NonExecutableSpecs

March 1, 2024

0.1 Non-Executable Specifications

Examples from "Specifications are not (necessarily) executabe" by Hayes and Jones [19] Notebook acompanying a submitted paper.

0.2 Section 2.1

Section 2.1 of [19] discusses the use of known partial functions and the issue of dealing with preconditions. The first example is a function update(f, d, a) which takes a file f represented as a sequence of lines and applies a set of deletions d and set of insertions a. Listing 1.1 below contains a B translation of the update function on line 6, along with example uses of the function within the properties and a B operation.

[4]:
```
::load
MACHINE UpdateFunction
// B encoding of the update function specification by Hayes   Jones
DEFINITIONS
  Line == STRING;
  Lines == seq(Line);
  update(f,d,a) == a(0) ^ conc( n.(n dom(f)| IF n d THEN [] ELSE [f(n)] END ^ a(n)));
CONSTANTS file, del, add, newfile1, newfile2
PROPERTIES
  file   Lines   del    (NATURAL1)   add      Lines
  file = [ "Line1", "Line2", "Line3"]
  del = {2,3}
  add = {0  [">ins0"] }   [ [], [">ins2a",">ins2b"], []]
  newfile1 = update(file,del,add)   newfile2 = update(file, ,add)
OPERATIONS
  f <-- UpdateFile(d,a) = PRE d   dom(file)   a  ({0} dom(file))→ Lines THEN
    f := update(file,d,a)
  END
END
```

[4]: Loaded machine: UpdateFunction

[5]:
```
:init
```

```
[5]: Machine constants were not set up yet. Automatically set up constants using
arbitrary transition: SETUP_CONSTANTS()
Executed operation: INITIALISATION()
```

Let us apply the update function to file consisting of two lines and deleting the first line:

```
[7]: update(["line1","line2"],{1},{0|->[],1|->[],2|->[]})
```

[7]: $\{(1 \mapsto \text{"line2"})\}$

We can insert a lines as follows:

```
[8]: update(["line1","line2"],{},{0|->["prelude"],1|->[" l1a", "␣
  ↪l1b"],2|->["postlude"]})
```

[8]: $\{(1 \mapsto \text{"prelude"}), (2 \mapsto \text{"line1"}), (3 \mapsto \text{" l1a"}), (4 \mapsto \text{" l1b"}), (5 \mapsto \text{"line2"}), (6 \mapsto \text{"postlude"})\}$

The following, however, is not well-defined. We cannot leave the additions a empty, as update accesses a(0) and a(n)

```
[11]: update(file,{},{})
```

```
Error from ProB: UNKNOWN
```

0.3 Section 2.2. Specifying by Inverse

Section 2.2 of [19] discusses specifying concepts indirectly by providing an inverse function. This is often the most natural way of defining a concept. The first example is defining the (largest) integer square root r of an integer n as follows: $r^2 \leq n < (r+1)^2$ We need to slightly rewrite the predicate for B, as we cannot chain the comparison operators:

```
[12]: r 2 n   n<(r+1) 2
```

[12]: *TRUE*

Solution: * $r = 0$ * $n = 0$

As you can see, ProB found a simple solution. We can compute specific integer square roots by specifying n:

```
[13]: n = 101   r 2 n   n<(r+1) 2
```

[13]: *TRUE*

Solution: * $r = 10$ * $n = 101$

We can also compute the integer square root for a variety of values:

```
[14]: {n,r•n:1..100   r 2 n   n<(r+1) 2| r}
```

[14]: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

[15]:
```
{n,r|n:80..100  r 2 n  n<(r+1) 2}
```

[15]:

$\{(80 \mapsto 8), (81 \mapsto 9), (82 \mapsto 9), (83 \mapsto 9), (84 \mapsto 9), (85 \mapsto 9), (86 \mapsto 9), (87 \mapsto 9), (88 \mapsto 9), (89 \mapsto 9), (90 \mapsto 9), (91 \mapsto 9), (92 \mapsto 9), (93 \mapsto 9), (94 \mapsto 9), (95 \mapsto 9), (96 \mapsto 9), (97 \mapsto 9), (98 \mapsto 9), (99 \mapsto 9), (100 \mapsto 10)\}$

