The Value-Passing Translator User Guide

Glenn Bruns
Bell Labs, Lucent Technologies
`grb@research.bell-labs.com`

December 3, 1996

1 Introduction

This document describes how to install and use `vp`, the value-passing CCS translator. This tool takes value-passing CCS as input and produces basic CCS as output. See [3] for more information on basic and value-passing CCS, and [1] for more information on the particular value-passing notation used by the translator. The output of `vp` is a file that can be loaded by the Edinburgh Concurrency Workbench. See [2] and URL

`http://www.dcs.ed.ac.uk/home/cwb`

for more information on the Edinburgh Concurrency Workbench.

2 Installation

`vp` is currently distributed as a compressed binary for Sun Sparc workstations. Therefore, installation involves simply uncompressing the file using the Unix `compress` program, and then moving the binary to a suitable directory.

3 Running the Translator

The translator is invoked as a Unix command, with a single parameter that names a file containing the value-passing CCS program. The translator expects the input file to have a '`vp`' suffix, but this suffix should not be included on the input line. For example, to translate the scheduler program `sched.vp` of the appendix:

```
% vp sched
```

Here the translator takes as input file `sched.vp` and produces as output file `sched.ccs`. The output file can be loaded by the Edinburgh Concurrency Workbench.

4 Implementation

To understand the error messages produced by `vp` one should know roughly how `vp` works. CCS programs are translated in two steps. First, the CCS program is translated to an ML program that consists of the application of a translation function to an ML data value. This value represents the abstract syntax of the agents in the CCS program. Second, the ML program is run to produce a file of basic CCS agent definitions. The advantage of using two steps is that much of the translation work can be done by the ML compiler. In particular, the binding of variables in the value-passing CCS program is done by using the 'let' construct of ML. The drawback of using two steps is that a badly-formed CCS program will lead to ML code that is poorly typed. As a result, some erroneous CCS programs cause the translator to fail and produce ML error messages.

5 Understanding Error Messages

Three kinds of errors are reported by `vp`. First are the syntax errors, which are reported by giving the line number of the first such error. Consider the label declaration of the scheduler program given in the appendix:

```
label a_nagents, b_nagents
```

If the comma of this line is replaced by a period, the tranlator reports:

```
syntax error found at DOT Line 7.
```

Next are the errors that lead to ML type violations. If the label declaration of the scheduler program is changed so that label b is not declared, the translator fails and the ML interpreter reports:

```
sched.ml:9.30-9.32 Error: unbound variable or constructor: vpb
sched.ml:18.30-18.32 Error: unbound variable or constructor: vpb
```

The translator prepends the string `vp` to all variables in user programs, so variable `vpb` refers to user variable `b`. The ML program complains that the variable is unbound because it has not been declared in the value-passing CCS program.

Last are the errors that are reported during the second phase of translation. If the label declaration of the scheduler program is changed so that label b is declared as a simple label, the translator reports:

```
Action type error: name b_0 used inconsistently with declaration.
```

The message is presented because label `b` has a parameter in the body of the program, but is declared as a simple label.

A Scheduler Program

```
(* Scheduler specification from page 115 of [3] *)

const n     = 3
const Index = {0,1,2}
const Iset  = pow(Index)

label a_Index, b_Index

agent Schedspec(i:Index,X:Iset) =
    if member(i,X) then
        sum(j:X,b_j.Schedspec(i,diff(X,{j})))
    else
        a_i.Schedspec((i + 1) mod n,union(X,{i}))
      + sum(j:X,b_j.Schedspec(i,diff(X,{j})))
```

References

[1] Glenn Bruns. *Distributed Systems Analysis with CCS.* Prentice Hall, 1996.

[2] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The Concurrency Workbench: A semantics based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, 1993.

[3] Robin Milner. *Communication and Concurrency.* Prentice Hall International, 1989.