

Each construct will be given in its presentation form, as displayed in the Rodin toolkit, followed by the ASCII form that is used for input to Rodin.

In the following: P, Q and R denote predicates;
 x and y denote single variables;
 z denotes a list of comma-separated variables;
 p denotes a pattern of variables, possibly including \mapsto and parentheses;
 S and T denote set expressions;
 U denotes a set of sets;
 m and n denote integer expressions;
 f and g denote functions;
 r denotes a relation;
 E and F denote expressions;
 E, F is a recursive pattern, ie it matches e_1, e_2 and also $e_1, e_2, e_3 \dots$; similarly for x, y ;

Freeness: The meta-predicate $\neg free(z, E)$ means that none of the variables in z occur *free* in E . This meta-predicate is defined recursively on the structure of E , but that will not be done here explicitly. The base cases are: $\neg free(z, \forall z \cdot P \Rightarrow Q)$, $\neg free(z, \exists z \cdot P \wedge Q)$, $\neg free(z, \{z \cdot P \mid F\})$, $\neg free(z, \lambda z \cdot P[E])$, and $free(z, z)$.

In the following the statement that P *must constrain* z means that the type of z must be at least inferable from P .

In the following, parentheses are used to show syntactic structure; they may of course be omitted when there is no confusion.

1 Predicates

A predicate is a function from some set X to Boolean (bool)

- 1. False: \perp false
- 2. True: \top true

Boolean cannot be used as a type for constants and variables. Instead EventB provides a set `BOOL` defined as an enumeration

$$\text{BOOL} = \{\text{FALSE}, \text{TRUE}\},$$

which can be used for *concrete* representations of *false* and *true*.

There is also a function `bool` that maps predicates into values in `BOOL`: `bool(\perp) = FALSE` and `bool(\top) = TRUE`.

- 1. Conjunction: $P \wedge Q$ $P \ \& \ Q$
Left associative.
- 2. Disjunction: $P \vee Q$ $P \ \text{or} \ Q$
Left associative.
- 3. Implication: $P \Rightarrow Q$ $P \Rightarrow Q$
Non-associative: this means that $P \Rightarrow Q \Rightarrow R$ must be parenthesised or an error will be diagnosed.
- 4. Equivalence: $P \Leftrightarrow Q$ $P \ \Leftrightarrow \ Q$
 $P \Leftrightarrow Q = P \Rightarrow Q \wedge Q \Rightarrow P$
Non-associative: this means that $P \Leftrightarrow Q \Leftrightarrow R$ must be parenthesised or an error will be diagnosed.
- 5. Negation: $\neg P$ not P
- 6. Universal quantification: $(\forall z \cdot P \Rightarrow Q)$ (!z.P => Q)

For all values of z satisfying P, Q (is true)
 The types of z must be inferable from the predicate P .

- 7. Existential quantification: $(\exists z \cdot P \wedge Q)$ (#z.P & Q)
The predicate P must constrain z .
- 8. Equality: $E = F$ $E = F$
- 9. Inequality: $E \neq F$ $E \neq F$

2 Sets

- 1. Singleton set: $\{E\}$ {E}
- 2. Set enumeration: $\{E, F\}$ {E, F}
See note on the pattern E, F at top of summary.
- 3. Empty set: \emptyset { }
- 4. Set comprehension: $\{z \cdot P \mid F\}$ { z . P | F }
General form: the set of all values of F for all values of z that satisfy the predicate P . P must *constrain* the variables in z .
- 5. Set comprehension: $\{F \mid P\}$ { F | P }
Special form: the set of all values of F that satisfy the predicate P . In this case the set of bound variables z are all the free variables in F .
 $\{F \mid P\} = \{z \cdot P \mid F\}$, where z is all the variables in F .
- 6. Set comprehension: $\{x \mid P\}$ { x | P }
A special case of item 5: the set of all values of x that satisfy the predicate P .
 $\{x \mid P\} = \{x \cdot P \mid x\}$
- 7. Union: $S \cup T$ $S \ \cup \ T$

Note: in EventB, relations and functions only ever have one argument, but that argument may be a pair or tuple, hence $f(E \mapsto F)$ $f(E \mapsto F)$ $f(E, F)$ is never valid.

5.2 Actions

Actions are used to change the state of a machine. There may be multiple actions, but they take effect concurrently, that is, in parallel. The semantics of events are defined in terms of *substitutions*. The substitution $[G]P$ defines a predicate obtained by replacing the values of the variables in P according to the action G . General substitutions are not available in the EventB language.

Note on concurrency: any single variable can be modified in at most one action, otherwise the effect of the actions would, in general, be inconsistent.

