A Concise Summary of the B mathematical toolkit[1]

Each construct will be presented in its publication form, followed by the boxed `ASCII form` that is used with the BToolkit.

In the following: P, Q and R denote predicates; x and y denote single variables; z denotes a list of variables; S and T denote set expressions; U denotes a set of sets; E and F denote expressions; m and n denote lists of integer expressions; f and g denote functions; r denotes a relation; s and t denote sequence expressions; G, H and I denote a generalized substitutions.

1 Predicates

A predicate is a function from some set X to Boolean. In B an implementation of the type *BOOL* is available from the machine *Bool_TYPE*.

The meta-predicate $z \setminus E$ ("z not free in 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 we won't do that here. The base cases are: $z \setminus (\forall z \cdot P)$, $z \setminus (\exists z \cdot P)$, $z \setminus \{z|P\}$, $z \setminus (\lambda z \cdot (P|E))$, and $\neg(z \setminus z)$.

A predicate P *constrains* the variable x if it contains a predicate of the form: $x \in S$, $x \subseteq S$, $x \subset S$, or $x = E$, where $x \setminus S$, $x \setminus E$.

1. Conjunction: $P \wedge Q$ `P & Q`
2. Disjunction: $P \vee Q$ `P or Q`
3. Implication: $P \Rightarrow Q$ `P => Q`
4. Equivalence: $P \iff Q$ `P <=> Q`.
   $P \iff Q = P \Rightarrow Q \wedge Q \Rightarrow P$
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)
   P must *constrain* the variables in z.
7. Existential quantification:
   $\exists z \cdot (P \wedge Q)$ `#(z).(P & Q)`
   There exists some values of z satisfying P for which Q. P must *constrain* the variables in z.
8. Substitution: $[G]\ P$ `[G] P`
9. Equality: $E = F$ `E = F`
10. Inequality: $E \neq F$ `E /= F`

2 Sets

1. Singleton set: $\{E\}$ `{E}`
2. Set enumeration: $\{E, F\}$ `{E, F}`
   Notice that the pattern E, F can be applied recursively to yield any finite enumeration.
3. Empty set: $\{\}$ `{}`
4. Set comprehension: $\{\ z \mid P\ \}$ `{ z | P }`
   The set of all values of z that satisfy the predicate P. P must *constrain* the variables in z.
5. Union: $S \cup T$ `S \/ T`
6. Intersection: $S \cap T$ `S /\ T`
7. Difference: $S - T$ `S-T`
   $S - T = \{x \mid x \in S \wedge x \notin T\}$
8. Ordered pair: $E \mapsto F$ `E |-> F`.
   $E \mapsto F = E, F$
   Note: in most places $E \mapsto F$ must be used, and E, F will not be accepted, but there are a few contexts, for example set comprehension: $\{x, y|P\}$, where $E \mapsto F$ is not accepted!
9. Cartesian product: $S \times T$ `S * T`
   $S \times T = \{x, y \mid x \in S \wedge y \in T\}$
10. Powerset: $\mathbb{P}(S)$ `POW(S)`
    $\mathbb{P}(S) = \{s \mid s \subseteq S\}$
11. Non-empty subsets: $\mathbb{P}_1(S)$ `POW1(S)`
    $\mathbb{P}_1(S) = \mathbb{P}(S) - \{\{\}\}$
12. Finite subsets: $\mathbb{F}(S)$ `FIN(S)`
13. Finite non-empty subsets: $\mathbb{F}_1(S)$ `FIN1(S)`
14. Cardinality: $\mathsf{card}(S)$ `card(S)`
    Defined only for finite sets
15. Generalized union: $\mathsf{union}(U)$ `union(U)`
    The union of all the elements of U.
    $\forall U \cdot U \in \mathbb{P}(\mathbb{P}(S)) \Rightarrow$
    $\mathsf{union}(U) = \{x \mid x \in S \wedge (\exists s \cdot s \in U \wedge x \in s)\}$
    where $x, s \setminus U$
16. Generalized intersection: $\mathsf{inter}(U)$ `inter(U)`
    The intersection of all the elements of U.
    $\forall U \cdot U \in \mathbb{P}(\mathbb{P}(S)) \Rightarrow$
    $\mathsf{inter}(U) = \{x \mid x \in S \wedge (\forall s \cdot s \in U \Rightarrow x \in s)\}$
    where $x, s \setminus U$
17. Generalized union:
    $\bigcup z \cdot (P \mid E)$ `UNION (z).(P | E)`
    P must *constrain* the variables in z.
    $(\forall z \cdot (P \Rightarrow E \subseteq T)) \Rightarrow$
    $\bigcup z \cdot (P \mid E) = \{x \mid x \in T \wedge (\exists z \cdot (P \wedge x \in E))\}$
    where $z \setminus T, P, E$
18. Generalized intersection:
    $\bigcap z \cdot (P \mid E)$ `INTER (z).(P | E)`
    P must *constrain* the variables in z.

