A More General Specification Language

Application of Axiomatic Language to CP

Walter W. Wilson
Lockheed Martin

Progress Toward the Holy Grail
August 28, 2017

http://axiomaticlanguage.org/

Language Goals

1. pure specification – what, not how
2. minimal, but extensible
3. metalanguage – able to imitate other languages

Specification by Enumeration

Idea: Program external behavior defined by infinite set of symbolic expressions that enumerate inputs and corresponding outputs.

Program that reads input file and writes output file:

```
(Program <input> <output>)
```

Enumeration of program to sort lines of a text file:

```
  (Program () ())            — empty input file
     …
(Program ("dog" "pig" "cat")   — 3-line input
         ("cat" "dog" "pig"))  — sorted output
     …
```

Recipe

- minimal, pure, definite Prolog with Lisp syntax
- higher-order generalization [HiLog]
- string variables

The Core Language (1)

Axioms generate valid expressions.

expression:

- **atom**, – `` `abc ``, `` `+ ``
- **expression variable**, – `%x`, `%1`
- **sequence** of >=0 expressions and **string variables**
  - `` (`M () $1 %) ``

The Core Language (1)

Axioms generate valid expressions.

expression:

atom, – `` `abc ``, `` `+ ``

expression variable, – `%x`, `%1`

sequence of >=0 expressions and **string variables**

– `` (`M () $1 %) ``

axiom: **conclusion** expr and >= 0 **condition** exprs

<*conclu*> < <*cond1*>, …, <*condn*>.

<*conclu*>. ! unconditional axiom

The Core Language (2)

axiom instance: substitute values for variables

```
  (`A %x $1)< (`B %x), (`C $1).
→ (`A `x `u ())< (`B `x), (`C `u ()).
```

The Core Language (2)

axiom instance: substitute values for variables

```
  (`A %x $1)< (`B %x), (`C $1).
→ (`A `x `u ())< (`B `x), (`C `u ()).
```

valid expressions: If all conditions of an axiom instance are valid expressions, the conclusion is valid.

```
  (`a `b).
  ((%) $ $)< (% $).

 → (`a `b),
   ((`a) `b `b),
   (((`a)) `b `b `b `b),
         …
```

Syntax Extensions

single char in single quotes:

```
'A' = (`char (`0 `1 `0 `0  `0 `0 `0 `1))
```

char string in single quotes within sequence:

```
(… 'abc' …) = (… 'a' 'b' 'c' …)
```

char string in double quotes:

```
"abc" = ('abc') = ('a' 'b' 'c')
```

symbol not starting with special char:

```
abc = (` "abc")
```

Example – Natural Numbers

Set of natural numbers:

```
(num (`z)).
(num (`s $))< (num ($)).
   → (num (`s `s `z))
```

Addition of natural numbers:

```
(plus %n (`z) %n)< (num %n).
(plus %1 (`s $2) (`s $3))<
  (plus %1 ($2) ($3)).
  → (plus (`s `z) (`s `z) (`s `s `z))
```

SEND + MORE = MONEY

```
! top-level axiom for generating solution as a valid expr

 (solution: $eqn)<
   (== ($eqn) (%S %E %N %D + %M %O %R %E = %M %O %N %E %Y)),
   (different- digit (%S %E %N %D %M %O %R %Y)),
     ! -- distinct 1-digit symbols
   (/= %S 0), (/= %M 0),
   (equation ($eqn)).        ! equation must be satisfied
     ! -- grammar merges adjacent digits to form number

→  (solution: 9 5 6 7 + 1 0 8 5 = 1 0 6 5 2)
```

See www.axiomaticlanguage.org/PTHG/SEND_MORE.txt for more details and …/PTHG/util/*.txt for utility predicates.

Sudoku Program

input file:

```
_ 1 _  _ 8 _  _ _ _
6 5 _  7 _ _  4 _ _
_ _ _  _ 2 4  _ 9 _
_ 9 _  _ _ 7  6 _ _
  ...
```

output file →

See …/PTHG/Sudoku.txt

```
           Sudoku Puzzle

Input:
  _  1  _ | _  8  _ | _  _  _
  6  5  _ | 7  _  _ | 4  _  _
  _  _  _ | _  2  4 | _  9  _
 ---------+---------+---------
  _  9  _ | _  _  7 | 6  _  _
              ...

Solution:
  2  1  4 | 9  8  6 | 5  7  3
  6  5  9 | 7  3  1 | 4  2  8
  8  7  3 | 5  2  4 | 1  9  6
 ---------+---------+---------
  4  9  8 | 2  1  7 | 6  3  5
              ...
```

Blend Program

Find lowest cost blend of 2 feed ingredients with sufficient nutrients.

```
! x1,x2,xf,cost: - optimal relative ingredient kg quantities and cost

 (x1,x2,xf,cost: %x1 %x2 %xf %cost)<  ! relative int ingredients & N-kg cost
   (== %N 100),  ! total quantity being considered -> resolution of solution
      ! -- could choose higher-resolution solution here, say, parts per 1000
   (iota_d %N %0-N),   ! decimal numbers from 0..N (APL fn name)
   (cartesian %0-N %0-N %0-N %args),  ! % ranges of the 3 ingredients
      ! -- This generalized cartesian product forms 3-element tuples
      !    of all combinations of integers from 0..N.
   (constraints    ! -- arg vars _0, _1, _2 represent ingredients filler,1,2
     ((_0 + _1 + _2 = %N)               ! 3 natural number vars sum to N
      (100 * _1 + 200 * _2 >= 90 * %N)    ! A nutrient grams required
      ( 80 * _1 + 150 * _2 >= 50 * %N)    ! B
      ( 40 * _1 +  20 * _2 >= 20 * %N)    ! C
      ( 10 * _1            >=  2 * %N)    ! D
     )   ! each constraint formula must evaluate to true
     %args %selected_args),  ! %selected args are tuples that satisfy cnstrs
   (minimize (40 * _1 + 60 * _2) %selected_args %cost (%xf %x1 %x2)).
```

See …/PTHG/blend.txt

Conclusion

- Specifications – software engineering benefit
  - Smaller, more readable, more reusable, more correct
  - Good for CP
- Minimal & pure – well-suited to proof
  - Equivalence of specification and program
  - Prove assertions to validate specification
- Implementation grand challenge
  - Transformation of specifcations to programs
  - A harder Holy Grail