]_{\mathcal{E}expr} : \text{CSPExpression} =$

$$[\![s.\text{left}]\!]_{\mathcal{E}expr^{30}} \text{ mod } [\![s.\text{right}]\!]_{\mathcal{E}expr^{31}}$$

Rule 18. Semantics of multiplication

$[\![s : Mult]\!]_{\mathcal{E}xpr} : CSPExpression =$

$[\![s.left]\!]_{\mathcal{E}xpr^{32}} \times [\![s.right]\!]_{\mathcal{E}xpr^{33}}$

Rule 19. Semantics of arithmetic negation

$[\![s : Neg]\!]_{\mathcal{E}xpr} : CSPExpression =$

$-[\![s.exp]\!]_{\mathcal{E}xpr^{34}}$

Rule 20. Semantics of logical negation

$[\![s : Not]\!]_{\mathcal{E}xpr} : CSPExpression =$

$\neg\ [\![s.exp]\!]_{\mathcal{E}xpr^{35}}$

Rule 21. Semantics of or expression

$[\![s : Or]\!]_{\mathcal{E}xpr} : CSPExpression =$

$[\![s.left]\!]_{\mathcal{E}xpr^{36}} \vee [\![s.right]\!]_{\mathcal{E}xpr^{37}}$

Rule 22. Semantics of parenthesised expression

$[\![s : ParExp]\!]_{\mathcal{E}xpr} : CSPExpression =$

$([\![s.exp]\!]_{\mathcal{E}xpr^{38}})$

Rule 23. Semantics of plus

$[\![s : Plus]\!]_{\mathcal{E}xpr} : CSPExpression =$

$[\![s.left]\!]_{\mathcal{E}xpr^{39}} + [\![s.right]\!]_{\mathcal{E}xpr^{40}}$

Rule 24. Semantics of range expression

$[\![s : RangeExp]\!]_{\mathcal{E}xpr} : CSPExpression =$

$\{x : \mathbb{N} \mid [\![s.lrange]\!]_{\mathcal{E}xpr^{41}}\ rel1\ x \wedge x\ rel2\ [\![s.rrange]\!]_{\mathcal{E}xpr^{42}}\}$

where

$rel1 = \text{if } (e.linterval = [) \text{ then } \leq \text{ else } <$

$rel2 = \text{if } (e.linterval =]) \text{ then } \geq \text{ else } >$

Rule 25. Semantics of sequence expression

$[\![s : SeqExp]\!]_{\mathcal{E}xpr} : CSPExpression =$

$\langle \{x : s.values \bullet [\![x]\!]_{\mathcal{E}xpr^{43}}\} \rangle$

Rule 26. Semantics of set expression

$[\![s : SetExp]\!]_{\mathcal{E}xpr} : CSPExpression =$

$\{\{x : s.values \bullet [\![x]\!]_{\mathcal{E}xpr^{44}}\}\}$

Rule 27. Semantics of tuple expression

$[\![s : TupleExp]\!]_{\mathcal{E}xpr} : CSPExpression =$

$(\{x : s.values \bullet [\![x]\!]_{\mathcal{E}xpr^{45}}\})$

3.2 Detailed Semantics: Timed Language

This section specifies the functions *alphaClockReset*[1], *compileWC*[1] and *wc*[1] as functions of the RoboTool specific metamodel.

Clocks

Functions related to clocks are formalised in this section.

Rule 28. alphaClockReset function

$alphaClockReset(e : ParExp) : ChannelSet =$

$alphaClockReset(e) = alphaClockReset^{2}(e.exp)$

Rule 29. alphaClockReset function

$alphaClockReset(e : Not) : ChannelSet =$

$alphaClockReset(e) = alphaClockReset^{3}(e.exp)$

Rule 30. alphaClockReset function

$alphaClockReset(e : CallExp) : ChannelSet =$

$alphaClockReset(e) = alphaClockResetCallArgs^{1}(e.args)$

Rule 31. alphaClockResetCallArgs function

$alphaClockResetCallArgs(s : seq\, Expression) : ChannelSet =$

$if\ \#(s) > 0\ then$
  $alphaClockReset^{4}(head(s)) \cup alphaClockResetCallArgs^{2}(tail(s))$
$else$
  $\emptyset$
$endif$

Rule 32. alphaClockReset function
alphaClockReset(e : And) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{5}(e.left) ∪ alphaClockReset^{6}(e.right))

Rule 33. alphaClockReset function
alphaClockReset(e : Or) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{7}(e.left) ∪ alphaClockReset^{8}(e.right))

Rule 34. alphaClockReset function
alphaClockReset(e : Implies) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{9}(e.left) ∪ alphaClockReset^{10}(e.right))

Rule 35. alphaClockReset function
alphaClockReset(e : Iff) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{11}(e.left) ∪ alphaClockReset^{12}(e.right))

Rule 36. alphaClockReset function
alphaClockReset(e : GreaterThan) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{13}(e.left) ∪ alphaClockReset^{14}(e.right))

Rule 37. alphaClockReset function
alphaClockReset(e : GreaterOrEqual) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{15}(e.left) ∪ alphaClockReset^{16}(e.right))

Rule 38. alphaClockReset function
alphaClockReset(e : LessThan) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{17}(e.left) ∪ alphaClockReset^{18}(e.right))

Rule 39. alphaClockReset function
alphaClockReset(e : LessOrEqual) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{19}(e.left) ∪ alphaClockReset^{20}(e.right))

Rule 40. alphaClockReset function
alphaClockReset(e : Equals) : ChannelSet =

alphaClockReset(e) = alphaClockReset^{21}(e.left) ∪ alphaClockReset^{22}(e.right))

Rule 41. alphaClockReset function

alphaClockReset(e : ClockExp) : ChannelSet =

alphaClockReset(e) = {|*clockReset*.id(e.clock)|}

Rule 42. alphaClockReset function

alphaClockReset(e : StateClockExp) : ChannelSet =

alphaClockReset(e) = {|x : SIDS |*entered*.id(x).id(e.state)|}

Waiting Condition elicitation

The following rules defined wc used for eliciting waiting conditions.

Rule 43. wc function

wc(e : Expression) : (Expression, Expression ↛ Variable) =

$wc((e)) = ((\pi_1(wc(e))), \pi_2(wc(e)))$
$wc(\neg e) = (\neg \pi_1(wc(e)), \pi_2(wc(e)))$
$wc(f(args)) = (f(\pi_1(wcArgSeq^1(args))), \pi_2(wcArgSeq^2(args)))$
$wc(e_1 \wedge e_2) = (\pi_1(wc(e_1)) \wedge \pi_1(wc(e_1)), \pi_2(wc(e_1)) \cup \pi_2(wc(e_2)))$
$wc(e_1 \vee e_2) = (\pi_1(wc(e_1)) \vee \pi_1(wc(e_1)), \pi_2(wc(e_1)) \cup \pi_2(wc(e_2)))$
$wc(e_1 => e_2) = (\pi_1(wc(e_1)) => \pi_1(wc(e_1)), \pi_2(wc(e_1)) \cup \pi_2(wc(e_2)))$
$wc(e_1 \text{ iff } e_2) = (\pi_1(wc(e_1)) \text{ iff } \pi_1(wc(e_1)), \pi_2(wc(e_1)) \cup \pi_2(wc(e_2)))$
$wc(since(C) > e) = (b, \{since(C) > e\} \mapsto b)$
$wc(sinceEntry(S) > e) = (b, \{sinceEntry(S) > e\} \mapsto b)$
$wc(e > since(C)) = (b, \{e > since(C)\} \mapsto b)$
$wc(e > sinceEntry(S)) = (b, \{e > sinceEntry(S)\} \mapsto b)$
$wc(e_1 > e_2) = (e_1 > e_2, \emptyset)$
$wc(since(C) >= e) = (b, \{since(C) >= e\} \mapsto b)$
$wc(sinceEntry(S) >= e) = (b, \{sinceEntry(S) >= e\} \mapsto b)$
$wc(e >= since(C)) = (b, \{e >= since(C)\} \mapsto b)$
$wc(e >= sinceEntry(S)) = (b, \{e >= sinceEntry(S)\} \mapsto b)$
$wc(e_1 >= e_2) = (e_1 >= e_2, \emptyset)$
$wc(since(C) < e) = (b, \{since(C) < e\} \mapsto b)$
$wc(sinceEntry(S) < e) = (b, \{sinceEntry(S) < e\} \mapsto b)$
$wc(e < since(C)) = (b, \{e < since(C))\} \mapsto b)$
$wc(e < sinceEntry(S)) = (b, \{e < sinceEntry(S)\} \mapsto b)$
$wc(e_1 < e_2) = (e_1 < e_2, \emptyset)$
$wc(since(C) <= e) = (b, \{since(C) <= e\} \mapsto b)$
$wc(sinceEntry(S) <= e) = (b, \{sinceEntry(S) <= e\} \mapsto b)$
$wc(e <= since(C)) = (b, \{e <= since(C)\} \mapsto b)$
$wc(e <= sinceEntry(S)) = (b, \{e <= sinceEntry(S)\} \mapsto b)$
$wc(e_1 <= e_2) = (e_1 <= e_2, \emptyset)$
$wc(since(C) == e) = (b, \{since(C) == e\} \mapsto b)$
$wc(sinceEntry(S) == e) = (b, \{sinceEntry(S) == e\} \mapsto b)$
$wc(e == since(C)) = (b, \{e == since(C)\} \mapsto b)$
$wc(e == sinceEntry(S)) = (b, \{e == sinceEntry(S)\} \mapsto b)$
$wc(e_1 == e_2) = (e_1 == e_2, \emptyset)$

where

b is a fresh identifier.

