Axiomatic Language

http://www.axiomaticlanguage.org/

Walter W. Wilson
wwwilson at acm dot org

Polyglot Programming DC
December 2, 2015

?

Ideal Programming Language?

?

?

Ideal *Formal*

Programming Language?

?

Outline

- Language Goals
- Main Idea – Specification by Enumeration
- The Core Language
  - Expressions
  - Axioms
  - Axiom Instances
  - Valid Expressions
- Syntax Extensions
- Examples
- Benefits
- Summary – Language Attributes

Language Goals

1. Pure specification – what, not how
2. Minimal, but extensible
3. Metalanguage – able to imitate other languages
4. Beauty!

Specification by Enumeration (1)

Idea: Program specified by infinite set of symbolic expressions that enumerate inputs and 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>)
```

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
   …
```

Specification by Enumeration (3)

Interactive program:

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

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

Enumeration of program to reverse user's input line:

```
 …
(Program
  ("Enter lines to reverse, empty line to halt:")
   "abc 123"                      – user's input line
  ("321 cba")                     – program's output line
   "Polyglot DC!"
  ("!CD tolgyloP")
   ""                             – blank line to halt
  ("Bye!"))                       – 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)
 …
```

Example – 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

```
  (Program %in %out)< (perm %in %out),
                      (ordered %out).
! <, <= - ordering of char strings
  (< `0 `1).            ! order of bits
  (< ($) ($ % $x)).     ! lexicographic ordering
  (< ($ %1 $1) ($ %2 $2))< (< %1 %2).
  (<= % %).     (<= %1 %2)< (< %1 %2).
! ordered – sequence is ordered
  (ordered ()).         ! empty seq is ordered
  (ordered (%)).        ! 1-elem seq is ordered
  (ordered (% %1 $))< (ordered (%1 $)),
                      (<= % %1).
! perm – permutation of a sequence
  (perm () ()).
  (perm (% $) ($1 % $2))< (perm ($) ($1 $2)).
```

Reverse Program Example

```
(Program
  ("Enter lines to reverse, empty line to halt")
   ""
  ("Bye!")).
(Program %begin %in (%out) $rest)<
  (Program %begin $rest),
  (reverse %in %out),
  (== %in (% $)).     ! input must be non-null

(== % %).      ! identical exprs
```

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

Benefits

- 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

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

  -> improved software reliability

Summary – Language Attributes (1)

- Goal of pure specification
  - Idealistic and ambitious
  - Must solve program synthesis grand challenge

- Minimal to the extreme

- Simple, clear semantics
  - No ugly non-logical operations
  - No built-in negation (but can be defined)
    (see “extended axiomatic language”)

- Specification by enumeration abstraction
  - Separates specification from implementation
  - Nice solution to I/O in declarative languages

Summary – Language Attributes (2)

- Higher-order power

- Metalanguage capability
  - Can incorporate advantages of other languages
  - Good host for embedded DSLs

- Potential software engineering benefit
  - Smaller code size → greater productivity
  - Proof → fewer bugs

Axiomatic Language

Power – Potential – Beauty