Little (but Hard) Theorems About Big Systems: Some Case Studies

J Strother Moore
Department of Computer Sciences
University of Texas at Austin

Marktoberdorf Summer School 2004

Lecture 4

Files from Last Time

- lecture3.ps – slides
- lecture3.proofs – session log of `isort` and `qsort`
- perm.lisp – ACL2 "book" loaded in session
- filter.lisp – ACL2 "book" loaded in session

- inadmissible.lisp – ACL2 proof of 2=7 from

```
(defun f (x)
  (if (equal x 1)
      nil
      (cons nil (f (- x 1)))))
```

  which is inadmissible (but can be added as an axiom).

Today

- `(subp x x)` – hide your eyes if you don't want to see my solution!
- a simple stack machine model
- a simple expression language model
- a simple compiler
- proof that the compiler is correct

The Method on `(subp x x)`

demo

The Stack Machine

We will formalize a machine:

`m` :

$programs \times environments \times stacks \rightarrow stacks$

Sample program:

```
((LOAD A)
 (PUSH 3)
 (MUL))
```

Instruction set:

(LOAD var

(PUSH $const$)

(ADD)

(MUL)

(DUP)

No instruction will change the environment (no STORE instruction).

Sample environment:

```
((A    . 20)
 (B    . 30)
 (X    . -5)
 (TEMP . 18))
```

Sample stack:

```
(push 1 (push 2 (push 3 nil)))
```

$\Rightarrow$

```
(1 2 3)
```

Expressions

We will compile simple expressions into this language and prove that we did it correctly.

The expression language is

```
<expr> := <variable> |
          <constant> |
          <unary-application> |
          <binary-application>
```

```
<unary-application>
  := (- <expr>) |
     (sq <expr>)

<binary-application>
  := (<expr> + <expr>) |
     (<expr> - <expr>) |
     (<expr> * <expr>)
```

Obvious Criticisms

Everything is trivial!

No `STORE` instruction.

No program counter.

No branch or conditionals.

No object creation.

No method invocation.

No exceptions or errors.

Response to the Criticism

Tomorrow we will handle all that and more.

But today I want you to really *understand* what is involved in modeling two computational paradigms and proving their correspondence formally.

I'll follow "The Method" to do my proofs and show you everything (except some trivial "abstract data type" work at the bottom).