[1] Version March 5, 2003©1996-2003 Ken Robinson

$(\forall z \cdot (P \Rightarrow E \subseteq T)) \Rightarrow$
$\bigcap z \cdot (P \mid E) = \{x \mid x \in T \wedge (\forall z \cdot (P \Rightarrow x \in E))\}$
where $z \setminus T, P, E$

2.1 Set predicates

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`
6. Not a proper subset: $s \not\subset t$ `S /<<: T`

3 Numbers

The following is based on the set of natural numbers (non-negative integers), but the operators extend (directly in most cases) to the set of integers.

1. The set of natural numbers: $\mathbb{N}$ `NAT`
2. The set of positive natural numbers: $\mathbb{N}_1$ `NAT1`
   $\mathbb{N}_1 = \mathbb{N} - \{0\}$
3. Minimum: $\mathsf{min}(S)$ `min(S)`
   Note: $S : \mathbb{F}_1(\mathbb{N})$
4. Maximum: $\mathsf{max}(S)$ `max(S)`
   Note: $S : \mathbb{F}_1(\mathbb{N})$
5. Sum: $m + n$ `m + n`
6. Difference: $m - n$ `m - n`
7. Product: $m \times n$ `m * n`
8. Quotient: m/n `m / n`
9. Remainder: $m \,\mathsf{mod}\, n$ `m mod n`
10. Interval: $m \,..\, n$ `m .. n`
    $m \,..\, n = \{\ i \mid m \le i \le n\ \}$.
11. Set summation:
    $\Sigma\ z \cdot (P \mid E)$ `SIGMA(z).(P | E)`
    $\{z \mid P\} = \{\} \Rightarrow \Sigma\ z \mid (P \mid E) = 0$.
12. Set product: $\Pi z \mid (P \mid E)$ `PI(z).(P | E)`
    Defined only for $\{z \mid P\} \neq \{\}$.

3.1 Number predicates

1. Greater: $m > n$ `m > n`
2. Less: $m < n$ `m < n`
3. Greater or equal: $m \ge n$ `m >= n`
4. Less or equal: $m \le n$ `m <= n`

4 Relations

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

1. Relations: $S \leftrightarrow T$ `S <-> T`
   $S \leftrightarrow T = \mathbb{P}(S \times T)$
2. Domain: $\mathsf{dom}(r)$ `dom(r)`
   $\forall r \cdot r \in S \leftrightarrow T \Rightarrow$
   $\mathsf{dom}(r) = \{x \mid (\exists y \cdot x \mapsto y \in r)\}$
3. Range: $\mathsf{ran}(r)$ `ran(r)`
   $\forall r \cdot r \in S \leftrightarrow T \Rightarrow$
   $\mathsf{ran}(r) = \{y \mid (\exists x \cdot x \mapsto y \in r)\}$
4. Forward composition: $p\,;q$ `p ; q`
   $\forall p, q \cdot p \in S \leftrightarrow T \wedge q \in T \leftrightarrow U \Rightarrow$
   $p\,;q = \{x, y \mid (\exists z \cdot x \mapsto z \in p \wedge z \mapsto y \in q)\}$
5. Backward composition: $p \circ q$ `p circ q`
   $p \circ q = q\,;p$
6. Identity: $\mathsf{id}(S)$ `id(S)`
   $\mathsf{id}(S) = \{x, y \mid x \in S \wedge y \in S \wedge x = y\}$.
7. Domain restriction: $S \lhd r$ `S <| r`
   $S \lhd r = \{x, y \mid x \mapsto y \in r \wedge x \in S\}$.
8. Domain substraction: $S \unlhd r$ `S <<| r`
   $S \unlhd r = \{x, y \mid x \mapsto y \in r \wedge x \notin S\}$.
9. Range restriction: $r \rhd T$ `r |> T`
   $r \rhd T = \{x, y \mid x \mapsto y \in r \wedge y \in T\}$.
10. Range subtraction: $r \unrhd T$ `r |>> T`
    $r \unrhd T = \{x, y \mid x \mapsto y \in r \wedge y \notin T\}$.
11. Inverse: r^{-1} `r~`
    $r^{-1} = \{y, x \mid x \mapsto y \in r\}$.
12. Relational image: $r[S]$ `r[S]`
    $r[S] = \{y \mid \exists x \cdot x \in S \wedge x \mapsto y \in r\}$.
13. Right overriding: $r_1 \Leftarrow r_2$ `r1 <+ r2`
    $r_1 \Leftarrow r_2 = r_2 \cup (\mathsf{dom}(r_2) \unlhd r_1)$.
14. Left overriding: $r_1 \Rightarrow r_2$ `r1 +> r2`
    $r_1 \Rightarrow r_2 = r_1 \cup (\mathsf{dom}(r_1) \unlhd r_2)$.
15. Direct product: $p \otimes q$ `p >< q`
    $p \otimes q = \{x, (y, z) \mid x \mapsto y \in p \wedge x \mapsto z \in q\}$.
16. Parallel product: $p \parallel q$ `p || q`
    $p \parallel q = \{(x, y), (m, n) \mid x \mapsto m \in p \wedge y \mapsto n \in q\}$.
17. Iteration: r^n `iterate(r,n)`
    $r \in S \leftrightarrow S \Rightarrow r^0 = \mathsf{id}(S) \wedge r^{n+1} = r\,;r^n$.
18. Closure: r^* `closure(r)`
    $r^* = \bigcup n \cdot (n \in \mathbb{N} \mid r^n)$.
19. Projection: $\mathsf{prj1}(S, T)$ `prj1(S,T)`
    $\mathsf{prj1}(S, T) =$
    $\{(x, y), z \mid x, y \in S \times T \wedge z = x\}$.
20. Projection: $\mathsf{prj2}(S, T)$ `prj2(S,T)`
    $\mathsf{prj2}(S, T) =$
    $\{(x, y), z \mid x, y \in S \times T \wedge z = y\}$.

