Chapter 5

Syntax-Safe Templates

Writing templates, and code generators in general, is a complex and error prone task. This complexity mainly results from mixing multiple languages in a template, executed at different stages, and the incompleteness of the object code. Manual verification of incomplete object code is hard to do and computers cannot execute incomplete code.

Text-based template evaluators do not improve the situation, as they are not able to check the object code. They are only aware of the syntax of the metalanguage, while the object language is considered as a string without any required structure. These evaluators only process and check the metacode and do not deal with the correctness of the rest of the template. Ignorance of the correctness of the object code can lead to undetected syntax errors [Sheard (2001)]. Misspellings in the object code, such as missing semicolons, are easily made and in such case text-based template evaluators generate syntactically incorrect code without giving a warning.

A test approach based on generating all possible outputs followed by verifying the result using a compiler or interpreter seems a valid route. However, to guarantee that a template generates syntactically correct sentences during production use, a possibly exploding amount of input data test cases must be defined for every template. Furthermore, an error must be manually traced back to its origin, which is not always obvious, such as sometimes experienced when using the C preprocessor [Ernst et al. (2002)], where the compiler error messages point to the post-processed code instead of the original source code. Checking the template directly offers accurate error messages, pointing to the origin of the error.

Beside the problems during development, dynamic text-based code generation as used in web applications can result in serious security issues, like malicious code injection. An example of malicious code injection is discussed in Chapter 7.

---

1 The complexity here considered is complexity in the broadest sense of the word and not for example computational complexity.
In order to remove the possibility of syntax errors in the generated code, the notion of syntax-safety for templates is introduced in this chapter [Arnoldus et al. (2007)]. Syntax-safety is a property of a code generator, where for every possible input the output of a syntax-safe code generator can be recognized by a parser for the intended resulting language, i.e. the code generator produces output sentences of the language $\mathcal{L}(G_{\text{intended}})$. The intended language is the language for which the code generator should produce sentences, for example, Java or C.

This chapter presents an approach based on constructing a grammar for templates containing the definition of the metalanguage and the object language of a template. The construction of a template grammar is generic and based on the combination of the metalanguage grammar and the (off-the-shelf) object language grammar, where only a combination grammar connecting both has to be defined manually.

The benefit of a template grammar is that not only the output code is syntactically correct, but also syntax errors are found in the template itself. Having a template grammar enables parsing the complete template, which ensures that all (sub)sentences are syntactically correct, without the need of compiling or interpreting generated code. Hence, this approach helps to avoid syntax errors, both in the metacode and in the object code, before the template is used for generating code.

Figure 5.1 shows the architecture used for syntax-safe template evaluation. The first stage is parsing the template using a template grammar resulting in a parse tree of the template. The second stage is the evaluation of templates. The evaluator uses the parsed template and input data as input and generates the output code. A syntax-safe template evaluator called Repleo is implemented. Repleo is a generic syntax-safe template evaluator system parameterized with the object language grammar to ensure the output is syntactically correct. The metalanguage of Repleo is unparser-complete, as discussed in the previous chapters. Chapter 6 discusses this evaluator.

### 5.1 Syntax-Safe Templates

During the discussion of the metalanguage for templates in Chapter 4, the object language of templates was ignored. The object language was considered as strings, i.e. sequences of alphabet symbols. Considering the object language as strings does not guarantee that the output sentences are well-formed with respect to their intended output language $\mathcal{L}(G_{\text{intended}})$. This section will discuss the requirements to ensure that a template cannot produce sentences that are not in $\mathcal{L}(G_{\text{intended}})$. 