- 1. *skip*, the null action: $skip$
 $skip$ denotes the empty set of actions for an event.
- 2. Simple assignment action: $x := E$ $x := E$
:= = “*becomes equal to*”: replace free occurrences of x by E .
- 3. Choice from set: $x \in S$ $x \in S$
: \in = “*becomes in*”: arbitrarily choose a value from the set S .
- 4. Choice by predicate: $z :| P$ $z :| P$
: $|$ = “*becomes such that*”: arbitrarily choose values for the variable in z that satisfy the predicate P . Within P , x refers to the value of the variable x before the action and x' refers to the value of the variable after the action.
- 5. Functional override: $f(x) := E$ $f(x) := E$
Substitute the value E for the expression f at point x .
This is a shorthand for $f(x) := E = f := f \Leftarrow \{x \mapsto E\}$.
- 6. Multiple action: $x, y := E, F$ $x, y := E, F$
Concurrent assignment of the values E and F to the variables x and y , respectively. This is equivalent mulitple single actions.

5 Models

- 1. Contexts: contain sets and constants used by other contexts or machines.

CONTEXT	Identifier
EXTENDS	Machine_Identifiers
SETS	Identifiers
CONSTANTS	Identifiers
AXIOMS	Predicates
THEOREMS	Predicates
END	

- 2. Machines: contain events.

MACHINE	Identifier
REFINES	Machine_Identifiers
SEES	Context_Identifiers
VARIABLES	Identifiers
INVARIANT	Predicates
THEOREMS	Predicates
VARIANT	Expression
EVENTS	Events
END	

5.1 Events

Event_name	
REFINES	Event_identifiers
ANY	Identifiers
WHERE	Predicates
WITH	Witnesses
THEN	Actions
END	

There is one distinguished event named *INITIALISATION* used to initialise the variables of a machine, thus establishing the invariant.

¹Version April 21, 2009©1996-2009 Ken Robinson

8. Intersection: $S \cap T$ `S \ / T`

9. Difference: $S \setminus T$ `S \ T`

10. Ordered pair: $E \mapsto F$ `E |-> F`

Ordered pair: $E \mapsto F$ `E |-> F`

$E \mapsto F \neq (E, F)$

In all places where an ordered pair is required, E, F will not be used. E, F must be accepted as an ordered pair, it is always a list.

The following is based on the set of integers, the set of positive (non-zero) natural numbers.

3 Numbers

1. The set of integer numbers: \mathbb{Z} `INT`
2. The set of natural numbers: \mathbb{N} `NAT`
3. The set of positive natural numbers: \mathbb{N}_+ `NAT1`
4. Minimum: $\min(S)$ `min(S)`
 $S \subset \mathbb{Z}$ and finite(S) or S must have a lower bound.
5. Maximum: $\max(S)$ `max(S)`
 $S \subset \mathbb{Z}$ and finite(S) or S must have an upper bound.
11. Cartesian product: $S \times T$ `S * T`
12. Powerset: $\mathbb{P}(S)$ `POW(S)`
 $\mathbb{P}(S) = \{s \mid s \subseteq S\}$
13. Non-empty subsets: $\mathbb{P}_1(S)$ `POW1(S)`
 $\mathbb{P}_1(S) = \mathbb{P}(S) \setminus \{\emptyset\}$
14. Cardinality: $\text{card}(S)$ `card(S)`
Defined only for finite(S).
15. Generalized union: $\text{union}(U)$ `union(U)`
The union of all the elements of U .
 $\bigcup U \cdot U \in \mathbb{P}(\mathbb{P}(S)) \Rightarrow$
 $\bigcup(U) = \{x \mid x \in S \vee \exists s \cdot s \in U \wedge x \in s\}$
where $\neg \text{free}(x, s, U)$
16. Generalized intersection: $\text{inter}(U)$ `inter(U)`
The intersection of all the elements of U .
 $\bigcap U \cdot U \in \mathbb{P}(\mathbb{P}(S)) \Rightarrow$
 $\bigcap(U) = \{x \mid x \in S \wedge \forall s \cdot s \in U \Rightarrow x \in s\}$
where $\neg \text{free}(x, s, U)$
17. Generalized union: $\text{UNION } z, p \mid E$ `UNION z, p | E`
 $\bigcup z \cdot p \mid E$
 F must constrain the variables in z .
 $\forall z \cdot P \Rightarrow E \subseteq T \Rightarrow$
 $\bigcup z \cdot P \mid E = \{x \mid x \in T \wedge \exists z \cdot P \vee x \in E\}$
where $\neg \text{free}(x, z, T), \neg \text{free}(x, P), \neg \text{free}(x, E)$,
 $\neg \text{free}(x, z)$
18. Generalized intersection: $\text{INTER } z, p \mid E$ `INTER z, p | E`
 $\bigcap z \cdot p \mid E$
 F must constrain the variables in z ,
 $\{z \mid P\} \neq \emptyset$,
 $(\forall z \cdot P \Rightarrow E \subseteq T) \Rightarrow$
 $\bigcap z \cdot P \mid E = \{x \mid x \in T \wedge (\forall z \cdot P \Rightarrow x \in E)\}$
where $\neg \text{free}(x, z), \neg \text{free}(x, P), \neg \text{free}(x, E)$,
 $\neg \text{free}(x, E)$.