Rule 44. wcArgSeq function

$\mathsf{wcArgSeq}(s : \mathsf{seq}(\mathsf{Expression})) : (\mathsf{seq}\,\mathsf{Expression}, \mathsf{Expression} \nrightarrow \mathsf{Variable}) =$

$\mathsf{if}\ \#(s) > 0\ \mathsf{then}$

$\quad (\langle \pi_1(\mathsf{wc}_{\mathsf{head}}) \rangle \frown \pi_1(\mathsf{wc}_{\mathsf{tail}}), \pi_2(\mathsf{wc}_{\mathsf{head}}) \cup \pi_2(\mathsf{wc}_{\mathsf{tail}}))$

$\quad \mathbf{where}$

$\quad\quad \mathsf{wc}_{\mathsf{head}} = \mathsf{wc}^2(\mathsf{head}(s))$

$\quad\quad \mathsf{wc}_{\mathsf{tail}} = \mathsf{wcArgSeq}^3(\mathsf{tail}(s))$

else

$\quad (\langle\rangle, \emptyset)$

endif

Waiting Condition as CSP processes

The following rules define the function compileWC which is used to define the CSP semantics of waiting conditions.

Rule 45. compileWC function

$\mathsf{compileWC}(t : \mathsf{Transition}, e : \mathsf{Expression}, v : \mathsf{Variable}) : \mathsf{TimedCSPProcess} =$

$\mathsf{compileWC}(t, \mathsf{since}(C) >= e, v) =$

let

$Reset = clockReset.\mathsf{id}(C) \rightarrow setWC\_\mathsf{vid}(v)!false \rightarrow Monitor$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^1(t)|\}) \\ \quad \underset{\mathcal{E}xpr^{\#6}}{\triangle_{[\![e]\!]}} \; setWC\_\mathsf{vid}(v)!true \rightarrow RUN(\{|\mathsf{triggerEvent}^2(t)|\}) \end{array} \right) \triangle Reset$$

within

$setWC\_\mathsf{vid}(v)!false \rightarrow Monitor$

$\mathsf{compileWC}(t, e >= \mathsf{since}(C), v) =$

let

$Reset = clockReset.\mathsf{id}(C) \rightarrow setWC\_\mathsf{vid}(v)!true \rightarrow Monitor$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^3(t)|\}) \\ \quad \underset{\mathcal{E}xpr^{\#7}}{\triangle_{[\![e]\!]}} \; setWC\_\mathsf{vid}(v)!false \rightarrow RUN(\{|\mathsf{triggerEvent}^4(t)|\}) \end{array} \right) \triangle Reset$$

within

$setWC\_\mathsf{vid}(v)!true \rightarrow Monitor$

$\mathsf{compileWC}(t, \mathsf{sinceEntry}(S) >= e, v) =$

let

$Reset = entered?x.\mathsf{id}(S) \rightarrow setWC\_\mathsf{vid}(v)!false \rightarrow Monitor$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^5(t)|\}) \\ \quad \underset{\mathcal{E}xpr^{\#8}}{\triangle_{[\![e]\!]}} \; setWC\_\mathsf{vid}(v)!true \rightarrow RUN(\{|\mathsf{triggerEvent}^6(t)|\}) \end{array} \right) \triangle Reset$$

within

$setWC\_\mathsf{vid}(v)!false \rightarrow Monitor$

Rule 46. compileWC function

compileWC(t : Transition, e : Expression, v : Variable) : TimedCSPProcess =

compileWC(t, e >= sinceEntry(S), v) =

let

$$Reset = entered?x.\mathrm{id}(S) \rightarrow setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{7}(t)|\}) \\ \quad \triangle_{[\![e]\!]_{Expr^{49}}} \; setWC\_\mathrm{vid}(v)!false \rightarrow RUN(\{|\mathrm{triggerEvent}^{8}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

compileWC(t, since(C) >= e, v) =

let

$$Reset = clockReset.\mathrm{id}(C) \rightarrow setWC\_\mathrm{vid}(v)!false \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{9}(t)|\}) \\ \quad \triangle_{[\![e]\!]_{Expr^{50}}} \; +1 \; setWC\_\mathrm{vid}(v)!true \rightarrow RUN(\{|\mathrm{triggerEvent}^{10}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!false \rightarrow Monitor$$

compileWC(t, e > since(C), v) =

let

$$Reset = clockReset.\mathrm{id}(C) \rightarrow setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{11}(t)|\}) \\ \quad \triangle_{[\![e]\!]_{Expr^{51}}} \; +1 \; setWC\_\mathrm{vid}(v)!false \rightarrow RUN(\{|\mathrm{triggerEvent}^{12}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

compileWC(t, sinceEntry(S) > e, v) =

let

$$Reset = entered?x.\mathrm{id}(S) \rightarrow setWC\_\mathrm{vid}(v)!false \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{13}(t)|\}) \\ \quad \triangle_{[\![e]\!]_{Expr^{52}}} \; +1 \; setWC\_\mathrm{vid}(v)!true \rightarrow RUN(\{|\mathrm{triggerEvent}^{14}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!false \rightarrow Monitor$$

compileWC(t, e > sinceEntry(S), v) =

let

$$Reset = entered?x.\mathrm{id}(S) \rightarrow setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{15}(t)|\}) \\ \quad \triangle_{[\![e]\!]_{Expr^{53}}} \; +1 \; setWC\_\mathrm{vid}(v)!false \rightarrow RUN(\{|\mathrm{triggerEvent}^{16}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

Rule 47. compileWC function

compileWC(t : Transition, e : Expression, v : Variable) : TimedCSPProcess =

compileWC(t, since(C) <= e, v) =

let

$$Reset = clockReset.\mathrm{id}(C) \rightarrow setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{17}(t)|\}) \\ \quad \triangle_{\substack{[\![e]\!] \\ \mathcal{E}xpr^{54}}} \; setWC\_\mathrm{vid}(v)!false \rightarrow RUN(\{|\mathrm{triggerEvent}^{18}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

compileWC(t, e <= since(C), v) =

let

$$Reset = clockReset.\mathrm{id}(C) \rightarrow setWC\_\mathrm{vid}(v)!false \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{19}(t)|\}) \\ \quad \triangle_{\substack{[\![e]\!] \\ \mathcal{E}xpr^{55}}} \; setWC\_\mathrm{vid}(v)!true \rightarrow RUN(\{|\mathrm{triggerEvent}^{20}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!false \rightarrow Monitor$$

compileWC(t, sinceEntry(S) <= e, v) =

let

$$Reset = entered?x.\mathrm{id}(S) \rightarrow setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{21}(t)|\}) \\ \quad \triangle_{\substack{[\![e]\!] \\ \mathcal{E}xpr^{56}}} \; setWC\_\mathrm{vid}(v)!false \rightarrow RUN(\{|\mathrm{triggerEvent}^{22}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

compileWC(t, e <= sinceEntry(S), v) =

let

$$Reset = entered?x.\mathrm{id}(S) \rightarrow setWC\_\mathrm{vid}(v)!false \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{23}(t)|\}) \\ \quad \triangle_{\substack{[\![e]\!] \\ \mathcal{E}xpr^{57}}} \; setWC\_\mathrm{vid}(v)!true \rightarrow RUN(\{|\mathrm{triggerEvent}^{24}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!false \rightarrow Monitor$$

compileWC(t, since(C) < e, v) =

let

$$Reset = clockReset.\mathrm{id}(C) \rightarrow setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathrm{triggerEvent}^{25}(t)|\}) \\ \quad \triangle_{\substack{([\![e]\!]+1) \\ \mathcal{E}xpr^{58}}} \; setWC\_\mathrm{vid}(v)!false \rightarrow RUN(\{|\mathrm{triggerEvent}^{26}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathrm{vid}(v)!true \rightarrow Monitor$$

Rule 48. compileWC function

$\mathsf{compileWC}(\mathsf{t} : \mathsf{Transition}, \mathsf{e} : \mathsf{Expression}, \mathsf{v} : \mathsf{Variable}) : \mathsf{TimedCSPProcess} =$

