Tools for Automatic Interface Generation in Scheme

Augustin Lux
Project Orion
INRIA - Sophia Antipolis, France *
Email: Augustin.Lux@sophia.inria.fr

Abstract

We present present two basic tools for the formal manipulation of C/C++ programs: a general syntax analyzer for C++, producing Abstract Syntax Trees, and an "unparser" translating ASTs to program text. These two Scheme programs are in particular used in our local Scheme implementation to automatically integrate C++ code into Scheme. The hard part of this problem is wrapper generation which is solved by a third tool working entirely on the AST level. The first two of these tools are completely general, and should be useful for numerous applications. The wrapper generator can easily be adapted to other implementations. The strong points of our tools are: handling of (almost) complete C++ (including overloading, operators, templates), use of ASTs for C++ programs, and the availability as Scheme code.

1 Automatic Interface Generation

Scheme, inasmuch as it is an incarnation of λ-calculus, is a completely open-ended language. From an abstract point of view, there is no reason why procedures and methods written in C or C++ cannot just be loaded into a Scheme environment. However, technically this is a hairy problem. Some seven years ago, in their paper on "Sweet Harmony", Davis et al. suggested how easy and transparent this connection should be (and also listed some of the inherent difficulties). In particular, they proposed that the "foreign function interface" procedures necessary to link C-code into a Lisp interpreter be produced automatically, from the information given by C++ header files. This idea has been taken up in several Scheme systems, as well as for other languages, like Perl, Tcl, Python. The home page of the "Simplified Wrapper and Interface Generator" SWIG gives a useful overwiew. However, all these program generators impose some limitations on the complexity of the C++ code that can be automatically handled.

2 Tools for Program Synthesis and Interface Generation

Because we wanted tight integration between Scheme and C++ in our Scheme platform called Ravi [1], we have developed a set of new tools, all written in Scheme, for program synthesis in general and for wrapper generation in particular.

[1] for use in Robotics, Ai, and computer VIsion

The first of these tools is a C++ parser that produces a nice abstract-tree structure for C++ declarations (and instructions). The general form of a declaration is

```
(c-declare storage-class name type [opt-fields])
```

For instance, `float (*tab[5])();`
has the abstract form

```
(c-declare #f tab (c-array (* (-> () float))) 5)
```

The abstract tree structure is completely independent of any application, we have also used it, for instance, to generate XML-interfaces. The unparser tool translates abstract tree structures into C++ files, that can then be compiled and linked with Scheme code.

Using these general tools, the interface generator works with abstract trees only, which makes the problem much more manageable.

All these programs tackle (almost) the full generality of real world programs: overloading, operator definitions, templates, preprocessor directives. Important information that cannot be deduced from header files (e.g. result parameters) may be supplied in a separate file, or in the form of special comments.

As an example, here is the interface function generated for the constructor of a class *Point*, defined as follows:

```
class Point{
public: Point (void);
        Point (int, int);
        Point (Point&);
   ...
```

The constructor shows the handling of overloading, using systematic type tests to guarantee correct argument types:

```
void Sc_Point_Sconstructor()
{
  ScVal * FB = & VS_ElemQ(- GetPar());
  int nbarg = ScMV::CI;
  if(nbarg == 0)
    SetResult(TypeC(TypeC_no_1,(void *)(new Point())));
  else if(IsFixNum(FB[0]))
    {
      int param_0 = GetFixVal(FB[0]);
      if(nbarg == 2)
        if(IsFixNum(FB[1]))
          {
            int param_1 = GetFixVal(FB[1]);
            SetResult(TypeC(TypeC_no_1,
                      (void *)(new Point(param_0,
                                         param_1))));
```

```
          }
        else goto bad;
      else goto bad;
    }
  else if(CheckTypeC(FB[0],TypeC_no_1))
    {
      Point * param_0 = (Point *)GetCPtr(FB[0]);
      if(nbarg == 1)
        SetResult(TypeC(TypeC_no_1,
                  (void *)(new Point(* param_0))));
      else goto bad;
    }
  else goto bad;
  return;
  bad:
Errorf("bad args for function %s",
       "Point::constructor");}
```

3 Conclusion

The interface generator is now used on a routine basis, especially for large programs in image processing containing several thousand methods. By using the C++-parser instead of the Scheme read function, we are also able to mimick something like a C++ interpreter. For the source code of the programs, see http://www-prima.inrialpes.fr/Ravi

4 References

[1] www.swig.org/index.html
Simplified Wrapper and Interface Generator
[2] http://www-prima.imag.fr/Ravi
[3] H.Davis et al.: Sweet harmony: the Talk/C++ connection. 1994 ACM Lisp Conference, p. 121