3.1 Number predicates

1. Greater: $m > n$ `m > n`
2. Less: $m < n$ `m < n`
3. Greater or equal: $m \geq n$ `m >= n`
4. Less or equal: $m \leq n$ `m <= n`
11. Interval: $m..n$ `m .. n`
 $m..n = \{i \mid m \leq i \wedge i \leq n\}$
9. Quotient: m/n `m / n`
10. Remainder: $m \bmod n$ `m mod n`
11. Interval: $m..n$ `m .. n`
 $m..n = \{i \mid m \leq i \wedge i \leq n\}$
1. Greater: $m > n$ `m > n`
2. Less: $m < n$ `m < n`
3. Greater or equal: $m \geq n$ `m >= n`
4. Less or equal: $m \leq n$ `m <= n`

4 Relations

A relation is a set of ordered pairs: a many to many mapping.

1. Relations: $S \mapsto T$ `S <-> T`
2. Domain: $\text{dom}(r)$ `dom(r)`
 $\forall r \cdot r \in S \mapsto T \Rightarrow$
 $\text{dom}(r) = \{x \cdot (\exists y \cdot x \mapsto y \in r)\}$
1. Set membership: $E \in S$ `E : S`
2. Set non-membership: $E \notin S$ `E / : S`
3. Subset: $S \subseteq T$ `S < : T`
4. Not a subset: $S \not\subseteq T$ `S /< : T`
5. Proper subset: $S \subset T$ `S << : T`

2.1 Set predicates

1. Lambda abstraction: $(\lambda p \cdot P \mid E)$ `(%p . P | E)`
 F must constrain the variables in p .
2. Total surjections: $S \mapsto T$ `S -->> T`
 $S \mapsto T = S \mapsto T \cup S \mapsto T$
3. Partial surjections: $S \mapsto T$ `S --> T`
 $S \mapsto T = \{f \cdot f \in S \mapsto T \wedge \text{ran}(f) = T\}$.
4. Total injections: $S \mapsto T$ `S >-> T`
 $S \mapsto T = S \mapsto T \cap S \mapsto T$
5. Partial injections: $S \mapsto T$ `S >-> T`
 $S \mapsto T = \{f \cdot f \in S \mapsto T \wedge f^{-1} \in T \mapsto S\}$.
6. Total surjections: $S \mapsto T$ `S -->> T`
 $S \mapsto T = S \mapsto T \cup S \mapsto T$
7. Bijections: $S \mapsto T$ `S <->> T`
 $S \mapsto T = S \mapsto T \cap S \mapsto T$
8. Lambda abstraction: $(\lambda p \cdot P \mid E)$ `(%p . P | E)`
 F must constrain the variables in p .
9. Function application: $f(E)$ `F(E)`
 $E \mapsto y \in f \Rightarrow E \in \text{dom}(f) \wedge f \in X \mapsto Y$, where $\text{type}(f) = \mathbb{P}(X \times Y)$.

3. Range: $\text{ran}(r)$ `ran(r)`
 $\forall r \cdot r \in S \mapsto T \Rightarrow$
 $\text{ran}(r) = \{y \cdot (\exists x \cdot x \mapsto y \in r)\}$

4. Total relation: $S \leftrightarrow T$ `S <<-> T`
if $r \in S \leftrightarrow T$ then $\text{dom}(r) = S$

5. Surjective relation: $S \twoheadrightarrow T$ `S <->> T`
if $r \in S \twoheadrightarrow T$ then $\text{ran}(r) = T$

6. Total surjective relation: $S \twoheadrightarrow T$ `S <->> T`
if $r \in S \twoheadrightarrow T$ then $\text{dom}(r) = S$ and $\text{ran}(r) = T$

7. Forward composition: $p \circ q$ `p ; q`
 $\forall p, q \cdot p \circ q \in S \mapsto T \wedge q \in T \mapsto U \Rightarrow$
 $p \circ q = \{x \mapsto y \mid (\exists z \cdot x \mapsto z \in p \wedge z \mapsto y \in q)\}$

8. Backward composition: $p \circ q$ `p circ q`
 $d \circ q = q \circ d$

9. Identity: $\text{id}(S)$ `id(S)`
 $\text{id}(S) = \{x \mapsto x \mid x \in S\}$. Note: in Rodin 1.0 id will be polymorphic, and the set will be inferred from the context.