$\mathsf{compileWC}(\mathsf{t}, \mathsf{e} < \mathsf{since}(\mathsf{C}), \mathsf{v}) =$

let

$Reset = clockReset.\mathsf{id}(\mathsf{C}) \rightarrow setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow Monitor$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^{27}(\mathsf{t})|\}) \\ \quad \triangle_{(\underbrace{[\![\mathsf{e}]\!]}_{\mathcal{E}xpr^{59}}+1)}\ setWC\_\mathsf{vid}(\mathsf{v})!true \rightarrow RUN(\{|\mathsf{triggerEvent}^{28}(\mathsf{t})|\}) \end{array} \right) \triangle Reset$$

within

$setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow Monitor$

$\mathsf{compileWC}(\mathsf{t}, \mathsf{sinceEntry}(\mathsf{S}) < \mathsf{e}, \mathsf{v}) =$

let

$Reset = entered?x.\mathsf{id}(\mathsf{S}) \rightarrow setWC\_\mathsf{vid}(\mathsf{v})!true \rightarrow Monitor$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^{29}(\mathsf{t})|\}) \\ \quad \triangle_{(\underbrace{[\![\mathsf{e}]\!]}_{\mathcal{E}xpr^{60}}+1)}\ setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow RUN(\{|\mathsf{triggerEvent}^{30}(\mathsf{t})|\}) \end{array} \right) \triangle Reset$$

within

$setWC\_\mathsf{vid}(\mathsf{v})!true \rightarrow Monitor$

$\mathsf{compileWC}(\mathsf{t}, \mathsf{e} < \mathsf{sinceEntry}(\mathsf{S}), \mathsf{v}) =$

let

$Reset = entered?x.\mathsf{id}(\mathsf{S}) \rightarrow setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow Monitor$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^{31}(\mathsf{t})|\}) \\ \quad \triangle_{(\underbrace{[\![\mathsf{e}]\!]}_{\mathcal{E}xpr^{61}}+1)}\ setWC\_\mathsf{vid}(\mathsf{v})!true \rightarrow RUN(\{|\mathsf{triggerEvent}^{32}(\mathsf{t})|\}) \end{array} \right) \triangle Reset$$

within

$setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow Monitor$

$\mathsf{compileWC}(\mathsf{t}, \mathsf{since}(\mathsf{C}) == \mathsf{e}, \mathsf{v}) =$

let

$Reset = clockReset.\mathsf{id}(\mathsf{C}) \rightarrow setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow Monitor$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^{33}(\mathsf{t})|\}) \\ \quad \triangle_{\underbrace{[\![\mathsf{e}]\!]}_{\mathcal{E}xpr^{62}}}\ setWC\_\mathsf{vid}(\mathsf{v})!true \rightarrow RUN(\{|\mathsf{triggerEvent}^{34}(\mathsf{t})|\}) \\ \quad \triangle_{(\underbrace{[\![\mathsf{e}]\!]}_{\mathcal{E}xpr^{63}}+1)}\ setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow RUN(\{|\mathsf{triggerEvent}^{35}(\mathsf{t})|\}) \end{array} \right) \triangle Reset$$

within

$setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow Monitor$

$\mathsf{compileWC}(\mathsf{t}, \mathsf{e} == \mathsf{since}(\mathsf{C}), \mathsf{v}) =$

let

$Reset = clockReset.\mathsf{id}(\mathsf{C}) \rightarrow setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow Monitor$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^{36}(\mathsf{t})|\}) \\ \quad \triangle_{\underbrace{[\![\mathsf{e}]\!]}_{\mathcal{E}xpr^{64}}}\ setWC\_\mathsf{vid}(\mathsf{v})!true \rightarrow RUN(\{|\mathsf{triggerEvent}^{37}(\mathsf{t})|\}) \\ \quad \triangle_{(\underbrace{[\![\mathsf{e}]\!]}_{\mathcal{E}xpr^{65}}+1)}\ setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow RUN(\{|\mathsf{triggerEvent}^{38}(\mathsf{t})|\}) \end{array} \right) \triangle Reset$$

within

$setWC\_\mathsf{vid}(\mathsf{v})!false \rightarrow Monitor$

Rule 49. compileWC function

compileWC(t : Transition, e : Expression, v : Variable) : TimedCSPProcess =

compileWC(t, sinceEntry(C) == e, v) =

let

$$Reset = entered?x.\mathsf{id}(S) \rightarrow setWC\_\mathsf{vid}(v)!false \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^{39}(t)|\}) \\ \quad \triangle_{[\![e]\!]_{\mathcal{E}xpr^{66}}} setWC\_\mathsf{vid}(v)!true \rightarrow RUN(\{|\mathsf{triggerEvent}^{40}(t)|\}) \\ \quad \triangle_{([\![e]\!]_{\mathcal{E}xpr^{67}}+1)} setWC\_\mathsf{vid}(v)!false \rightarrow RUN(\{|\mathsf{triggerEvent}^{41}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathsf{vid}(v)!false \rightarrow Monitor$$

compileWC(t, e == sinceEntry(S), v) =

let

$$Reset = entered?x.\mathsf{id}(S) \rightarrow setWC\_\mathsf{vid}(v)!false \rightarrow Monitor$$

$$Monitor = \left(\begin{array}{l} RUN(\{|\mathsf{triggerEvent}^{42}(t)|\}) \\ \quad \triangle_{[\![e]\!]_{\mathcal{E}xpr^{68}}} setWC\_\mathsf{vid}(v)!true \rightarrow RUN(\{|\mathsf{triggerEvent}^{43}(t)|\}) \\ \quad \triangle_{([\![e]\!]_{\mathcal{E}xpr^{69}}+1)} setWC\_\mathsf{vid}(v)!false \rightarrow RUN(\{|\mathsf{triggerEvent}^{44}(t)|\}) \end{array} \right) \triangle Reset$$

within

$$setWC\_\mathsf{vid}(v)!false \rightarrow Monitor$$

4. Probabilistic Semantics

In this chapter, we specify the probabilistic semantics of types and expressions as functions of the RoboTool specific metamodel.

4.1 Types

Rule 50. **Types**

$[\![\text{type} : \text{Type}]\!]_t : \text{Type}_{\text{pr}} =$

if type ∈ TypeRef
  if type.ref ∈ PrimitiveType
    if type.ref.name = "int"
      $\lfloor \text{nat} \rightsquigarrow \text{false} \rfloor_{\text{IntType}}$
    else if type.ref.name = "nat"
      $\lfloor \text{nat} \rightsquigarrow \text{true} \rfloor_{\text{IntType}}$
    else if type.ref.name = "boolean"
      $\lfloor \rfloor_{\text{BoolType}}$
    else if type.ref.name = "real"
      $\lfloor \rfloor_{\text{DoubleType}}$
    else
      Error: Not yet supported!
  else if type.ref ∈ Enumeration
    $\lfloor \text{bottom} \rightsquigarrow 0, \text{top} \rightsquigarrow \text{type.ref.constants.size}() - 1 \rfloor_{\text{RangeType}}$
  else
    Error: Not yet supported!
else
  Error: Not yet supported!

4.2 Expressions

Rule 51. Expressions

$[\![e : \text{Expression}]\!]_e : \text{Expr}_{pr} =$

if e ∈ RefExp **then** $[\![e.ref]\!]_{e^1}$ **else if** e ∈ ParExp **then** $[\![e.exp]\!]_{e^2}$

else if e ∈ Iff **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^3}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^4} \rfloor_{\text{IffBoolExpr}}$

else if e ∈ Implies **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^5}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^6} \rfloor_{\text{ImplBoolExpr}}$

else if e ∈ Or **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^7}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^8} \rfloor_{\text{OrBoolExpr}}$

else if e ∈ And **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^9}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{10}} \rfloor_{\text{AndBoolExpr}}$

else if e ∈ Equals **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{11}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{12}} \rfloor_{\text{EqBoolExpr}}$

else if e ∈ Different **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{13}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{14}} \rfloor_{\text{NeqBoolExpr}}$

else if e ∈ GreaterThan **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{15}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{16}} \rfloor_{\text{GreaterBoolExpr}}$

else if e ∈ GreaterOrEqual **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{17}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{18}} \rfloor_{\text{GreaterEqBoolExpr}}$

else if e ∈ LessThan **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{19}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{20}} \rfloor_{\text{LessBoolExpr}}$

else if e ∈ LessOrEqual **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{21}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{22}} \rfloor_{\text{LessEqBoolExpr}}$

else if e ∈ IfExpression **then**

$\lfloor \text{cond} \rightsquigarrow [\![e.condition]\!]_{e^{23}}, \text{left} \rightsquigarrow [\![e.ifexp]\!]_{e^{24}}, \text{right} \rightsquigarrow [\![e.elseexp]\!]_{e^{25}} \rfloor_{\text{CondBoolExpr}}$