4.1 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: $S \mathrel{+\!\!\!\!\!\rightarrow} T$ `S +-> T`
   $S \mathrel{+\!\!\!\!\!\rightarrow} T = \{ r \mid r \in S \leftrightarrow T \wedge r^{-1} \,;\, r \subseteq \mathsf{id}(T) \}$.

2. Total functions: $S \longrightarrow T$ `S --> T`
   $S \longrightarrow T = \{ f \mid f \in S \mathrel{+\!\!\!\!\!\rightarrow} T \wedge \mathsf{dom}(f) = S \}$.

3. Partial injections: $S \mathrel{\rightarrowtail\!\!\!\!\!\!+} T$ `S >+> T`
   $S \mathrel{\rightarrowtail\!\!\!\!\!\!+} T = \{ f \mid f \in S \mathrel{+\!\!\!\!\!\rightarrow} T \wedge f^{-1} \in T \mathrel{+\!\!\!\!\!\rightarrow} S \}$.
   One-to-one relations.

4. Total injections: $S \rightarrowtail T$ `S >-> T`
   $S \rightarrowtail T = S \mathrel{\rightarrowtail\!\!\!\!\!\!+} T \cap S \longrightarrow T$.

5. Partial surjections: $S \mathrel{+\!\!\!\!\!\twoheadrightarrow} T$ `S +->> T`
   $S \mathrel{+\!\!\!\!\!\twoheadrightarrow} T = \{ f \mid f \in S \mathrel{+\!\!\!\!\!\rightarrow} T \wedge \mathsf{ran}(f) = T \}$.
   Onto relations.

6. Total surjections: $S \twoheadrightarrow T$ `S -->> T`
   $S \twoheadrightarrow T = S \mathrel{+\!\!\!\!\!\twoheadrightarrow} T \cap S \longrightarrow T$.

7. Bijections: $S \mathrel{\rightarrowtail\!\!\!\!\!\twoheadrightarrow} T$ `S >->> T`
   $S \mathrel{\rightarrowtail\!\!\!\!\!\twoheadrightarrow} T = S \rightarrowtail T \cap S \twoheadrightarrow T$.
   One-to-one and onto relations.

8. Lambda abstraction:
   $\lambda z \cdot (P \mid E)$ `%z.(P|E)`
   P must *constrain* the variables in z.
   $\lambda z \cdot (P \mid E) = \{ z, y \mid z \in \{ z \mid P \} \wedge y = E \}$, where $y \setminus P$ and $y \setminus E$.

9. Function application: $f(E)$ `f(E)`
   $E \mapsto y \in f \Rightarrow f(E) = y$.

4.2 Sequences

Sequences are ordered aggregations, and can be modelled by functions whose domains are finite, coherent domains $1 \,..\, n$.

1. The empty sequence: $[\,]$ `<>`
   $[\,] = \{\}$.
   Note: $[\,]$ is used for all sequences except the empty ASCII sequence!

