Генерация зависимых языков по спецификации пользователя

Выступающий: Гарифуллин Шамиль Раифович
Научный руководитель: Исаев Валерий Иванович

СПбАУ

5 июня 2017 г.

Введение

Языки с зависимыми типами — типы могут зависеть от термов. Одна из частых ошибок при программировнии на языке Haskell — взятие первого элемента пустого списка.

```
head :: [a] -> a
head (x:_) = x
head [] = error "No head!"
```

В языке с зависимыми типами мы можем усилить ограничения на входные данные функции.

```
head :: {n : N} -> Vec a (suc n) -> a
head (x:_) = x
```

Реализация языков с зависимыми типами

- При написании функции проверки типов нужно уметь вычислять выражения языка.
  - Можно ли применить функцию takes_only_fib_list к [1, 3] ++ [1, 2, 3]?
  - $2 + 3$ *in fibonacci* == *True*? Зависит от определений $+$, *in* и *fibonacci*
  - *fibonacci* = [1, 1, 2, 3, 5, 8, ...]
- Требуются следующие операции над выражениями: проверка на равенство, подстановка в переменную и абстракция
- Схема алгоритма проверки всегда одна и та же — есть возможность кодогенерации

Цели и задачи

Цель: Реализовать алгоритм генерации функции вычисления и проверки типов зависимых языков по спецификации в виде модуля на Haskell.

Задачи:

- Разработка языка спецификации и налагемых им ограничений
- Выбор внутреннего представления АСД и генерация структур данных конструкций языка
- Генерация кода функций проверки типов и вычисления термов

Язык спецификаций

Необходимо определить язык спецификации и ограничить выразимые языки для дальнейшей генерации.

Функциональные языки программирования состоят из:

- Конструкций (true, false, bool, if)
- Правил типизации конструкций (правил вывода)
- Правил вычисления (редукций)

Формальное определение языка Bool

$$\frac{}{\vdash} \qquad \frac{\Gamma \vdash A}{\Gamma, x : A \vdash}, x \notin \Gamma \qquad \frac{\Gamma \vdash}{\Gamma \vdash x : A}, x : A \in \Gamma$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash a : B}$$

$$\frac{}{\Gamma \vdash \mathit{Bool}} \qquad \frac{}{\Gamma \vdash \mathit{True} : \mathit{Bool}} \qquad \frac{}{\Gamma \vdash \mathit{False} : \mathit{Bool}}$$

$$\frac{\Gamma, x : \mathit{Bool} \vdash T \qquad \Gamma \vdash t : \mathit{Bool} \qquad \Gamma \vdash a : T[x := \mathit{True}] \qquad \Gamma \vdash b : T[x := \mathit{False}]}{\Gamma \vdash \mathit{if}(t, T, a, b) : T[x := t]}$$

Неявные предположения формального определения

$$\frac{\Gamma, x : \mathit{Bool} \vdash T \qquad \Gamma \vdash t : \mathit{Bool} \qquad \Gamma \vdash a : T[x := \mathit{True}] \qquad \Gamma \vdash b : T[x := \mathit{False}]}{\Gamma \vdash \mathit{if}(t, T, a, b) : T[x := t]}$$

- Все выражения делятся на два вида: термы и типы
- Конструкция if принимает аргументы вида (терм, тип, терм, терм)
- Контекст типа T шире, чем стандартный контекст Γ

Типизация спецификации языка Bool

```
DependentSorts:
  tm, ty
Constructs:
  if: (tm,0)*(ty,1)*(tm,0)*(tm,0) -> tm
  bool: ty
  true : tm
  false : tm
```

Спецификация правила вывода конструкции if

$$\frac{\Gamma, x : \mathit{Bool} \vdash T \qquad \Gamma \vdash t : \mathit{Bool} \qquad \Gamma \vdash a : T[x := \mathit{True}] \qquad \Gamma \vdash b : T[x := \mathit{False}]}{\Gamma \vdash \mathit{if}(t, T, a, b) : T[x := t]}$$

Все переменные обозначающие выражения языка который мы определяем — называются *метапеременными*

```
forall x.T : ty, t : tm, a : tm, b : tm
  |- t : bool, x : bool |- T def,
  |- a : T[x:=true], |- b : T[x:=false]
  |-----------------------------------
  |- if(t, x.T, a, b) : T[x:=t]
```

Вывод типов на примере конструкции if

$$\frac{\Gamma, x : Bool \vdash T \qquad \Gamma \vdash t : Bool \qquad \Gamma \vdash a : T[x := True] \qquad \Gamma \vdash b : T[x := False]}{\Gamma \vdash if(t, T, a, b) : T[x := t]}$$

1. На вход подается контекст и $if(t, T, a, b)$
2. Расширяем контекст типом *bool* и проверяем определенность типа T
3. Проверяем, что тип t совпадает с типом *bool*
4. Выводим тип a и проверяем, что его н.ф. совпадает с н.ф. $T[x := true]$
5. Выводим тип b и проверяем, что его н.ф. совпадает с н.ф. $T[x := false]$
6. Возвращаем тип выражения $T[x := t]$

Выражение в *нормальной форме* — выражение, к которому нельзя применить редукций

Ограничения накладываемые языком спецификации

- Все метапеременные передаются в виде аргументов конструкции
- Конструкция в заключении, возвращающая терм, должна быть проаннотирована типом
- Запрещены равенства в заключении, для этого используются редукции
- Одно правило введения на каждую конструкцию языка
- Запрещено перекрытие переменных в контексте

Реализация: представление АСД языка

- От представления языка требуются возможности:
  - Проверки выражений на равенство
  - Подстановки в переменные и абстракции
  - Сопоставление с образцом для нормализации выражений
- Все эти задачи, кроме последней, решает библиотека bound
- bound использует обобщённые индексы де Брейна
- Можно упростить — использовать обычные индексы

Реализация: алгоритм

1. Проходит проверка корректности спецификации
2. Генерируется представление выражений языка при помощи упрощённой библиотеки bound
3. Генерируются операции подстановки, абстракции и проверки на равенство выражений
4. Генерируются функции вывода типов и приведения в нормальную форму

Результаты

Реализована генерация алгоритма проверки типов и вычислителя зависимых языков по спецификации.

- Спроектирован типизированный язык спецификации
- Генерация структуры данных выражений языка и операций над ними
- Генерация кода функций вывода типов термов и функции нормализации специфицированного языка

Репозиторий проекта:
`https://github.com/esengie/fpl-exploration-tool/`