Correctness Issues in Transformational Refinement

Peter Kilpatrick, M Clint, T J Harmer and S Fitzpatrick

Department of Computer Science, The Queen's University of Belfast,
Belfast BT7 1NN, Northern Ireland

Transformational refinement in the context of this talk refers to the *automatic* transformation of a specification expressed in a functional programming language (typically SML, LISP or Miranda) to an efficient implementation expressed in an imperative programming language (typically FORTRAN or one of its parallel derivitives). In particular, our work focuses on the derivation of efficient numerical mathematical algorithms for execution on a range of parallel machines.
A transformational derivation proceeds not as a monolithic translation process (c.f. compilation) but rather as a sequence of sub-derivations which convert a program via a number of intermediate forms which lie between its specification expressed in a specification language and its implementation expressed in the target language. For example, rather than being translated directly from SML to Fortran, a specification may first be converted to the λ-notation, then into Fortran:

$$\text{SML} \longrightarrow \lambda\text{-notation} \longrightarrow \text{Fortran.}$$

Each of these transitions may be sub-divided into further intermediate forms. Thus a derivation is divided into sub-derivations, with one sub-derivation being used to create one intermediate form. For example, one sub-derivation may be responsible for function unfolding; another for common sub-expression elimination, etc.
A (sub-)derivation consists of a sequence of *transformation rules*. A transformation rule is a rewrite rule consisting of a pattern and a replacement, both defined using a wide spectrum grammar. For example,

```
<ident>"1" + <ident>"1" ==> 2*<ident>"1"
```

is a simple transformation rule. The <ident>s are non-terminals (corresponding to 'identifiers') in the grammar. The same label ("1") requires the non-terminals to match the same identifier; in the replacement <ident>"1" refers to whatever identifier was matched in the pattern. Thus this transformation rule will convert the expression `x+x` to `2 * x`.

A transformational refinement is applied to a specification by a transformation engine, TAMPR, which operates by constructing a syntax tree representation of the specification and applying the sequence of transformations comprising the derivation using an exhaustive post-order application strategy. That is, each transformation rule (or set of related rules) is repeatedly applied to the tree until no further application is possible.
In this talk we consider issues relating to the correctness of this transformation system. We first establish what we mean by correctness in this context and then describe a framework in which correctness proofs may be undertaken.
A transformational derivation is correct provided:

- application of each transformation rule in the derivation preserves the meaning of the tree;
- application of a transformation rule (or set of rules) terminates; and
- the result of applying the derivation is a valid program segment in the target language.

The wide spectrum grammar is defined using VDM-SL notation and the transformation rules are described as VDM-SL functions.