

A Tiny Specification Metalanguage

Walter W. Wilson, Yu Lei
The University of Texas at Arlington

www.axiomaticlanguage.org

SEKE 2012
July 1-3, 2012
Redwood City, California

Language Goals

- 1) pure specification language
 - what, not how
 - smart translator needed
- 2) minimal, but extensible
 - as small and simple as possible
 - nothing built-in that can be defined
- 3) metalanguage
 - able to imitate other languages

Specification by Interpretation

Idea: External behavior specified by a static infinite set of symbolic expressions that enumerate inputs and corresponding outputs.

append function:

(append () () ())

...

(append (a b) (c d e) (a b c d e))

...

Specification by Interpretation (2)

a program:

(Program *<input>* *<output>*)

sorting program:

...

(Program ("dog" "horse" "cow") ! input
("cow" "dog" "horse")) ! output

...

-- Infinite set of these expressions specifies sorting program.

Specification by Interpretation (3)

Interactive program:

(Program *<out>* *<in>* *<out>* ... *<in>* *<out>*)

-- *<out>* is 0 or more lines typed by program

-- *<in>* is 1 line typed by user

Each Program expression defines a possible execution history.

Infinite set of these expressions specifies the interactive program.

-- No awkward, ugly I/O operations.

Axiomatic language is just a formal system for defining these infinite sets.

Overview

- Pure, definite Prolog with Lisp syntax
- Higher-order generalization [HiLog, 1993]
- “string variables”

The Core Language

Finite set of axioms generates infinite set of valid expressions.

an **expression**:

an **atom** – a primitive, indivisible element,

an **expression variable**,

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

syntax:

atoms: `abc, `+

expression variables: %1, %n

string variables: \$xyz, \$

sequences: (), (`M (%x \$2))

The Core Language (2)

axiom – a **conclusion** expression and zero or more **condition** exprs:

<conclu> < *<cond1>*, ..., *<condn>*.

<conclu>. ! unconditional axiom

axiom instance - substitute values for expression and string variables

– arbitrary expression for an expression variable

– string of expressions and string variables for a string variable

(`a %x \$1) < (`b \$1 %x).

→ (`a `c (\$) `d) < (`b (\$) `d `c).

The Core Language (3)

valid expression – conclusion of axiom instance is valid expression
if all conditions are valid expressions

$(\text{`a `b}).$

$((\%) \$ \$) < (\% \$).$

→

$(\text{`a `b}),$

$((\text{`a) `b `b}),$

$(((\text{`a})) `b `b `b `b),$

...

Syntax Extensions

characters & strings:

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

(... 'abc' ...) = (... 'a' 'b' 'c' ...)

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

symbols:

abc = (` "abc")

Example – Sorting

! Program – sorting program

(Program %i n %out) < (perm %i n %out), (ordered %out).

! <, <= - ordering of char strings

(< `0 `1). ! order of bits

(< (\$) (\$ %x \$x)). ! lexicographic ordering

(< (\$ %1 \$1) (\$ %2 \$2)) < (< %1 %2).

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

! ordered – ordered sequence

(ordered ()). ! empty seq ordered

(ordered (%)). ! 1- elem seq ordered

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

! perm – permutation of a sequence

(perm () ()).

(perm (\$1 % \$2) (\$3 % \$4)) < (perm (\$1 \$2) (\$3 \$4)).

Lines of Code Comparison

phonecode [Prechelt 2000] – min & median non-comment loc:

- tcl – 44, 100
- rexx – 53, 120
- python – 42, 85
- perl – 49, 75
- Java – 107, 240
- C++ – 150, 235
- C – 188, 240

axiomatic language: 54 (non-utility code) (not tested)

Lines of Code Comparison (2)

minimum spanning forest:

minimum spanning tree examples (non-i/o): 25, 34, 49, 65

axiomatic language (MSF): 15 (non-utility) (not tested)

<http://www.axiomaticlanguage.org/examples.html>

Conclusions

- Language Attributes
 - Pure specification – what declarative programming should be
 - Minimal in the extreme
 - Simple, clear semantics
 - No ugly non-logical features
 - Specification by interpretation
 - No awkward non-declarative input/output
 - Higher-order power
 - Metalanguage capability
- SE benefit
 - Greater reusability, smaller code size?
 - Need more examples!
- Difficulty of implementation