2. The set of finite sequences: $\mathsf{seq}(S)$ `seq S`
   $\mathsf{seq}(S) = \{ f \mid f \in \mathbb{N}_1 \mathrel{+\!\!\!\!\!\rightarrow} S \wedge \exists\, n \cdot n \in \mathbb{N} \wedge \mathsf{dom}(f) = 1 \,..\, n \}$.

3. The set of finite non-empty sequences:
   $\mathsf{seq}_1(S)$ `seq1(S)`
   $\mathsf{seq}_1(S) = \mathsf{seq}(S) - \{[\,]\}$.

4. The set of injective sequences:
   $\mathsf{iseq}(S)$ `iseq(S)`
   $\mathsf{iseq}(S) = \mathsf{seq}(S) \cap (\mathbb{N}_1 \mathrel{\rightarrowtail\!\!\!\!\!\!+} S)$.

5. Permutations: $\mathsf{perm}(S)$ `perm(S)`
   $\mathsf{perm}(S) = \mathsf{iseq}(S) \cap (\mathbb{N}_1 \twoheadrightarrow S)$.
   The set of bijective sequences.

6. Sequence concatenation: $s \frown t$ `s^t`
   $s \frown t$ is the sequence formed by appending the sequence t to the sequence s.

7. Prepend element: $E \rightarrow s$ `E -> s`
   $E \rightarrow s = [E] \frown s$.

8. Append element: $s \leftarrow E$ `s <- E`
   $s \leftarrow E = s \frown [E]$.

9. Singleton sequence: $[E]$ `[E]`
   $[E] = \{ 1 \mapsto E \}$.

10. Sequence construction: $[E, F]$ `[E,F]`
    $[E, F] = [E] \leftarrow F$.

11. Size: $\mathsf{size}(s)$ `size(s)`
    $\mathsf{size}(s) = \mathsf{card}(s)$.

12. Reverse: $\mathsf{rev}(s)$ `rev(s)`
    $\forall\, i \cdot i \in \mathsf{dom}(s) \Rightarrow$
    $\quad \mathsf{rev}(s)(i) = s(\mathsf{size}(s) + 1 - i)$.

13. Take: $s \uparrow n$ `s /|\n`
    $s \uparrow n = 1 \,..\, n \lhd s$.

14. Drop: $s \downarrow n$ `s \|/ n`
    $s \downarrow n = (\lambda m \cdot (m \in \mathbb{N} \mid m + n)) \,;\, (1 \,..\, n \mathrel{\lhd\!\!\!\!\!-} s)$.
    $(s \downarrow n)(i) = s(i + n)$

15. First element: $\mathsf{first}(s)$ `first(s)`
    $\mathit{first}(s) = s(1)$
    Defined only for non-empty sequence.

16. Last element: $\mathsf{last}(s)$ `last(s)`
    $\mathsf{last}(s) = s(\mathit{size}(s))$
    Defined only for non-empty sequence.

17. Tail: $\mathsf{tail}(s)$ `tail(s)`
    $\mathit{tail}(s) = s \downarrow 1$
    Defined only for non-empty sequence.
    $\mathsf{first}(s) \rightarrow \mathsf{tail}(s) = s$.

18. Front: $\mathsf{front}(s)$ `front(s)`
    $\mathsf{front}(s) = s \uparrow (\mathit{size}(s) - 1)$
    Defined only for non-empty sequence.
    $\mathsf{front}(s) \leftarrow \mathsf{last}(s) = s$.

19. Generalized concatenation:
    $\mathsf{conc}(ss)$ `conc(ss)`
    Defined on sequences of sequences.
    $\mathsf{conc}([\,]) = [\,]$
    $\mathsf{conc}(s \leftarrow E) = \mathsf{conc}(s) \frown E$.

20. Strings: "..." `"..."`
    Sequences of characters are delimited by quotes.

5 Substitutions

The state of a machine can be changed by substituting values for the variables in the state. The following substitutions formalize a number of alternative ways of achieving this.

1. Substitution: $[G]P$ `[G]P`
   $[G]P$ is a predicate obtained by replacing the values of the variables in P according to the substitution G.

2. The null substitution: *skip* `skip`
   $[\mathit{skip}]R = R$.

3. Simple substitution: $x := E$ `x := E`
Replace free occurrences of x by E.

4. Boolean substitution: $x{:=}\mathsf{bool}(P)$ `x := bool(P)`
Substitute the Boolean values *TRUE* and *FALSE* according to the truth of P.

5. Choice from set: $x :\in S$ `x :: S`
Arbitrarily choose a value from the set S.

6. Choice by predicate: $x : P$ `x : P`
Arbitrarily choose a value that satisfies the predicate P. P must *constrain* the variable x.