else if e ∈ Not **then** $\lfloor \text{value} \rightsquigarrow [\![e.exp]\!]_{e^{26}} \rfloor_{\text{NotBoolExpr}}$

else if e ∈ Plus **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{27}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{28}} \rfloor_{\text{PlusExpr}}$

else if e ∈ Minus **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{29}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{30}} \rfloor_{\text{MinusExpr}}$

else if e ∈ Mult **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{31}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{32}} \rfloor_{\text{MultExpr}}$

else if e ∈ Div **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{33}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{34}} \rfloor_{\text{DivExpr}}$

else if e ∈ Modulus **then** $\lfloor \text{left} \rightsquigarrow [\![e.left]\!]_{e^{35}}, \text{right} \rightsquigarrow [\![e.right]\!]_{e^{36}} \rfloor_{\text{ModuloExpr}}$

else if e ∈ Neg **then** $\lfloor \text{value} \rightsquigarrow [\![e.exp]\!]_{e^{37}} \rfloor_{\text{UMinusExpr}}$

else if e ∈ IntegerExp **then** $\lfloor \text{value} \rightsquigarrow e.value \rfloor_{\text{IntLitExpr}}$

else if e ∈ FloatExp **then** $\lfloor \text{value} \rightsquigarrow e.value \rfloor_{\text{DoubleLitExpr}}$

else if e ∈ BooleanExp **then** $\lfloor \text{value} \rightsquigarrow (\textbf{if } e.value = \text{“true”} \textbf{ then } \text{true } \textbf{else } \text{false}) \rfloor_{\text{BoolLitExpr}}$

else if e ∈ VarExp **then** $\lfloor \text{name} \rightsquigarrow id(e.value.name) \rfloor_{\text{VarExpr}}$

else if e ∈ CallExp **then** $\lfloor \text{name} \rightsquigarrow e.function.ref.name, \text{args} \rightsquigarrow \{a : e.args \bullet [\![a]\!]_{e^{38}}\} \rfloor_{\text{CallExpr}}$

II RoboSim Physical Modelling

5. Mapping to SDF

This chapter specifies the evaluator of expressions as a function of the RoboTool specific metamodel.

Rule 52. Negation Expression

$$[\![\text{e} : \text{Neg}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$-\underbrace{[\![\text{e.exp}]\!]^{\text{cvals}}_{\mathcal{EXP}^1}}$$

Rule 53. Plus Expression

$$[\![\text{e} : \text{Plus}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\underbrace{[\![\text{e.left}]\!]^{\text{cvals}}_{\mathcal{EXP}^2}} + \underbrace{[\![\text{e.right}]\!]^{\text{cvals}}_{\mathcal{EXP}^3}}$$

Rule 54. Minus Expression

$$[\![\text{e} : \text{Minus}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\underbrace{[\![\text{e.left}]\!]^{\text{cvals}}_{\mathcal{EXP}^4}} - \underbrace{[\![\text{e.right}]\!]^{\text{cvals}}_{\mathcal{EXP}^5}}$$

Rule 55. Multiplication Expression

$$[\![\text{e} : \text{Mult}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\underbrace{[\![\text{e.left}]\!]^{\text{cvals}}_{\mathcal{EXP}^6}} * \underbrace{[\![\text{e.right}]\!]^{\text{cvals}}_{\mathcal{EXP}^7}}$$

Rule 56. Division Expression

$[\![\text{e} : \text{Div}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$

$[\![\text{e.left}]\!]^{\text{cvals}}_{\mathcal{EXP}^{8}} / [\![\text{e.right}]\!]^{\text{cvals}}_{\mathcal{EXP}^{9}}$

Rule 57. Modulus Expression

$[\![\text{e} : \text{Modulus}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$

$[\![\text{e.left}]\!]^{\text{cvals}}_{\mathcal{EXP}^{10}} \,\%\, [\![\text{e.right}]\!]^{\text{cvals}}_{\mathcal{EXP}^{11}}$

Rule 58. Paranthesised Expression

$[\![\text{e} : \text{ParExp}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$

$[\![\text{e.exp}]\!]^{\text{cvals}}_{\mathcal{EXP}^{12}}$

Rule 59. Range Expression

$[\![\text{e} : \text{RangeExp}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$

$\{\{\text{v} : \text{IntegerValue} \mid \text{left} \leq \text{v} \leq \text{right} \bullet \text{v}\}\}$
where
lrange $= [\![\text{e.lrange}]\!]^{\text{cvals}}_{\mathcal{EXP}^{13}}$
rrange $= [\![\text{e.rrange}]\!]^{\text{cvals}}_{\mathcal{EXP}^{14}}$
lopen $=$ **if** e.linterval $=$ `(` **then** `true`
else `false`
ropen $=$ **if** e.rinterval $=$ `)` **then** `true`
else `false`
left $=$ **if** lopen $=$`false` **then** lrangle.value **else** lrange.value $+ 1$
right $=$ **if** ropen $=$`false` **then** rlrangle.value **else** rrange.value $- 1$

Rule 60. Ref Expression

$[\![\text{e} : \text{RefExp}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$

if e.ref $\in$ Variable $\wedge$ e.ref.modifier $=$ VariableModifier.CONST **then**
if e.ref.initial $\neq$ null **then** $[\![\text{e.ref.initial}]\!]^{\text{cvals}}_{\mathcal{EXP}^{15}}$ **else** cvals(e.ref)
else if e.ref $\in$ Index **then** cvals(e.ref)
else 0.0

Rule 61. Integer Expression

$[\![\text{e} : \text{IntegerExp}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$

IntegerValue(e.value)

Rule 62. Double Expression

$$[\![e : \text{DoubleExp}]\!]^{cvals}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\text{DoubleValue}(e.\text{value})$$

Rule 63. String Expression

$$[\![e : \text{StringExp}]\!]^{cvals}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\text{StringValue}(e.\text{value})$$

Rule 64. Boolean Expression

$$[\![e : \text{BooleanExp}]\!]^{cvals}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\textbf{if } e.\text{value} = \text{TRUE } \textbf{then } \texttt{true } \textbf{else } \texttt{false}$$

Rule 65. Tuple Expression

$$[\![e : \text{TupleExp}]\!]^{cvals}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\underset{\mathcal{EXP}^{16}}{(\{v : e.\text{values} \bullet [\![v]\!]^{cvals}\})}$$

Rule 66. Set Expression

$$[\![e : \text{SetExp}]\!]^{cvals}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\underset{\mathcal{EXP}^{17}}{\{\{v : e.\text{values} \bullet [\![v]\!]^{cvals}\}\}}$$

Rule 67. Sequence Expression

$$[\![e : \text{SeqExp}]\!]^{cvals}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\underset{\mathcal{EXP}^{18}}{\langle\{v : e.\text{values} \bullet [\![v]\!]^{cvals}\}\rangle}$$

Rule 68. Unit Expression

$$[\![e : \text{UnitExp}]\!]^{cvals}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$$

$$\textbf{if } e.\text{unit} \in \text{UnitReference } \textbf{then}$$
$$\quad \textbf{if } e.\text{unit.ref} \in \text{DerivedUnitDefinition } \textbf{then}$$
$$\quad\quad \underset{\mathcal{EXP}^{19}}{[\![e.\text{exp}]\!]^{cvals}} * \underset{\mathcal{EXP}^{20}}{[\![e.\text{unit.relation}]\!]^{cvals}}$$
$$\quad \textbf{else } \underset{\mathcal{EXP}^{21}}{[\![e.\text{exp}]\!]^{cvals}}$$
$$\textbf{else } \underset{\mathcal{EXP}^{22}}{[\![e.\text{exp}]\!]^{cvals}}$$

Rule 69. Call Expression

$[\![\text{e} : \text{CallExp}]\!]^{\text{cvals}}_{\mathcal{EXP}} : (\text{Variable} \nrightarrow \text{Value}) \nrightarrow \text{Value} =$

```
if e.function ∈ RefExp then
  if e.function.ref ∈ Function then
    if e.function.ref.name = sin then sin(e.args.1.value)
    else if e.function.ref.name = cos then cos(e.args.1.value)
    else 0.0
  else 0.0
else 0.0
```

6. Tool support

As indicated in Chapter **??**, RoboSim (and RoboChart) are supported by an Eclipse-based [3] tool called RoboTool. We have extended it to support block diagrams in RoboSim as described here. RoboTool, and all the examples discussed here, including that in [**BLMRBRVBM10**], are all available[1].

