Haskell:
специфікація і реалізація

Специфікація і реалізація

1. Специфікація Haskell
   a. Загальна інформація
   b. Лексична структура
   c. Вирази
2. Модуль Prelude
   a. Структура
   b. Базові типи і функції

Специфікація Haskell

Загальна інформація

Haskell is a general purpose, purely functional programming language	Haskell — це суто функціональна мова програмування загального призначення.
Haskell provides higher-order functions, non-strict semantics, static polymorphic typing, user-defined algebraic datatypes, pattern-matching, list comprehensions, a module system, a monadic I/O system, and a rich set of primitive datatypes, including lists, arrays, arbitrary and fixed precision integers, and floating-point numbers.	Haskell забезпечує функції вищого порядку, несувору семантику, статичну поліморфну типізацію, визначені користувачем алгебраїчні типи даних, зіставлення шаблонів, генерування списків, модульну систему, монадну систему вводу-виводу та багатий набір примітивних типів даних, включаючи списки , масиви, цілі числа довільної та фіксованої точності та числа з плаваючою комою.

Специфікація Haskell

Загальна інформація

Визначено синтаксис програм на Haskell і неформальну абстрактну семантику для значення таких програм.

Залежними від реалізації залишаються способи маніпулювання, інтерпретації, компіляції програм Haskell тощо.

Це включає такі проблеми, як природа середовища програмування та повідомлення про помилки, що повертаються для невизначених програм (тобто програм, які формально оцінюють $\perp$).

Специфікація Haskell

Загальна інформація

Структура програми

Специфікація Haskell

Загальна інформація

Структура програми

Ядро Haskell

- використовується багато зручних синтаксичних структур, які стали популярними у функціональному програмуванні. Такий синтаксичний цукор надається шляхом перекладу на простіші конструкції.
- результатом перекладу буде програма, написана на невеликій підмножині Haskell, яку називають ядром Haskell.
- ядро формально не визначено, але це, по суті, злегка підсолоджений варіант лямбда-числення з прямою денотаційною семантикою.

Специфікація Haskell

Загальна інформація

Структура програми
Ядро Haskell
Значення та типи

- Вираз обчислюється як значення та має статичний тип. Значення та типи не змішуються в Haskell.
- Помилки в Haskell семантично еквівалентні ⊥ (“bottom”, “основа”). Технічно їх неможливо відрізнити від незавершення, тому мова не містить механізму для виявлення помилок або дій після них. Однак реалізації, ймовірно, намагатимуться надати корисну інформацію про помилки.

Специфікація Haskell

У Haskell є шість типів імен:
для змінних і конструкторів - позначають значення;
для змінних типу, конструкторів типів і класів типів посилаються на сутності, пов'язані з системою типів;
і назви модулів посилаються на модулі.

Є два обмеження щодо іменування:
Назви змінних і змінних типу є ідентифікаторами, які починаються з малих літер або підкреслення;
інші чотири типи імен є ідентифікаторами, що починаються з великих літер.
Ідентифікатор не можна використовувати як назву конструктора типу та класу в одній області.
Це єдині обмеження; наприклад, Int може одночасно бути назвою модуля, класу та конструктора в одній області.

Лексична структура

1. Нотація (БНФ-подібна)

pat$_{\langle \textbf{pat}' \rangle}$ difference-elements generated by **pat** except those generated by **pat′**

q - qualified
id - identifier
var - variable
con - constructor
sym - symbol
ty - type
mod - module
cls - class

Вирази

Всі імена посилаються на Prelude

Є неасоціативні оператори

Заперечення є єдиним префіксним оператором у Haskell

Граматика неоднозначна щодо обсягу лямбда-абстракцій, виразів let і умовних виразів.

Вирази

Всі імена посилаються на Prelude

Є неасоціативні оператори

Заперечення є єдиним префіксним оператором у Haskell

Граматика неоднозначна щодо обсягу лямбда-абстракцій, виразів let і умовних виразів. Неоднозначність вирішується мета-правилом, згідно з яким кожна з цих конструкцій простягається якомога далі вправо.

Вирази

3 Expressions
- 3.1 Errors
- 3.2 Variables, Constructors, Operators, and Literals
- 3.3 Curried Applications and Lambda Abstractions
- 3.4 Operator Applications
- 3.5 Sections
- 3.6 Conditionals
- 3.7 Lists
- 3.8 Tuples
- 3.9 Unit Expressions and Parenthesized Expressions
- 3.10 Arithmetic Sequences
- 3.11 List Comprehensions
- 3.12 Let Expressions
- *3.13 Case Expressions*
- *3.14 Do Expressions*
- *3.15 Datatypes with Field Labels*
- *3.16 Expression Type-Signatures*
- *3.17 Pattern Matching*

Модуль Prelude

a. Структура
b. Базові типи і функції

Contents

Модуль Prelude