Bill J. Ellis
June 13, 2003

Mechanics of Visualising VCs

1 Introduction

This is a short note to describe a possible architecture for visualising SPARK Verification Conditions (VCs). The key requirements expected of such an architecture are:

- **Exploit existing GUI support -** Building a GUI from scratch is fantastically complicated. An existing GUI system should be used instead.
- **Retrieve VCs -** The VCs will need to be loaded into the system.
- **Work with VCs -** To work with the VCs, they will need to exist inside some VC data structure. This data structure will need to support both the transformation and eventual presentation of VCs.

2 Prolog, powerful Prolog

Every VC is collection of Prolog terms. For example:

```
procedure_data_filter_6.
H1:     for_all (i___1: integer, ((i___1 >= i_type__first) and
        (i___1 <= i_type__last)) -> ((element(d, [i___1]) >=
        integer__first) and (element(d, [i___1]) <= integer__last))) .
H2:    loop__1__i >= i_type__first .
H3:    loop__1__i <= i_type__last .
H4:    element(d, [loop__1__i]) >= 0 .
H5:    element(d, [loop__1__i]) <= 100 .
    ->
C1:    r + element(d, [loop__1__i]) >= integer__first .
C2:    r + element(d, [loop__1__i]) <= integer__last .
C3:    loop__1__i >= i_type__first .
C4:    loop__1__i <= i_type__last .
```

It is straight forward to load these VCs into Prolog. Code for this purpose already exists as part of the NuSPADE project. This process is straight forward as Prolog *is* a term processing system. More significantly, Prolog can then work with these terms using powerful pattern matching facilities. For example:

```
| ?- for_all(VAR: _, ((VAR >= FROM) and (VAR <= TO)) -> EXPR) =
         for_all(i___1: integer, ((i___1 >= i_type__first) and
         (i___1 <= i_type__last)) -> ((element(d, [i___1]) >=
         integer__first) and (element(d, [i___1]) <= integer__last))).

VAR = i___1,
FROM = i_type__first,
TO = i_type__last,
EXPR = element(d,[i___1])>=integer__first and element(d,[i___1])<=integer__last
```

3 Java, beautiful Java

Java (http://java.sun.com/) and the Java Foundation Classes (http://java.sun.com/products/jfc/), provide comprehensive GUI and drawing facilities. In particular, Swing (GUI components) and Java 2D (2D drawing) are likely to be useful for presenting VCs within a user friendly environment.

*Notes in this series are for ϵ-baked ideas, for $1 \geq \epsilon \geq 0$. Only exceptionally should they be cited or distributed outwith the NuSPADE project (EPSRC GR/R24081).

4 Java and Prolog

Based on the above observations, it is reasonable to join Java and Prolog to provide the architecture of the VC visualisation system. Sicstus (`http://www.sics.se/sicstus/`) have implemented Jasper (`http://www.sics.se/sicstus/docs/latest/html/sicstus/Mixing-Java-and-Prolog.html`), a bi-directional interface between Java and Sicstus Prolog, for this very purpose. Thus the following might be considered:

```
/---------\  calls   /------------\
|Java     |--------->|Prolog      |
|(Display)|          |(Processing)|
|         |<---------|            |
\---------/ results  \------------/
                            ^
                            |load the external VC files
                            |
                        |------|
                        |VCs   |
                        |------|
```

Note that this places Java as the main program. It would be possible to instead have Prolog as the main program, calling Java as required. However, as Java provides the GUI, it seems more appropriate to give it the main program loop.

Note that 'results' requires Java to have a suitable data structure. Such a data structure is provided as part of Jasper. By placing the sophisticated (heuristic based) term processing in Prolog, Java can focus on processing these results for display.

5 Yes, fine, but does it work?

Sicstus is a commercial Prolog system with generally good documentation. The Java part of Jasper is documented through Javadoc (`http://www.sics.se/sicstus/docs/latest/html/jasper/`) and the Prolog part within the regular Sicstus manuals. There are some 'hello world' like applications to experiment with the system.

I have not used Jasper for longer than a few hours. However, during these hours, i managed to get the examples running on my home computer.