RoboTool has been implemented as a number of Eclipse plugins, which provide various facilities such as parsers, type-checkers, validator, graphical editors, and code and mathematical model generators. Physical-modelling plugins to RoboTool provide facilities for parsing, type checking, validation, graphical editing, and model generation. Parsing and type checking extends the existing parsers and type checkers for RoboChart and RoboSim, and the validator implements the well-formedness conditions in Section **??**. The graphical editor provides two types of block diagrams, one for describing p-models and the other for platform mappings as described in Chapter **??**.

The code generators for the physical-modelling plugins follow the same patterns used for RoboChart, and use extension points to allow user to contribute new generator. We have implemented two model generators described next. The SDF generator (Section 6.1) implements the rules in Chapter **??**, and the *CyPhyCircus* generator (Section 6.2) produces the semantics in Chapter **??**. The generated SDF document can be combined with the automatically generated code for RoboSim modules to yield simulations. The *CyPhyCircus* models are for use with the theorem prover Isabelle/UTP [**FBCMW18**]. We plan to extend our generators to support other technologies, including hybrid model checkers.

6.1 SDF generator

With RoboTool, we can design a p-model in the graphical environment of RoboSim, and create an SDF file by pressing a button. The document generated can be imported into CoppeliaSim to visualize and use in a simulation. The implementation uses Xtend [10] as a model-to-text transformer, so that the rules are in direct

[1] https://www.cs.york.ac.uk/robostar/notations-tools/

correspondence with the code. RoboTool, as well providing support for the practical application of our work, also validates our rules.

Since, as already mentioned, a p-model may be defined in terms of constants whose values are not given, the SDF generator asks for a value for these constants. This means we can easily generate multiple documents by varying the values of constants, and it is particularly useful for use of the simulation in the context of evolutionary robotics and design-space exploration.

Some aspects of the implementation needed to be tailored to CoppeliaSim. (So, we envisage that other very similar generators may become useful to deal with different simulators.) For example, qualified names can become long, and CoppeliaSim imposes a limit on the size of elements names. RoboTool creates coded short and still unique names for the elements, and produces, alongside the SDF document, a table that maps the qualified names to the coded names (and vice-versa).

In addition, we have used SDF 1.6, the latest version supported by CoppeliaSim. The more recent SDF 1.7 comes with semantics for the `frame` element, and supports custom elements and attributes. As already said, RoboSim, for clarity, fixes the frame of reference for poses using the containment relationship, and our translation does not make use of ad hoc frame elements. So, this change has no impact in our translation. The possibility to define custom elements and attributes also has no impact, but this facility may make annotations even more useful, since they are tied to any SDF element. In summary, apart from changing the definition of the `version` attribute in Rule **??**, there is nothing that needs changing to obtain SDF 1.7 documents.

Finally, CoppeliaSim does not implement the SDF actuators. Their inclusion in documents, however, does not cause problems. If we use Gazebo [8], for example, the definition of actuators is useful.

6.2 CyPhyCircus generator and equation solver

The *CyPhyCircus* generator produces the models described in Chapter **??**, suitable for analysis using Isabelle/UTP [**FBCMW18**]. It outputs model files for well-formed p-models in the folder `cpc-gen`. RoboTool uses Eclipse's extension points to support implementation of code generators.

Because Isabelle/UTP is a full-fledged theorem prover (as opposed to a finite-state model checker such as FDR or the Simulink [9] verifier), the models produced can be reasoned without extra information being provided by the user. For instance, our p-models allow uninitialised constants in a p-model connected to a platform mapping, allowing the effective modelling of a family of p-models, which can then be analysed to identify, for instance, restriction over these constants that guarantee a particular property holds. For other targets such as hybrid model-checkers and simulators, extra information must be provided to the generators to fully-instantiate the p-model.

The *CyPhyCircus* generator calculates the processes described in Chapter **??**, but delegates the task of formulating the system of equations to a specialised tool, a SymPy-based solver as described in Section **??**. It takes into account that some of the equations that describe the dynamics of the system are standard (rigid body motion) and not represented in the diagram. By simplifying and solving the equations where possible, the SymPy-based solver simplifies the task of verification.

The SymPy-based solver relies on the `xml.etree`, `sympy`, and `textx` packages. Equations in a p-model are

extracted by parsing the model XML file into a Python ElementTree structure, and traversing it in a breadth-first search. Inputs, outputs, and local variables are declared as functions of time with the SymPy *Function*() method, which creates function expressions. Constants are defined simply with the *symbols*() method as workspace variables. Equations themselves are solved as equalities, which facilitates the use of SymPy solvers. All equations are split across the '=' character into left and right hand sides for the SymPy *Eq*() method. Simple expressions without an '=' character are assumed to be equal to zero and are evaluated as such by methods in `sympy.solvers`. However, multiple equalities in one line are not currently supported.

Solution is performed by iterating thought the list of outputs in each element, and searching the list of equalities for variables that match the needed outputs. All equations are defined in the Python workspace using the `exec()` method directly on their text strings. This allows SymPy to do the work of simplifying equations with common variables into their final forms. The `solveset()` method is used to algebraically solve simple equations for output variables. Differential equations are identified from their semantics in the p-model and are solved separately using the `dsolve()` method to obtain a symbolic expression for the solution of a differential equation.

Symbolic solutions are converted into discrete variables and operations through substring search, and translated back into the RoboSim XML schema using the Python textX meta-language for seamless re-integration with the p-model. If no equations or corresponding variables are provided, none will be produced by the solver as a logical open-world assumption implies that no statement about solution will be made if no information is present.

III Appendices

A. Complete RoboChart Metamodel

This appendix contains the complete metamodel of RoboChart specified in Ecore and formatted by the tool `OCLinEcore`. The syntax of the representation used in this appendix is available here.

A summary of the concepts of Ecore can be found here, and a tutorial is available here.

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore' ;

package robochart : robochart = 'http://www.robocalc.circus/RoboChart' {
class RCPackage
{
  attribute name : String[?];
  property imports : Import[*] { composes };
  property interfaces : Interface[*] { composes };
  property robots : RoboticPlatformDef[*] { composes };
  property types : TypeDecl[*] { composes };
  property machines : StateMachineDef[*] { composes };
  property controllers : ControllerDef[*] { composes };
  property modules : RCModule[*] { composes };
  property operations : OperationDef[*] { composes };
  property functions : Function[*] { composes };
}
class Import
{
  attribute importedNamespace : String[1];
}
abstract class NamedElement
{
  attribute name : String[1];
}
abstract class TypeDecl extends NamedElement;
class PrimitiveType extends TypeDecl;
class RecordType extends TypeDecl
{
```

```
  property fields : Field[+] { composes };
}
class Field extends Member,NamedExpression;
abstract class TypedNamedElement extends NamedElement
{
  property type : Type[1] { composes };
}
abstract class Member extends TypedNamedElement;
class Enumeration extends TypeDecl
{
  property literals : Literal[*|1] { composes };
}
class Literal extends TypeDecl,NamedExpression
{
  property types : Type[*] { composes };
}
class NameType extends TypeDecl
{
  property type : Type[1] { composes };
}
abstract class Type;
class ProductType extends Type
{
  property types : Type[2..*] { ordered composes };
}
class RelationType extends Type
{
  property domain : Type[1] { composes };
  property range : Type[1] { composes };
}
class FunctionType extends RelationType;
class SetType extends Type
{
  property domain : Type[1] { composes };
}
class SeqType extends SetType;
class TypeRef extends Type
{
  property ref : TypeDecl[1];
}
class AnyType extends Type
{
  attribute identifier : String[1];
}
class VariableList
{
  attribute modifier : VariableModifier[1];
  property vars : Variable[+] { composes };
}
enum VariableModifier { serializable }
{
  literal VAR;
  literal CONST;
}
```

```
class Variable extends TypedNamedElement,NamedExpression
{
  property initial : Expression[?] { composes };
  attribute modifier : VariableModifier[1] { derived transient volatile };
}
class Event extends NamedElement
{
  property type : Type[?] { composes };
  attribute broadcast : Boolean[1];
}
class Function extends TypedNamedElement,NamedExpression
{
  property parameters : Parameter[*] { ordered composes };
  property preconditions : Expression[*] { composes };
  property postconditions : Expression[*] { composes };
}
class Parameter extends Variable;
class OperationSig extends NamedElement
{
  attribute terminates : Boolean[1];
  property parameters : Parameter[*] { ordered composes };
  property preconditions : Expression[*] { composes };
  property postconditions : Expression[*] { composes };
}
abstract class Operation extends NamedElement,ConnectionNode;
class OperationDef extends Operation,OperationSig,StateMachineBody;
abstract class Reference;
class OperationRef extends Operation,Reference
{
  property ref : OperationDef[1];
}
class Interface extends NamedElement,BasicContext;
abstract class BasicContext
{
  property variableList : VariableList[*] { composes };
  property operations : OperationSig[*] { composes };
  property events : Event[*] { composes };
  property clocks : Clock[*] { composes };
}
abstract class RoboticPlatform extends NamedElement,ConnectionNode;
class RoboticPlatformDef extends Context,RoboticPlatform;
abstract class Context extends BasicContext
{
  property pInterfaces : Interface[*] { !unique };
  property rInterfaces : Interface[*] { !unique };
  property interfaces : Interface[*] { !unique };
}
class RoboticPlatformRef extends RoboticPlatform,Reference
{
  property ref : RoboticPlatformDef[1];
}
abstract class StateMachine extends NamedElement,ConnectionNode;
class StateMachineDef extends StateMachineBody,StateMachine;
class StateMachineRef extends StateMachine,Reference
```

