Generic programming in the mCRL2 tool set

Wieger Wesselink, Jeroen van der Wulp and Jeroen Keiren

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{j.w.wesselink, j.j.a.keiren}@tue.nl, jeroen.vanderwulp@gmail.com

1 Introduction

The mCRL2 tool set [GKM+08] is a tool set for verification and validation of concurrent processes, based on process algebra specifications. The mCRL2 language is based on the Algebra of Communicating Processes (ACP), which is extended to include data and time. This paper reports on experiences with generic programming in C++ as applied in the implementation of the tool set. C++ *concepts*, a type system for templates [RS06], form a key ingredient of this style of programming. Using concept definitions, requirements on template types can be defined that are type checked during compile time. The main benefits for the mCRL2 tool set are uniform and flexible interfaces that operate on well-defined types, and a significant increase in code reuse. The use of concepts also promotes the writing of code that corresponds closely to pseudo code, since the chosen concepts correspond naturally with domain specific concepts. This will be illustrated by a simple use case, namely substitution functions.

Generic programming is about generalizing software components, to enable reuse in a wide variety of situations. In C++, generic programming is enabled using templates. C++ *concepts* are proposed as a means to type check template types. A concept is a set of requirements (valid expressions, associated types, semantic invariants, complexity guarantees, and so on) that a type must fulfill to be correctly used as an argument in a call to a generic algorithm, see [RS06]. Language support for concepts has been proposed [GJS+06] for the next version of the C++ standard, C++0x. Concepts will be used to make the specification of the C++ standard library more complete and precise. A derivative of the GNU C++ compiler [Gre08] already implements language support for concepts. In the mCRL2 tool set we have used a portable library for concept checking.

Most uses of generic programming in general, and more specifically the use of concepts, that are described in the literature treat the construction of data structures and algorithms that operate on these, see e.g. [GL05].

2 The mCRL2 tool set

The mCRL2 tool set stores data structures like state spaces and parameterized boolean equation systems using terms. A frequent operation in algorithms on

these data structures is substitution on terms. A substitution concept has been defined, and applied throughout the tool set, together with a traversal framework for data structures. This has made quite a number of existing replace functions on terms obsolete, and many more ad hoc solutions for less general substitution problems as well. The introduction of the substitution concept has also opened the way to easily experiment with different implementations of substitution functions, instead of with a fixed number of predefined ones. C++ concepts have been applied at several other places. We have, for example, defined a rewriter concept that has helped to make the term rewriter fully replaceable.

3 Example

By means of example we take the substitution concept that has been defined in the tool set. A substitution is a function that maps variables from a domain $\mathbb{V}$ to terms of a range $\mathbb{T}$. Let x be a variable, t a term, and $\sigma : \mathbb{V} \to \mathbb{T}$ a substitution that models a `Substitution` concept, that has the following requirements:

- `Substitution::variable_type` is a valid C++ type
- `Substitution::term_type` is a valid C++ type
- $\sigma(x)$ is a valid expression of type `Substitution::term_type`
- $\sigma[x] = t$ is a valid C++ expression

The first three requirements correspond to the ordinary mathematical function concept. We have used the domain specific names `variable_type` and `term_type` instead of the usual names `argument_type` and `result_type`. The third requirement defines the syntax for function application and expresses that the function call operator must be overloaded. The last requirement applies to mutable substitutions, and defines the syntax for updating a substitution. The semantics of the statement $\sigma[x] = t$ is that $\sigma := \sigma[x \to t]$, where $\sigma[x \to t](y) = t$ if $y = x$ and $\sigma[x \to t](y) = \sigma(y)$ otherwise.

An algorithm that takes an argument σ that models the `Substitution` concept, can only rely on the four requirements that are given here. Using a technique called concept checking one can detect at compile time if an argument type satisfies all these requirements. Moreover, one can easily test if an algorithm does not inadvertently use other properties of a type. As an example we consider the implementation of a function f that applies the substitution $x := t$ to a process p:

```
void f(process &p, variable x, term t) {
  substitution sigma;    // use a predefined map based substitution class
  sigma[x] = t;          // add the substitution x -> t to sigma
  substitute(p, sigma);  // apply sigma globally to terms in p
}
```

Here `substitute` is a generic function with the following interface:

```
template <typename Object, typename Substitution>
void rewrite(Object& o, Substitution sigma);
```

It applies σ to an object o that may be one of the mCRL2 data structures containing terms. Overloads have been added for C++ standard containers and mCRL2 specific containers containing elements of these data structures. To implement the `substitute` function a reusable traversal framework for hierarchical mCRL2 data structures has been defined.

4 Discussion

The substitution concept that was discussed in the previous section may look deceptively simple. Applying this concept globally however has lead to a remarkable cleanup of the code. At many places interfaces have become simpler and at the same time more generic. Virtually all of the more complicated and less generic solutions for substitutions could be removed. An interesting and challenging application was the definition of a term rewriter concept. Applying this concept globally took a significant effort, caused by problems in the existing interface. The result of this operation is that many algorithms in the mCRL2 tool set that require a rewriter for their computations have been parameterized with a `Rewriter` template argument. This has significant advantages. First of all it opens the possibility to experiment with different implementations of a rewriter without having to modify the code of these algorithms. This is useful for instance for doing benchmarks. Another advantage of using concepts in the interfaces of algorithms is the explicit documentation of type requirements. Having clear semantics of types is a necessary step towards making correct implementations. Finally, the usage of concepts promotes the writing of clean code that corresponds almost one to one to pseudo code descriptions of algorithms. For many algorithms in the mCRL2 tool set descriptions in pseudo code are available.

Concluding we can say that the application of generic programming techniques in the mCRL2 tool set attributes to increased productivity through uniform and flexible interfaces, and to decreased maintenance costs through an increased level of reuse of code written at a higher level of abstraction.

References

[GJS+06] D. Gregor, J. Järvi, J. Siek, B. Stroustrup, G. Dos Reis, and A. Lumsdaine. Concepts: Linguistic support for generic programming in C++. In *Proc. OOPSLA'06*, pages 291–310. ACM Press, October 2006.

[GKM+08] J.F. Groote, J. Keiren, A. Mathijssen, B. Ploeger, F. Stappers, C. Tankink, Y. Usenko, M. van Weerdenburg, W. Wesselink, T. Willemse, and J. van der Wulp. The mCRL2 toolset. In *Proc. WASDeTT'08)*, 2008.

[GL05] R. Garcia and A. Lumsdaine. MultiArray: a C++ library for generic programming with arrays. *Softw., Pract. Exper.*, 35(2):159–188, 2005.

[Gre08] D. Gregor. ConceptGCC — a prototype compiler for cpp concepts. http://www.generic-programming.org/software/ConceptGCC/, January 2008.

[RS06] G. Dos Reis and B. Stroustrup. Specifying c++ concepts. In *Proc. POPL'06*, pages 295–308, New York, NY, USA, 2006. ACM.