7. Functional override: $f(x) := E$ `f(x) := E`
Substitute the value E for the expression f at point x.
$f(x) := E \;=\; f := f \lessdot \{x \mapsto E\}$.

8. Multiple substitution:
$x, y := E, F$ `x,y := E,F`
Concurrent substitution of the values E and F for the free occurrences of x and y, respectively.

9. Parallel substitution: $G \parallel H$ `G || H`
Apply the substitutions G and H concurrently. Parallel substitution is not given a general definition; it is eliminated by rewriting rules. Notice $[x := E]R \parallel [y := F]R = [x, y := E, F]R$.

10. Sequential substitution: $G\,;H$ `G ; H`
Apply the substitution G and then H.
$[G\,;H]R = [G]([H]R)$.

11. Precondition: $P \mid G$ `P | G`
Substitution G is subject to a precondition, P.
$[P \mid G]R = P \wedge [G]R$.

12. Guarding: $P \Longrightarrow G$ `P ==> G`
Substitution G applies only if state satisfies the guard P.
$[P \Longrightarrow G]R \;=\; P \Rightarrow [G]R$.

13. Alternatives: $G \,[\!]\, H$ `G [] H`
Either G or H.
$[G \,[\!]\, H]R = [G]R \wedge [H]R$.

14. Unbounded choice: $@\,z \cdot G$ `@z . G`
Choose any values for z. $[@\,z \cdot G]R = \forall\, z \cdot [G]R$.

5.1 Alternative syntax

1. Grouping: **BEGIN** G **END**

2. **PRE** P **THEN** G **END**
$= P \mid G$

3. **IF** P **THEN** G **ELSE** H **END**
$= (P \Longrightarrow G) \,[\!]\, (\neg P \Longrightarrow H)$

4. **IF** P **THEN** G **END**
= **IF** P **THEN** G **ELSE** *skip* **END**

5. **IF** P_1 **THEN** G_1 **ELSIF** P_2 **THEN** G_2 . . . **ELSE** G_n **END**

6. **IF** P_1 **THEN** G_1 **ELSIF** P_2 **THEN** G_2 . . . **ELSIF** P_n **THEN** G_n **END**

7. **CHOICE** G **OR** H **END**
$= G \,[\!]\, H$

8. **SELECT** P **THEN** G **WHEN** . . . **WHEN** Q **THEN** H **ELSE** I **END**
$= P \Longrightarrow G \,[\!]\, \ldots \,[\!]\, Q \Longrightarrow H \,[\!]\, \neg P \wedge \ldots \wedge \neg Q \Longrightarrow I$

9. **SELECT** P **THEN** G **WHEN** . . . **WHEN** Q **THEN** H **END**
$= P \Longrightarrow G \,[\!]\, \ldots \,[\!]\, Q \Longrightarrow H$

10. **CASE** E **OF EITHER** m **THEN** G **OR** n **THEN** H . . . **ELSE** I **END**
$= E \in \{m\} \Longrightarrow G \,[\!]\, E \in \{n\} \Longrightarrow H \ldots E \notin \{m, n, \ldots\} \Longrightarrow I$

11. **CASE** E **OF EITHER** m **THEN** G **OR** n **THEN** H . . . **END**
default case *skip*

12. **VAR** z **IN** G **END**
$= @\,z \cdot G$

13. **ANY** z **WHERE** P **THEN** G **END**
$= @\,z \cdot P \Longrightarrow G$

14. **LET** x **BE** x = E **IN** G **END**
$= @\,x \cdot x = E \Longrightarrow G$, where $x \setminus E$

5.2 While loop substitution

WHILE P **DO** G **VARIANT** E **INVARIANT** Q **END**

The while-loop substitution is allowed only in implementation machines. The definition of the substitution [**WHILE** P **DO** G **VARIANT** E **INVARIANT** Q **END**]R involves a least fixed point and is not normally used. Instead, an approximation to the substitution is used.

Given some predicate R:

$$
\begin{array}{lcl}
Q \wedge P & \Rightarrow & [G]\ Q \\
Q \wedge P & \Rightarrow & E \in \mathbb{N} \\
Q \wedge P & \Rightarrow & [n := E][G](E < n) \\
\neg P \wedge Q & \Rightarrow & R
\end{array}
$$

$\Rightarrow$

$Q \;\Rightarrow\;$ [**WHILE** P **DO** G **VARIANT** E **INVARIANT** Q **END**] R

where n is a *new* variable satisfying $n \setminus E$ and $n \setminus G$.