10. Domain restriction: $S \triangleright r$ `S >| r`
 $S \triangleright r = \{x \mapsto y \mid x \mapsto y \in r \wedge x \in S\}$.

11. Domain subtraction: $S \triangleleft r$ `S <| r`
 $S \triangleleft r = \{x \mapsto y \mid x \mapsto y \in r \wedge x \notin S\}$.

12. Range restriction: $r \triangleleft T$ `r |> T`
 $r \triangleleft T = \{x \mapsto y \mid x \mapsto y \in r \wedge y \in T\}$.

13. Range subtraction: $r \triangleleft T$ `r |>> T`
 $r \triangleleft T = \{x \mapsto y \mid y \in r \wedge y \notin T\}$.

14. Inverse: r^{-1} `r ~`
 $r^{-1} = \{y \mapsto x \mid x \mapsto y \in r\}$.

15. Relational image: $r[S]$ `r[S]`
 $r[S] = \{y \mid \exists x \cdot x \in S \wedge x \mapsto y \in r\}$.

16. Overriding: $r_1 \triangleleft r_2$ `r1 <+ r2`
 $r_1 \triangleleft r_2 = r_2 \cup (\text{dom}(r_2) \triangleleft r_1)$.

17. Direct product: $p \otimes q$ `p <> q`
 $p \otimes q = \{x \mapsto (y \mapsto z) \mid x \mapsto y \in p \wedge x \mapsto z \in q\}$.

18. Parallel product: $p \parallel q$ `p || q`
 $p \parallel q = \{x, y, m, n \cdot x \mapsto m \in p \wedge y \mapsto n \in q \mid (x \mapsto y) \mapsto (m \mapsto n)\}$.

19. Projection: $\text{prj}_1(S, T)$ `prj1(S, T)`
 $\text{prj}_1(S, T) = \{x \mapsto y \mid x \mapsto y \in S \times T\}$.

20. Projection: $\text{prj}_2(S, T)$ `prj2(S, T)`
 $\text{prj}_2(S, T) = \{(x \mapsto y) \mid x \mapsto y \in S \times T\}$.
Projections are expected to become polymorphic.

4.1 Iteration and Closure

Iteration and closure are important functions on relations that are not currently part of the kernel EventB language. They can be defined in a Context, but not polymorphically.

Note: iteration and reflexive closure will be implemented in a proposed extension of the mathematical language. The operators will be non-associative.

1. Iteration: r^+ `S <->> T`
 $r \in S \mapsto S \Rightarrow r^+ = \text{id}(S) \vee r^{n+1} = r \circ r^n$.

2. Reflexive Closure: r^* `p circ q`
 $r^* = \bigcup n \cdot (n \in \mathbb{N} \mid r^n)$.
Note: $r^0 \subseteq r^*$.

3. Irreflexive Closure: r^+ `id(S)`
 $r^+ = \bigcup n \cdot (n \in \mathbb{N}_+ \mid r^n)$.
Note: $r^0 \not\subseteq r^+$.

Could be defined as a function $\text{rclosure}(r)$.

Could be defined as a function $\text{rclosure}(r)$.

depending on r .

4.2 Functions

A function is a relation with the restriction that each element of the domain is related to a unique element in the range: a many to one mapping.

1. Partial functions: $S \mapsto T$ `S +> T`
 $S \mapsto T = \{r \cdot r \in S \mapsto T \wedge r \text{ eqId}(T)\}$.

2. Total functions: $S \mapsto T$ `S --> T`
 $S \mapsto T = \{f \cdot f \in S \mapsto T \wedge \text{dom}(f) = S\}$.

3. Partial injections: $S \mapsto T$ `S >+> T`
 $S \mapsto T = \{f \cdot f \in S \mapsto T \wedge f^{-1} \in T \mapsto S\}$.

4. Total injections: $S \mapsto T$ `S >-> T`
 $S \mapsto T = S \mapsto T \cap S \mapsto T$

5. Partial surjections: $S \mapsto T$ `S --> T`
 $S \mapsto T = \{f \cdot f \in S \mapsto T \wedge \text{ran}(f) = T\}$.

6. Total surjections: $S \mapsto T$ `S -->> T`
 $S \mapsto T = S \mapsto T \cup S \mapsto T$

7. Bijections: $S \mapsto T$ `S <->> T`
 $S \mapsto T = S \mapsto T \cap S \mapsto T$

8. Lambda abstraction: $(\lambda p \cdot P \mid E)$ `(%p . P | E)`
 F must constrain the variables in p .

9. Function application: $f(E)$ `F(E)`
 $E \mapsto y \in f \Rightarrow E \in \text{dom}(f) \wedge f \in X \mapsto Y$, where $\text{type}(f) = \mathbb{P}(X \times Y)$.