[16]:
```
 isqrt = {n,r|n:80..100  r 2 n  n<(r+1) 2} &
 isqrt(n) = r &
 isqrt(n+10) = r
```

[16]:

TRUE

Solution: * $r = 9$ * $isqrt = \{(80 \mapsto 8), (81 \mapsto 9), (82 \mapsto 9), (83 \mapsto 9), (84 \mapsto 9), (85 \mapsto 9), (86 \mapsto 9), (87 \mapsto 9), (88 \mapsto 9), (89 \mapsto 9), (90 \mapsto 9), (91 \mapsto 9), (92 \mapsto 9), (93 \mapsto 9), (94 \mapsto 9), (95 \mapsto 9), (96 \mapsto 9), (97 \mapsto 9), (98 \mapsto 9), (99 \mapsto 9), (100 \mapsto 10)\}$ * $n = 81$

0.4 Section 2.3 Combining Clauses in a Specification

Section 2.3 of [19] is concerned with specifying by combining properties, e.g., via the logical conjunction. The first example is the specification of a sorting algorithm, which is a combination of specifying that the result must a) be sorted and b) be a permutation of the input. The Listing 1.3 below contains a faithful translation of the example from [19].

[21]:
```
::load
MACHINE PermutationSort_v2
 // example from HayesJones for sorting sequence without duplicates
// v2 using B's perm operator
DEFINITIONS
 is_ordered(s) ==  (i,j).(i dom(s)  j dom(s)  i<j  s(i) < s(j));
 is_permutation(s1,s2) == s2:perm(ran(s1))
CONSTANTS in,out
PROPERTIES
  in = [10,5,3,4,1,20,11,33,0,6,88,100,2,7,19,13]
  is_ordered(out)  is_permutation(in,out)
END
```

[21]: Loaded machine: PermutationSort_v2

[ ]:
```
:init
```

[24]:
```
out
```

[24]:

$\{(1 \mapsto 0), (2 \mapsto 1), (3 \mapsto 2), (4 \mapsto 3), (5 \mapsto 4), (6 \mapsto 5), (7 \mapsto 6), (8 \mapsto 7), (9 \mapsto 10), (10 \mapsto 11), (11 \mapsto 13), (12 \mapsto 19), (13 \mapsto 20), (14 \mapsto 33), (15 \mapsto 88), (16 \mapsto 100)\}$

[29]:
```
is_ordered(res)  is_permutation([3,1000,20,2**50,16],res)
```

[29]:

TRUE

Solution: * $res = \{(1 \mapsto 3), (2 \mapsto 16), (3 \mapsto 20), (4 \mapsto 1000), (5 \mapsto 1125899906842624)\}$

Below is a lambda abstraction defining unsorted input sequences that can be used for benchmarking:

[32]:
```
n=50 & in1 = %i.(i:1..n| (i mod 2)*(n+1)+i) &
is_ordered(res)   is_permutation(in1,res)
```

