Axiomatic Language

http://www.axiomaticlanguage.org/

Walter W. Wilson
Lockheed Martin

Emerging Languages Camp
Strange Loop 2013
September 18, 2013

Language Goals

- Pure specification – what, not how
- Minimal, but extensible – as small as possible
- Metalanguage – able to imitate other languages
- Beauty!

Specification by Enumeration (1)

Idea: Program specified by infinite set of symbolic expressions that enumerate all possible inputs or sequences of inputs along with the corresponding outputs.

concatenation function:

```
(concat <seq1> <seq2> <seq1+seq2>)

(concat () () ())
    …
(concat (a b c) (d e) (a b c d e))
    …
```

Specification by Enumeration (2)

Program that reads input file and writes output file:

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

Sorting program enumeration:

```
(Program () ())
 …
(Program ("dog" "pig" "cat")   ! 3-line input
         ("cat" "dog" "pig")   ! output file
 …
```

Specification by Enumeration (3)

Interactive program:

```
(Program <outs> <in> <outs> <in> …
         <outs> <in> <outs>)
```

<in> - single line typed by user
<outs> -- 0 or more lines typed by program

Calculator program enumeration:

```
 …
(Program
  ("Enter expression to evaluate, empty line to halt:")
   "2 + 3"                          -- user's input line
  ("5")                             -- program's output line
   "   6*(1 + ( 5)) "               -- spaces between tokens ok
  ("36")
   ""                -- blank input line tells program to halt
  ("Bye! -- 2 expressions evaluated"))   -- last output line
 …
```

The Core Language – Expressions

Axioms generate valid expressions.

expression:

atom – a primitive indivisible element,

expression variable,

or **sequence** of zero or more expressions and **string variables**.

syntax:

atoms: **`` `abc ``**, **`` `+ ``**

expression variables: **`%w`**, **`%3`**

string variables: **`$`**, **`$xyz`**

sequences: **`()`**, **``(`M %x (`a $2))``**

The Core Language – Axioms

axiom:

conclusion expression and zero or more **condition** expressions

syntax:

```
<conclu> < <cond1>, …, <condn>.
<conclu>.        ! an unconditional axiom
```

The Core Language – Axiom Instances

axiom instance – substitute values for the axiom variables

- expression for an expression var
- string of >=0 expressions & string vars for a string var

axiom:
```
(`A %x $)< (`B %x %y), (`C $).
```

instance:
```
(`A `x `u %)< (`B `x ()), (`C `u %).
```

– substitute `` `x `` for `%x`, `()` for `%y`, and `` `u % `` for `$`

The Core Language – Valid Expressions

valid expressions – If the conditions of an axiom instance are valid expressions, the conclusion is a valid expression.

example axiom set:

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

instances:

```
(`a `b).
((`a) `b `b)< (`a `b).
(((`a)) `b `b `b `b)< ((`a) `b `b).
…
```

valid expressions:

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

Examples – Natural Number Addition

Set of natural numbers:

```
(`number (`0)).
(`number (`s $))< (`number ($)).
   → (`number (`0)),
     (`number (`s `0)),
     (`number (`s `s `0)),
      …
```

Addition of natural numbers:

```
(`plus %n (`0) %n)< (`number %n).
(`plus %1 (`s $2) (`s $3))<
  (`plus %1 ($2) ($3)).
 → (`plus (`0) (`0) (`0)),
   (`plus (`s `0) (`0) (`s `0)),
   (`plus (`0) (`s `0) (`s `0)),
     …
```

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 Functions on Sequences

Concatenation of sequences:

```
(concat ($1) ($2) ($1 $2)).
  → (concat (x y) (z) (x y z))
```

Membership in a sequence:

```
(member % ($1 % $2)).
  → (member b (a b c d))
```

Reverse of a sequence:

```
(reverse () ()).
(reverse (% $) ($rev %))<
  (reverse ($) ($rev)).
  → (reverse (u v) (v u))
```

Sorting Program Example (1)

Ordering of bit expressions:

```
(< `0 `1).
(< ($) ($ % $x)).   ! lexicographic ordering
(< ($ %1 $1) ($ %2 $2)< (< %1 %2).

(<= % %).
(<= %1 %2)< (< %1 %2).

(ordered ()).
(ordered (%)).
(ordered (% %1 $))< (ordered (%1 $)),
                           (<= % %1).
```

Sorting Program Example (2)

Permutation:

```
(permute () ()).
(permute (% $) ($1 % $2))<
    (permute ($) ($1 $2)).
```

Characters and strings:

```
(bit `0).
(bit `1).
(bitseq ()).
(bitseq (% $))< (bit %), (bitseq $).
(char (`char %8bits))< (bitseq %8bits),
    (=length %8bits (* * * *  * * * *)).
(=length () ()).
(=length (% $) (%2 $2))<(=length ($) ($2)).
(charstr ()).
(charstr (% $))< (char %), (charstr ($)).
```

Sorting Program Example (3)

File is sequence of character strings:

```
(file ()).
(file (% $))< (charstr %), (file ($)).
```

Sorting function on bit expressions:

```
(sort %seq %sorted)<
  (permute %seq %sorted),
  (ordered %sorted).
```

Program sorts input file:

```
(Program %input %output)< (file %input),
  (sort %input %output).
```

– see website for Calculator Program example

Metalanguage Example

Procedural language function as unconditional axiom:

```
(function FACTORIAL (N) is
    variables I FACT ;      !local var names
  begin     ! arguments & vars are untyped
    I := 0 ;
    FACT := 1 ;           ! = 0!
    while I < N loop      ! loop until I = N
      I := I + 1 ;
      FACT := FACT * I ;  ! FACT = I!
    end loop ;
  end FACTORIAL return FACT).  ! FACT = N!
```

– combines with language definition axioms (see website)

→ `(FACTORIAL (`s `s `s `0) (`s `s `s `s `s `s `0))`

– see [SEKE 2012] on website for Lisp-like example

Conclusions (1)

- Specifications
  - Smaller & more readable than algorithms
  - More reusable than code constrained by efficiency
    - Easier to generalize
    - Higher-order capability can extract common patterns

  → greater programmer productivity

- Minimal
  - Tiny Turing-complete formal system
  - Yet human readable, expressive, & extensible

- Metalanguage
  - Able to incorporate capabilities of other languages

Conclusions (2)

- Specification by enumeration
  - Specify external behavior without internal operation
  - Avoids awkwardness of I/O in declarative languges
    - Avoids ugliness of I/O operations in Prolog

- Simplicity & purity of language is well-suited to proof
  - Guarantee correctness of implementation
  - Prove assertions to validate specifications

  → improved software reliability

- Implementation Challenge!

wwwilson *at* acm *dot* org