```
{
  property ref : StateMachineDef[1];
}
class StateMachineBody extends Context,NodeContainer;
class Clock extends NamedElement;
abstract class NodeContainer
{
  property nodes : Node[*] { composes };
  property transitions : Transition[*] { composes };
}
abstract class Node extends NamedElement;
class Junction extends Node;
class Initial extends Junction;
class State extends Node,NodeContainer
{
  property actions : Action[*] { composes };
}
class Final extends State;
class ProbabilisticJunction extends Junction;
class Transition extends NamedElement
{
  property source : Node[1];
  property target : Node[1];
  property trigger : Communication[?] { composes };
  property deadline : Expression[?] { composes };
  property condition : Expression[?] { composes };
  property action : Statement[?] { composes };
  property probability : Expression[?] { composes };
  property reset : ClockReset[*] { ordered composes };
}
class Communication
{
  property event : Event[?];
  property _from : Variable[?];
  property _predicate : Expression[?] { composes };
  property parameter : Variable[?];
  property value : Expression[?] { composes };
  attribute _type : TriggerType[1];
}
enum CommunicationType { serializable }
{
  literal SIMPLE;
  literal INPUT;
  literal OUTPUT;
  literal SYNC;
}
abstract class Action
{
  property action : Statement[1] { composes };
}
class EntryAction extends Action;
class DuringAction extends Action;
class ExitAction extends Action;
abstract class Controller extends NamedElement,ConnectionNode;
```

```
class ControllerDef extends Context,Controller
{
  property machines : StateMachine[*] { composes };
  property lOperations : Operation[*] { composes };
  property connections : Connection[*] { composes };
}
class Connection
{
  property from : ConnectionNode[1];
  property to : ConnectionNode[1];
  property efrom : Event[1];
  property eto : Event[1];
  attribute async : Boolean[1];
  attribute bidirec : Boolean[1];
}
class ControllerRef extends Controller
{
  property ref : ControllerDef[1];
}
class RCModule extends NamedElement
{
  property connections : Connection[*] { composes };
  property nodes : ConnectionNode[*] { composes };
}
abstract class Statement;
class TimedStatement extends Statement
{
  property stmt : Statement[1] { composes };
  property deadline : Expression[1] { composes };
}
class Wait extends Statement
{
  property duration : Expression[1] { composes };
}
class Skip extends Statement;
class IfStmt extends Statement
{
  property expression : Expression[1] { composes };
  property _'then' : Statement[1] { composes };
  property _'else' : Statement[?] { composes };
}
class Assignment extends Statement
{
  property left : Assignable[1] { composes };
  property right : Expression[1] { composes };
}
class CommunicationStmt extends Statement
{
  property communication : Communication[1] { composes };
}
class SeqStatement extends Statement
{
  property statements : Statement[2..*] { ordered composes };
}
```

```
class ParStmt extends Statement
{
  property stmt : Statement[1] { composes };
}
class Call extends Statement
{
  property operation : OperationSig[1];
  property args : Expression[*] { ordered composes };
}
class ClockReset extends Statement
{
  property clock : Clock[1];
}
abstract class Expression;
class ResultExp extends Expression;
class ArrayExp extends Expression
{
  property value : Expression[1] { composes };
  property parameters : Expression[*] { composes };
}
class ClockExp extends Expression
{
  property clock : Clock[1];
}
class StateClockExp extends Expression
{
  property state : State[1];
}
abstract class BinaryExpression extends Expression
{
  property left : Expression[1] { composes };
  property right : Expression[1] { composes };
}
class Iff extends BinaryExpression;
class Implies extends BinaryExpression;
class Or extends BinaryExpression;
abstract class QuantifierExpression extends Expression
{
  property variables : Variable[+] { composes };
  property suchthat : Expression[?] { composes };
  property predicate : Expression[1] { composes };
}
class Forall extends QuantifierExpression;
class Exists extends QuantifierExpression
{
  attribute unique : Boolean[1];
}
class LambdaExp extends Expression
{
  property variables : Variable[+] { composes };
  property suchthat : Expression[?] { composes };
  property expression : Expression[1] { composes };
}
class DefiniteDescription extends LambdaExp;
```

```
class IfExpression extends Expression
{
  property condition : Expression[1] { composes };
  property ifexp : Expression[1] { composes };
  property elseexp : Expression[1] { composes };
}
class Declaration extends NamedElement,NamedExpression
{
  property value : Expression[1] { composes };
}
class LetExpression extends Expression
{
  property declarations : Declaration[+] { composes };
  property expression : Expression[1] { composes };
}
class And extends BinaryExpression;
class Not extends Expression
{
  property exp : Expression[1] { composes };
}
class InExp extends Expression
{
  property member : Expression[1] { composes };
  property set : Expression[1] { composes };
}
class TypeExp extends Expression
{
  property type : Type[1] { composes };
}
class Equals extends BinaryExpression;
class Different extends BinaryExpression;
class GreaterThan extends BinaryExpression;
class GreaterOrEqual extends BinaryExpression;
class LessThan extends BinaryExpression;
class LessOrEqual extends BinaryExpression;
class Plus extends BinaryExpression;
class Minus extends BinaryExpression;
class Modulus extends BinaryExpression;
class Mult extends BinaryExpression;
class Div extends BinaryExpression;
class Cat extends BinaryExpression;
class Neg extends Expression
{
  property exp : Expression[1] { composes };
}
class Selection extends Expression
{
  property receiver : Expression[1] { composes };
  property member : Member[1];
}
class IntegerExp extends Expression
{
  attribute value : ecore::EInt[1];
}
```

```
class FloatExp extends Expression
{
  attribute value : ecore::EFloat[1];
}
class StringExp extends Expression
{
  attribute value : String[1];
}
class BooleanExp extends Expression
{
  attribute value : String[1];
}
class VarExp extends Expression
{
  property value : Variable[1];
}
class RefExp extends Expression
{
  property ref : NamedExpression[1];
}
class ToExp extends Expression;
class FromExp extends Expression;
class IdExp extends Expression;
class AsExp extends Expression
{
  property exp : Expression[1] { composes };
  property type : Type[1] { composes };
}
class IsExp extends Expression
{
  property exp : Expression[1] { composes };
  property type : Type[1] { composes };
}
class EnumExp extends Expression
{
  property type : Enumeration[1];
  property literal : Literal[1];
}
class ParExp extends Expression
{
  property exp : Expression[1] { composes };
}
class SeqExp extends Expression
{
  property values : Expression[*] { ordered composes };
}
class SetExp extends Expression
{
  property values : Expression[*] { composes };
}
class SetComp extends Expression
{
  property variables : Variable[+] { composes };
  property predicate : Expression[?] { composes };
```

```
  property expression : Expression[?] { composes };
}
class SetRange extends Expression
{
  property start : Expression[1] { composes };
  property end : Expression[1] { composes };
}
class TupleExp extends Expression
{
  property values : Expression[2..*] { ordered composes };
}
class RangeExp extends Expression
{
  attribute linterval : String[1];
  property lrange : Expression[1] { composes };
  property rrange : Expression[1] { composes };
  attribute rinterval : String[1];
}
class CallExp extends Expression
{
  property function : Expression[1] { composes };
  property args : Expression[*] { ordered composes };
}
class ElseExp extends Expression;
abstract class Assignable;
class VarSelection extends Assignable
{
  property receiver : Assignable[1] { composes };
  property member : Member[1];
}
class ArrayAssignable extends Assignable
{
  property value : Assignable[1] { composes };
  property parameters : Expression[+] { composes };
}
class VarRef extends Assignable
{
  property name : Variable[1];
}
abstract class ConnectionNode;
abstract class NamedExpression;
class WaitingCondition
{
  property expression : Expression[?];
  property transitions : Transition[*] { ordered };
  attribute name : String[?] = '' { id };
}
class WaitingConditionRef extends Expression
{
  property ref : WaitingCondition[?];
}
}
```

B. Complete RoboSim Metamodel

This appendix contains the complete metamodel of RoboSim specified in Ecore and formatted by the tool `OCLinEcore`. The syntax of the representation used in this appendix is available here.

A summary of the concepts of Ecore can be found here, and a tutorial is available here.

