From Programs to Executions: An Odyssey in Language Translation

(with examples in Scheme)

Keith D. Cooper
Rice University
Houston, Texas
December 2000

An Example

Sum the series

\[ n + n-1 + n-2 + \ldots + 1 \]

In Scheme, we might write

\[
\text{(define (summation n)}
\text{(cond}) [\text{(= n 0) 0]}
\text{[else (+ n (summation (sub1 n)))])})
\text{(summation 3)}
\]

How do we really go from (summation 3) to an answer?
We explain DrScheme’s behavior by saying that it performs a series of rewriting steps:

\[
\text{(summation 3)} \\
\Rightarrow \ (\text{cond} \ [(= 3 0) 0] \\
\text{[else} \ (+ 3 \ (\text{summation (sub1 3))})]) \\\n\Rightarrow (+ 3 \ (\text{summation 2})) \\
\Rightarrow (+ 3 \ (\text{cond} \ [(= 2 0) 0] \\
\text{[else} \ (+ 2 \ (\text{summation (sub1 2))})])]) \\
\Rightarrow (+ 3 (+ 2 \ (\text{summation 1}))) \\
\Rightarrow (+ 3 (+ 2 \ (\text{cond} \ [(= 1 0) 0] \\
\text{[else} \ (+ 1 \ (\text{summation (sub1 1))})])]) \\
\Rightarrow (+ 3 (+ 2 (+ 1 \ (\text{cond} \ [(= 0 0) 0] \\
\text{[else} \ (+ 0 \ (\text{summation (sub1 0))})])])]) \\
\Rightarrow (+ 3 (+ 2 (+ 1 0))) \\
\Rightarrow (+ 3 (+ 2 1)) \\
\Rightarrow (+ 3 3) \\
\Rightarrow 6
\]

... a long series of rewriting steps ...

It eventually produces the answer: 6

Is that how it really works? Probably not

Does it matter? Not unless we can tell the difference
The Big Lie(s)

Programming languages deal with abstractions
• Infinite precision numbers
• Symbols
• Lists, structs, vectors, trees
• Functions, programs, name spaces

Computers deal with a limited repertoire of simpler ideas
• Finite integers, floating-point numbers
• Memory locations
• Small set of fundamental operations

Language implementation must make good on the lies!

What is DrScheme?

Imagine a contract for DrScheme:

\[ \text{DrScheme: program } x \text{ inputs } \rightarrow \text{ results} \]

DrScheme is a program that manipulates programs

In particular, it
• Creates and maintains the Scheme Environment
  > Functions, objects, definitions,
  > Abstractions like “local” and “define-struct”
• Checks to see that programs are well formed
• Executes programs

DrScheme implements the programming language Scheme
Implementing Programming Languages

Two principal ways to “implement” a language

- Interpreter: program x inputs → results

- Compiler: program → program

Inside an Interpreter

- Represent the program in some internal form
  
  (+ 3 4 5) ⇒ (list + 3 4 5)

- Traverse that data structure and produce answers
  
  (list + 3 4 5) ⇒ 12

Along the way

- Manages the name space
  - Variables, arguments/parameters, symbols, free variables
- Manages storage (the computer’s memory)
- Manages communication with outside world
  - Programmer or user, external files, other programs …
The Conceptual View

Programs → Executions

The Conceptual View

DrScheme

1. You enter your code in the definitions window
2. You enter an expression in the interactions window
3. DrScheme rewrites until it has a solution

What Really Happens?

Behind the scene, the computer runs the program “DrScheme”

• DrScheme is a program that executes, or interprets, Scheme programs.
• You know enough to write a simple version of this program!

• Scheme Environment only exists in the interpreter’s memory and the programmer’s imagination
What does this “computer” look like?

How does it work?

Think of registers & memory as vectors in Scheme

Think of the functional & control units as programs that consume commands & create side effects (set!)
Are the commands in Scheme?

![Diagram showing registers, functional unit, data memory, program memory, and control unit with PC.]

Such computers have been built

- They have not proven to be cost effective
- More general processors are the rule (today)

What commands does the “computer” run?

Computer’s instruction set

- Low-level, imperative commands
  - Arithmetic operations
  - Memory operations
  - Control operations
  - Location-oriented programming

- We call these operations “assembly-language”

<table>
<thead>
<tr>
<th>Arithmetic Operations</th>
<th>Memory Operations</th>
<th>Control Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>add r1, r2 =&gt; r3</td>
<td>load r1 =&gt; r2</td>
<td>branch r1 -&gt; r2</td>
</tr>
<tr>
<td>sub r1, r2 =&gt; r3</td>
<td>store r1 =&gt; r2</td>
<td>branchj r1 -&gt; t2</td>
</tr>
<tr>
<td>mult r1, r2 =&gt; r3</td>
<td>loadj c1 =&gt; r2</td>
<td>call t1 -&gt; t.1</td>
</tr>
<tr>
<td>div r1, r2 =&gt; r3</td>
<td>copy r1 =&gt; r2</td>
<td>return</td>
</tr>
</tbody>
</table>