Teaching the Construction of Domain Specific Languages

Pieter Koopman
Rinus Plasmeijer

the context of Teaching the Construction of Domain Specific Languages

- we have a master course advanced functional programming
- topics:
  - generic programming
  - GADTs
  - monads
  - iTask
  - testing (QuickCheck / Gast)
  - Language semantics and implementation, ..
- department requires a focus on Domain Specific Languages
  - embedding a DSL in a FPL is well known
  - how does this effects our course?

new organisation of advanced functional programming

- use one running example for DSL implementation strategies

- stress the DSL aspect in other topics
  - task oriented programming: iTask
  - model-based testing: Gast

the context of Teaching the Construction of Domain Specific Languages

- department requires a focus on Domain Specific Languages
  - shallow and deep embedding of DSLs
  - While from Nielson & Nielson as running example
  - introduce tooling to optimise the implementation of While
    - generic programming, monads, GADTs, ..
    - the DSLs give us an excellent motivation for introducing these techniques
  - apply those techniques to serious DSLs
    - iTask, Gast, ..
- this improves our course
  - we teach the interesting topics in FP with a better motivation

deep embedding: representation of expressions from while

the grammar

dot to avoid name conflicts

priority should be fixed by additional grammar rules

$$
\begin{array}{lll}
a & & \\
= & n & \text{// number} \\
| & v & \text{// variable} \\
| & a + a & \\
| & a - a & \\
| & a * a &
\end{array}
$$

the data type

infix constructor with binding power

```
:: AExpr
   = Int Int
   | Var Var
   | (+.)   infixl 6 AExpr AExpr
   | (-.)   infixl 6 AExpr AExpr
   | (*.)   infixl 7 AExpr AExpr
:: Var :== String
```


deep embedding: semantics of arithmetic expressions

Scott brackets

$$
\begin{array}{l}
\mathcal{A} : a \rightarrow State \rightarrow Number \\
\mathcal{A}\ [[n]]\ s = \mathcal{N}\ [[n]] \\
\mathcal{A}\ [[v]]\ s = s\ v \\
\mathcal{A}\ [[a_1{+}a_2]]s = \mathcal{A}\ [[a_1]]\ s + \mathcal{A}\ [[a_2]]\ s \\
\mathcal{A}\ [[a_1{-}a_2]]\ s = \mathcal{A}\ [[a_1]]\ s - \mathcal{A}\ [[a_2]]\ s \\
\mathcal{A}\ [[a_1{*}a_2]]s = \mathcal{A}\ [[a_1]]\ s \times \mathcal{A}\ [[a_2]]\ s
\end{array}
$$

Clean

```
A :: AExpr State → Int
A (Int  n) s = n
A (Var v) s = s v
A (x +. y)s = A x s + A y s
A (x -. y) s = A x s -  A y s
A (x *. y) s = A x s * A y s
```

see Nielson & Nielson 1992
only the syntax is improved

deep embedding of while

take home message

- sd

- asas
- asas

adsd

- sdsds
- sads

asas

- sdsd