Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
Electronic Notes in Theoretical Computer Science 124 (2005) 101–112
www.elsevier.com/locate/entcs

Programmable Rewriting Strategies in Haskell
— White Paper —

Ralf Lämmel [1,2]

[1] Vrije Universiteit, Amsterdam
[2] Centrum voor Wiskunde en Informatica, Amsterdam

Abstract

Programmable rewriting strategies provide a valuable tool for implementing traversal functionality in grammar-driven (or schema-driven) tools. The working Haskell programmer has access to programmable rewriting strategies via two similar options: (i) the *Strafunski* bundle for generic functional programming and language processing, and (ii) the "*Scrap Your Boilerplate*" approach to generic functional programming. Basic rewrite steps are encoded as monomorphic functions on datatypes. Rewriting strategies are polymorphic functions composed from appropriate basic strategy combinators.

We will briefly review programmable rewriting strategies in Haskell. We will address the following questions:

- What are the merits of Haskellish strategies?
- What is the relation between strategic programming and generic programming?
- What are the challenges for future work on functional strategies?

Keywords: Rewrite startegies, programming languages, Haskell, functional programming

1 Strategic programming

Our use of the term 'strategy' originates from the work on programmable rewriting strategies for term rewriting à la Stratego [30,40,38]. Strategic pro-

⋆ This white paper served as an invited position paper for the 4th International Workshop on *Reduction Strategies in Rewriting and Programming* (WRS 2004), June 2, 2004, Aachen, Germany. The paper was presented at the WRS 2004 round table "*Strategies in programming languages today*".

doi:10.1016/j.entcs.2004.11.021

grammers can separate basic rewrite steps from the overall scheme of traversal and evaluation. These schemes are programmable by themselves! There are one-layer traversal primitives that facilitate the definition of whatever recursion pattern for traversal. There are further, perhaps less surprising, basic combinators for controlling the evaluation in terms of the order of steps, the choices to be made, the fixpoints to be computed, and others. An extended exposition of what we call 'strategic programming' can be found in [24].

Related forms of programmable strategies permeate computer science. For instance, evaluation strategies without any traversal control are useful on their own in rewriting [7,4]. In theorem proving, one uses a sort of strategies as proof tactics and tacticals [33]. In parallel functional programming, one uses a sort of strategies to synthesise parallel programs [36].

2 Functional strategies in *Strafunski*

The *Strafunski* project [1,27,19,25,26,28] incarnated programmable rewriting strategies for functional programming, namely for Haskell. Strategies are essentially polymorphic functions on datatypes (or 'term types'). The basic rewrite steps are readily specified as monomorphic functions on datatypes. For instance, the following rewrite step encodes some sort of constant elimination for arithmetic expressions:

```
const_elim  :: Expr -> Maybe Expr
const_elim  ((Const 0) `Plus` x)  =  Just x
const_elim  _                     =  Nothing
```

In concrete syntax, and without Haskellish noise, this reads as "`0 + x -> x`". In the example, we wrap the result of the rewrite step in the `Maybe` monad, which allows us to observe success vs. failure of a rewrite step. We can use stacked monads (rather than just `Maybe`) in rewrite steps and strategies. This allows us to deal with state, environment, nondeterminism, and backtracking.

In *Strafunski*, there are two (monadic) types of strategies:

- `TP` — type-preserving strategies: domain and co-domain coincide.
- `TU` — type-unifying strategies: all datatypes are mapped to one result type.

Strafunski's strategy library is based on primitive strategy combinators:

- `idTP` — the identity function.
- `failTP` — the always failing strategy.
- `adhocTP` — update strategy in one type.
- `seqTP` — sequential composition.
- `choiceTP` — left-biased choice.