```
import robochart : './robochart.ecore#/';

package robosim : robosim = 'http://www.robocalc.circus/RoboSim'
{
  class InputContext extends robochart::Context;
  class OutputContext extends robochart::Context;
  class SimModule extends robochart::RCModule,Cyclic;
  class SimControllerDef extends robochart::ControllerDef,Cyclic;
  class SimMachineDef extends robochart::StateMachineDef,SimContext,Cyclic;
  class SimOperationDef extends robochart::OperationDef,SimContext;
  class ExecTrigger extends robochart::Communication;
  class SimRefExp extends robochart::Expression
  {
    property element : robochart::NamedElement[?];
    property exp : robochart::Expression[?] { composes };
    property variable : robochart::Variable[?];
    property predicate : robochart::Expression[?] { composes };
  }
  class SimCall extends robochart::Call;
  class SimVarRef extends robochart::VarRef;
  class CycleExp extends robochart::Expression;
  class ExecStatement extends robochart::Statement;
  class SimContext extends robochart::Context
  {
    property inputContext : robochart::Context[1] { composes };
    property outputContext : robochart::Context[1] { composes };
  }
  class Cyclic
```

```
  {
    property cycleDef : robochart::Expression[1] { composes };
  }
  class OutputCommunication extends robochart::Statement
  {
    property event : robochart::Event[1];
    property value : robochart::Expression[?] { composes };
  }
}
```

C. Complete Physical Modelling Metamodel

This appendix contains the complete metamodel of RoboSim Physical Modelling specified in Ecore and formatted by the tool `OCLinEcore`. The syntax of the representation used in this appendix is available here.

A summary of the concepts of Ecore can be found here, and a tutorial is available here.

```
import ecore : 'http://www.eclipse.org/emf/2002/Ecore';
import robochart : 'robochart.ecore#/';
import robosim : 'robosim.ecore#/';

package physmod : physmod = 'http://www.robocalc.circus/PhysMod'
{
  class PMPackage extends robochart::BasicPackage
  {
    property mappings : PlatformMapping[*|1] { composes };
    property pmodels : PModel[*|1] { composes };
    property types : robochart::TypeDecl[*|1] { composes };
    property links : LinkDef[*|1] { composes };
    property bodies : BodyDef[*|1] { composes };
    property sensors : SensorDef[*|1] { composes };
    property actuators : ActuatorDef[*|1] { composes };
    property joints : JointDef[*|1] { composes };
    property fragments : Fragment[*|1] { composes };
    property unitdefinitions : UnitDefinition[*|1] { composes };
    property templates : AnnotationTemplate[*] { composes };
    property functions : robochart::Function[*|1] { composes };
    property constants : robochart::VariableList[*|1] { ordered composes };
  }
  class PModel extends robochart::NamedElement
  {
    property parts : Part[*|1] { composes };
    property links : Link[*|1] { composes };
    property constants : robochart::VariableList[*|1] { ordered composes };
  }
```

```
class Pose
{
  property x : robochart::Expression[1] { composes };
  property y : robochart::Expression[1] { composes };
  property z : robochart::Expression[1] { composes };
  property roll : robochart::Expression[1] { composes };
  property pitch : robochart::Expression[1] { composes };
  property yaw : robochart::Expression[1] { composes };
}

/*
* Realisations
*/
abstract class Realisation extends robochart::NamedElement
{
  property pose : Pose[?] { composes };
  property index : Index[?] { composes };
}
abstract class ReferenceRealisation extends Realisation
{
  property def : Definition[1];
}
abstract class LocalRealisation extends Realisation
{
  property def : Definition[1] { composes };
}
abstract class Definition
{

  /*
  * this attribute is optional. it is not necessary inside local definitions
  */
  attribute name : String[?] = '__def__';
}

/*
* LinkA LinkDef defines a link in a library. It can be used be referencing it in a
    p-model.
*/
class LinkDef extends Definition
{
  property inertial : Inertial[?] { composes };
}

/*
* A LinkRealisation is an instance of a link used in a p-model. It can be either a
    reference or local realisation
*/
class Link extends Realisation
{
  property fixed : FixedConnection[*|1] { composes };
  property bodies : Body[*|1] { composes };
  property sensors : Sensor[*|1] { composes };
  property actuators : Actuator[*|1] { composes };
```

```
  property joints : Joint[*|1] { composes };
}
class ReferenceLink extends Link,ReferenceRealisation;
class LocalLink extends Link,LocalRealisation;

/*
* Body
*/
class BodyDef extends Definition
{
  property geometry : Geometry[1] { composes };
  property laser_retro : robochart::Expression[?] { composes };
  property bounce : Bounce[?] { composes };
  property friction : Friction[?] { composes };
  property transparency : robochart::Expression[?] { composes };
  property inertial : Inertial[?] { composes };
}
class Body extends Realisation;
class ReferenceBody extends Body,ReferenceRealisation;
class LocalBody extends Body,LocalRealisation;

/*
* Joint
*/
class JointDef extends Definition,DynamicDevice,AnnotatedElement;
class Joint extends Realisation,robochart::NamedExpression
{
  property flexible : FlexibleConnection[?] { composes };
  property sensors : Sensor[*|1] { composes };
  property actuators : Actuator[*|1] { composes };
}
class ReferenceJoint extends Joint,ReferenceRealisation,AnnotatedElement
{
  property instantiations : Instantiation[*|1] { ordered composes };
}
class LocalJoint extends Joint,LocalRealisation;

/*
* Sensor
*/
class SensorDef extends Definition,DynamicDevice,AnnotatedElement;
class Sensor extends Realisation,robochart::NamedExpression
{
  property relation : robochart::Expression[?] { composes };
}
class ReferenceSensor extends Sensor,ReferenceRealisation,AnnotatedElement
{
  property instantiations : Instantiation[*|1] { ordered composes };
}
class LocalSensor extends Sensor,LocalRealisation;

/*
* Actuator
*/
```

```
class ActuatorDef extends Definition,DynamicDevice,AnnotatedElement;
class Actuator extends Realisation,robochart::Variable
{
  property relation : robochart::Expression[?] { composes };
}
class ReferenceActuator extends Actuator,ReferenceRealisation,AnnotatedElement
{
  property instantiations : Instantiation[*|1] { ordered composes };
}
class LocalActuator extends Actuator,LocalRealisation;

/*
* Other definitions
*/
class Inertial
{
  property mass : robochart::Expression[?] { composes };
  property inertia : Inertia[?] { composes };
  property pose : Pose[?] { composes };
}
class Inertia
{
  property ixx : robochart::Expression[1] { composes };
  property ixy : robochart::Expression[1] { composes };
  property ixz : robochart::Expression[1] { composes };
  property iyy : robochart::Expression[1] { composes };
  property iyz : robochart::Expression[1] { composes };
  property izz : robochart::Expression[1] { composes };
}
abstract class Geometry;
class Box extends Geometry
{
  property x : robochart::Expression[1] { composes };
  property y : robochart::Expression[1] { composes };
  property z : robochart::Expression[1] { composes };
}
class Cylinder extends Geometry
{
  property radius : robochart::Expression[1] { composes };
  property length : robochart::Expression[1] { composes };
}
class Sphere extends Geometry
{
  property radius : robochart::Expression[1] { composes };
}
abstract class DynamicDevice
{
  property inputs : robochart::VariableList[*|1] { composes };
  property outputs : robochart::VariableList[*|1] { composes };
  property locals : robochart::VariableList[*|1] { composes };
  property constants : robochart::VariableList[*|1] { ordered composes };
  property equations : robochart::Expression[*|1] { composes };
}
class FlexibleConnection
```

```
{
  property link : Link[1];
  property part : Part[?];
  property joint : Joint[?];
}
class Mesh extends Geometry
{
  attribute shape : VVMesh[1];
  property scaling : robochart::Expression[1] { composes };
}
class Bounce
{
  property restitution_coefficient : robochart::Expression[?] { composes };
  property threshold : robochart::Expression[?] { composes };
}
class Friction
{
  property translational : Translational[?] { composes };
  property torsional : Torsional[?] { composes };
}
class Torsional
{
  property coefficient : robochart::Expression[?] { composes };
  property surface_radius : robochart::Expression[?] { composes };
}
class Translational
{
  property coefficient1 : robochart::Expression[?] { composes };
  property coefficient2 : robochart::Expression[?] { composes };
  property direction : robochart::Expression[?] { composes };
  property slip1 : robochart::Expression[?] { composes };
  property slip2 : robochart::Expression[?] { composes };
}
class Instantiation
{
  property name : robochart::Variable[?];
  property value : robochart::Expression[?] { composes };
}
class Action
{
  property statements : robochart::Statement[*|1] { composes };
}
class PlatformMapping extends robochart::NamedElement
{
  property outputEvents : OutputEventMapping[*|1] { composes };
  property operations : OperationMapping[*|1] { composes };
  property dmodel : robosim::SimModule[1];
  property pmodel : PModel[1];
  property inputEvents : InputEventMapping[*|1] { composes };
  property variables : VariableMapping[*|1] { composes };
}
class OutputEventMapping extends Mapping
{
  property event : robochart::Event[1];
```