[32]: *TRUE*

Solution: * $res = \{(1 \mapsto 2), (2 \mapsto 4), (3 \mapsto 6), (4 \mapsto 8), (5 \mapsto 10), (6 \mapsto 12), (7 \mapsto 14), (8 \mapsto 16), (9 \mapsto 18), (10 \mapsto 20), (11 \mapsto 22), (12 \mapsto 24), (13 \mapsto 26), (14 \mapsto 28), (15 \mapsto 30), (16 \mapsto 32), (17 \mapsto 34), (18 \mapsto 36), (19 \mapsto 38), (20 \mapsto 40), (21 \mapsto 42), (22 \mapsto 44), (23 \mapsto 46), (24 \mapsto 48), (25 \mapsto 50), (26 \mapsto 52), (27 \mapsto 54), (28 \mapsto 56), (29 \mapsto 58), (30 \mapsto 60), (31 \mapsto 62), (32 \mapsto 64), (33 \mapsto 66), (34 \mapsto 68), (35 \mapsto 70), (36 \mapsto 72), (37 \mapsto 74), (38 \mapsto 76), (39 \mapsto 78), (40 \mapsto 80), (41 \mapsto 82), (42 \mapsto 84), (43 \mapsto 86), (44 \mapsto 88), (45 \mapsto 90), (46 \mapsto 92), (47 \mapsto 94), (48 \mapsto 96), (49 \mapsto 98), (50 \mapsto 100)\}$ * $in1 = \{(1 \mapsto 52), (2 \mapsto 2), (3 \mapsto 54), (4 \mapsto 4), (5 \mapsto 56), (6 \mapsto 6), (7 \mapsto 58), (8 \mapsto 8), (9 \mapsto 60), (10 \mapsto 10), (11 \mapsto 62), (12 \mapsto 12), (13 \mapsto 64), (14 \mapsto 14), (15 \mapsto 66), (16 \mapsto 16), (17 \mapsto 68), (18 \mapsto 18), (19 \mapsto 70), (20 \mapsto 20), (21 \mapsto 72), (22 \mapsto 22), (23 \mapsto 74), (24 \mapsto 24), (25 \mapsto 76), (26 \mapsto 26), (27 \mapsto 78), (28 \mapsto 28), (29 \mapsto 80), (30 \mapsto 30), (31 \mapsto 82), (32 \mapsto 32), (33 \mapsto 84), (34 \mapsto 34), (35 \mapsto 86), (36 \mapsto 36), (37 \mapsto 88), (38 \mapsto 38), (39 \mapsto 90), (40 \mapsto 40), (41 \mapsto 92), (42 \mapsto 42), (43 \mapsto 94), (44 \mapsto 44), (45 \mapsto 96), (46 \mapsto 46), (47 \mapsto 98), (48 \mapsto 48), (49 \mapsto 100), (50 \mapsto 50)\}$ * $n = 50$

0.5 Section 2.4 Negation in Specifications

Section 2.4 of [19] deals with specification by negation, which is an extremely interesting topic. While the conjunction seen in permutation sort can be dealt with by Prolog, negation is a more tricky issue. Indeed, Prolog's negation-as- failure [7] cannot be used to generate solutions, only prune them. Constraint logic programming, however, can provide a constructive version of negation [36, 11] which is also implemented in ProB.

0.5.1 GCD (Greatest Common Divisor)

Example Listing 1.11 contains a trans- lation of the greatest common divisor (GCD) example from [19]. We have to provide a definition of divides, as it is not built-in in B. In the properties section we use our gcd definition to compute the GCD for two examples and in the assertions we check that the results are correct.

[17]:
```
::load
MACHINE GCD
 // Example from Section 2.4 of "Specifications are not (necessarily) executable"
DEFINITIONS
 divides(d,i) == (i mod d = 0) & d>0 & d <= i;
 is_cd(d,i,j) == divides(d,i) & divides(d,j);
 gcd(d,i,j) == is_cd(d,i,j) & not(#e.(e:NATURAL1 & is_cd(e,i,j) & e>d))
CONSTANTS g1, g2
PROPERTIES
  gcd(g1,12,8) &
  gcd(g2,100,60)
```

```
ASSERTIONS
  g1=4; g2=20
END
```

[17]: Loaded machine: GCD

[18]:
```
:init
```

[18]: Machine constants were not set up yet. Automatically set up constants using arbitrary transition: SETUP_CONSTANTS()
Executed operation: INITIALISATION()

[19]:
```
gcd(x,300,77)
```

[19]: *TRUE*

Solution: * $x = 1$

[20]:
```
gcd(x,155,70)
```

[20]: *TRUE*

Solution: * $x = 5$

[ ]:
```
```