```
  property parameter : EventParameter[?] { composes };
}
class OperationMapping extends Mapping
{
  property operation : robochart::OperationSig[1];
}
class EventParameter extends robochart::Variable;
class InputEventMapping extends Mapping
{
  property event : robochart::Event[1];
  property parameter : EventParameter[?] { composes };
  property predicate : robochart::Expression[*|1] { composes };
}
class VariableMapping extends Mapping
{
  property variable : robochart::Variable[1];
}
class Limit
{
  property lower : robochart::Expression[?] { composes };
  property upper : robochart::Expression[?] { composes };
  property maxEffort : robochart::Expression[?] { composes };
  property maxVelocity : robochart::Expression[?] { composes };
  property stiffness : robochart::Expression[?] { composes };
  property dissipation : robochart::Expression[?] { composes };
}
class Dynamics
{
  property damping : robochart::Expression[?] { composes };
  property friction : robochart::Expression[?] { composes };
  property springReference : robochart::Expression[?] { composes };
  property springStiffness : robochart::Expression[?] { composes };
}
abstract class Mapping
{
  property variables : robochart::VariableList[*|1] { composes };
  property equations : robochart::Expression[*|1] { composes };
  property actions : robochart::Statement[*|1] { ordered composes };
}
class Fragment extends robochart::NamedElement
{
  property links : Link[*|1] { composes };
  property joints : Joint[*|1] { composes };
  property actuators : Actuator[*|1] { composes };
  property sensors : Sensor[*|1] { composes };
}
class Index extends robochart::NamedElement,robochart::NamedExpression
{
  property range : robochart::Expression[?] { composes };
}
class FixedConnection
{
  property link : Link[1];
  property part : Part[?];
```

```
  property sourceLink : Link[?];
}

/*
* Expressions
*/
class TimeVariable extends robochart::Variable;
class DerivativeExp extends robochart::Expression
{
  property function : robochart::Expression[?] { composes };
  property variable : robochart::Variable[?] { composes };
}
class IntegralExp extends robochart::Expression
{
  property function : robochart::Expression[?] { composes };
  property variable : robochart::Variable[?] { composes };
  property lower_limit : robochart::Expression[?] { composes };
  property upper_limit : robochart::Expression[?] { composes };
}
class UnitExp extends robochart::Expression
{
  property exp : robochart::Expression[?] { composes };
  property unit : Unit[?] { composes };
}
class UnitlessExp extends robochart::Expression
{
  property exp : robochart::Expression[?] { composes };
}
datatype VVMesh : 'java.lang.String' { serializable };
abstract class Unit;
class UnitDefinition
{
  attribute name : String[?];
}
class DerivedUnitDefinition extends UnitDefinition
{
  property unit : Unit[?] { composes };
  property relation : robochart::Expression[?] { composes };
}
class UnitReference extends Unit
{
  property ref : UnitDefinition[?];
}
class ProductUnit extends Unit
{
  property left : Unit[1] { composes };
  property right : Unit[1] { composes };
}
class QuotientUnit extends Unit
{
  property left : Unit[1] { composes };
  property right : Unit[1] { composes };
}
class ParenthesizedUnit extends Unit
```

```
{
  property unit : Unit[1] { composes };
}
class ExponentialUnit extends Unit
{
  property unit : Unit[1] { composes };
  attribute exponent : ecore::EInt[1];
}

/*
* Annotations
*/
abstract class AnnotatedElement
{
  property _annotation : Annotation[?] { composes };
}
class AnnotationTemplate extends robochart::NamedElement
{
  property parameters : AnnotationParameter[*] { composes };
  property extensions : AnnotationTemplate[*];
}
class AnnotationParameter extends robochart::NamedElement;
class TemplateParameter extends AnnotationParameter
{
  property template : AnnotationTemplate[?];
}
class AnnotationValueParameter extends AnnotationParameter
{
  property type : robochart::Type[?] { composes };
  property default : robochart::Expression[?] { composes };
}
class AnnotationLinkParameter extends AnnotationParameter
{
  property default : Link[?];
}
class Annotation
{
  property template : AnnotationTemplate[1];
  property instantiations : AnnotationInstantiation[*] { composes };
}
abstract class AnnotationInstantiation;
class AnnotationValueInstantiation extends AnnotationInstantiation
{
  property parameter : AnnotationValueParameter[?];
  property value : robochart::Expression[?] { composes };
}
class AnnotationLinkInstantiation extends AnnotationInstantiation
{
  property parameter : AnnotationLinkParameter[?];
  property link : Link[?];
}
class AnnotationTemplateInstantiation extends AnnotationInstantiation
{
  property parameter : TemplateParameter[?];
```

```
    property instantiations : AnnotationInstantiation[*] { composes };
    property template : AnnotationTemplate[?];
  }
  class Part extends robochart::NamedElement,Realisation
  {
    property instantiations : Instantiation[*|1] { ordered composes };
    property model : PModel[1];
    property flexible : FlexibleConnection[*|1] { composes };
    property fixed : FixedConnection[*|1] { composes };
  }
  class TimeExp extends robochart::Expression;
  class ActuatorType extends robochart::NamedExpression,robochart::NamedElement
  {
    property fields : robochart::Field[*] { composes };
  }
}
```

Credits

LaTeX style based on the The Legrand Orange Book Template by Mathias Legrand and Vel from LaTeXTemplates.com. Licensed under CC BY-NC-SA 3.0.

Icons used in RoboTool and this report have been obtained from www.flaticon.com. Individual credits are given below.


Icon made by Iconnice from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Sarfraz Shoukat from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Freepik from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Dario Ferrando from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Lyolya from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Freepik from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Google from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Freepik from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Freepik from www.flaticon.com is licensed by CC 3.0 BY


Icon made by Freepik from www.flaticon.com is licensed by CC 3.0 BY

Bibliography

[1] ISO/IEC 13568:2002. *Information Technology – Z Formal Specification Notation – Syntax, Type System and Semantics*. International Standard (cited on pages 7, 11).

[2] Ana Cavalcanti, Augusto Sampaio, Alvaro Miyazawa, Pedro Ribeiro, Madiel Conserva Filho, André Didier, Wei Li, and Jon Timmis. "Verified simulation for robotics". In: *Science of Computer Programming* 174 (2019), pages 1–37. ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2019.01.004. URL: http://www.sciencedirect.com/science/article/pii/S0167642318301655.

[3] *Eclipse*. http://eclipse.org/. Accessed: 2018-01-06 (cited on page 37).

[4] J. C. P. Woodcock and J. Davies. *Using Z – Specification, Refinement, and Proof*. Prentice-Hall, 1996 (cited on page 7).

[5] Alvaro Miyazawa, Ana Cavalcanti, Sharar Ahmadi, Mark Post, and Jon Timmis. *RoboSim Physical Modelling Reference Manual*. Technical report. https://robostar.cs.york.ac.uk/publications/techreports/reports/physmod-reference.pdf (cited on page 7).

[6] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. "RoboChart: Modelling and verification of the functional behaviour of robotic applications". In: *Software & Systems Modeling* (Jan. 2019). ISSN: 1619-1374. DOI: 10.1007/s10270-018-00710-z. URL: doi.org/10.1007/s10270-018-00710-z.

[7] Alvaro Miyazawa, Pedro Ribeiro, Kangfen Ye, Ana Cavalcanti, Wei Li, Jim Woodcock, and Jon Timmis. *RoboChart Reference Manual*. Technical report. https://www.cs.york.ac.uk/circus/publications/techreports/reports/robochart-reference.pdf (cited on pages 7, 11).

[8] N. Koenig and H. Andrew. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Volume 3. IEEE, 2004, pages 2149–2154 (cited on page 38).

[9] *Simulink*. www.mathworks.com/products/simulink. The MathWorks,Inc. (cited on page 38).

[10] *Xtext*. https://eclipse.org/Xtext/. Accessed: 2016-05-25 (cited on page 37).

Index of Semantic Rules

In this index you'll find the list of semantic functions in alphabetic order, and page where they are defined. Timed versions of existig semantic rules are indexed by a **timed** item under the entry for the semantic function. Semantic functions exclusive to the timed model are identified by a **timed** annotation in parenthesis after the rule name. Rules whose names are abbreviation (e.g., S) are annotated with the full name in parenthesis.

Index of Calls to Semantic Rules

In this index you'll find the location of call to the semantic rules. For each call of a semantic function, the page number superscripted with the usage index is provided. The index of the call is unique with respect to the semantic function, and also shown superscripted in the call location.