Лабораторная работа #1 Вычисления в Isabelle

Данный материал основан на:

- Книге Тобиаса Нипкоу, Лоуренса Паулсон и Маркуса Уэйнзель «Isabelle/HOL: A Proof Assistant for Higher Logic»
- Материалах сайта, посвященного Isabelle http://isabelle.in.tum.de/
- Книге Кэннета Роузен «Discrete Mathematics and Its Applications»

Подборку материалов их компиляцию и адаптацию к курсу формальных методов выполнили:

- Якимов И.А. ivan.yakimov.research@yandex.ru
- Кузнецов А.С. askuznetsov@sfu-kras.ru

На лекции вы ознакомились с системой типов в Isabelle. Она должна быть вам знакома по работе с Haskell, так как язык Isabelle/ML очень похож на него.

Типы и термы

Термы это примерно тоже, что и выражения в императивных языках программирования.
Чтобы объявить терм в Isabelle можно использовать ключевое слово **term**:
Isabelle автоматически выводит типы термов. Давайте посмотрим, как это работает на примере (тип выражения выводится на экран в дополнительном экране редактора Isabelle/Jedit).

Терм	Тип	Комментарий
term "x"	"'a"	Так как с **x** *не* связан конкретный тип, то в качестве типа выступает переменная **'a** (переменная типа, вместо которой можно подставить любой конкретный тип).
term "x::nat"	"nat"	Тип задан явно при помощи конструкции **::nat** — можно сказать, что вместо переменной типа **'a** из предыдущего примера был подставлен конкретный тип **nat**.
term "x + 1"	"'a"	Сложение является полиморфной функцией, и если тип **x** явно не задан, то мы предполагаем что она результат может быть целым числом, рациональной дробью и так далее — иметь любой тип, для которого определена операция сложения.
term "hd"	"'a list ⇒ 'a"	Функция **hd** (возвращающая первый элемент списка) является отображением (то есть функцией) списков на единичные элементы.
term "hd [1::nat,2]"	"'a"	Так как к функции **hd** уже применен аргумент в виде списка [1::nat,2], то она возвращает константу типа **nat**. Можете ли вы сказать зачем мы использовали [1::nat,2], а не просто [1,2]? Есть ли разница между двумя этими выражениями и если есть, то в чем?

Прим.: Комментарий в Isabelle заключается в скобки со звездочками:
(* текст комментария *)
Составные части терма должны быть одного и того же типа, то есть термы должны быть *хорошо типизированными*. Ниже приведен пример *плохо типизированных* термов.

Терм	Ошибка	Комментарий
term "x::nat = True"	Inner syntax error⌂ Failed to parse term	Мы не можем сравнивать натуральные числа и булевы значения, так как они принадлежат к различным доменам — nat = {1,2,...} и bool = {true, false}, соответственно.

term "x::nat ≠ True"	Inner syntax error⌂ Failed to parse term	Обратите внимание, в C++ и многих других языках программирования, мы можем сравнивать целые числа и булевы значения, при этом не происходит ошибки компиляции и результат в некоторых случаях может быть **true**!
		Можете ли вы сказать почему так происходит?
		
		#include <iostream>
		int main() {
		int x = 0;
		bool y = false;
		if (x == y)
		std::cerr << "we can compare int and bool in C++" << std::endl;
		}
		
		g++ example.cpp && ./a.out
		we can compare int and bool in C++

Таблица — некоторые символы. Сначала введите символ, после того как появится подсказка — нажмите на TAB и Jedit вставит нужный x-символ.

ASCII-код или аббревиатура	X-символ	Комментарий
~	¬	Логическое НЕ
/\	∧	Логическое И
\/	∨	Логическое ИЛИ
<\forall>	∀	Квантор всеобщности
<\exists>	∃	Квантор существования
==	≡	Эквивалентность

Вычисления в Isabelle/ML

Вычисления в функциональных языках могут быть реализованы при помощи техники под названием *редукция графа*. Рассмотрим данный процесс на простом примере.

Пусть у нас есть функция:

$$f\ x = (x - 2) * (x + 3)$$

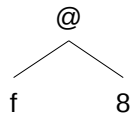
По определению функция f с одним аргументом x вычисляет значение выражения

'(x — 2) * (x + 3)'.

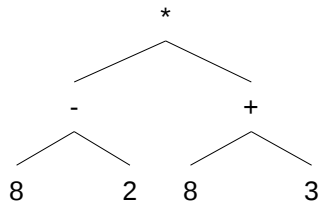
Предположим что мы вызвали

f 8

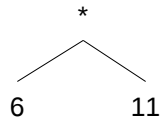
То есть применили функцию f к аргументу 8. Мы можем представить нашу программу следующим образом:



Где @ обозначает функциональную аппликацию (применение, или «вызов» функции). Результат применение аргумента 8 к функции f дает:



Теперь мы можем вычислить вычитание '8 — 2' и сложение '8 + 3' *в любом порядке*. В результате получим:



На последнем шаге, умножение дает:

66

Из данного примера видно, что:

- Выполнение функциональной программы состоит из вычисления значения выражения
- Естественным представлением функциональной программы является *дерево* (или, в общем случае, *граф*)
- Вычисление состоит из последовательности простых шагов, называемых *редукциями*. Каждая редукция приводит к локальной трансформации графа (то есть к *редукции графа*) *Прим.: мы с вами уже познакомились с бета-редукцией в лекции.*
- Редукции подвыражений можно *безопасно* проводить в разном порядке, а именно *параллельно*, так как они не могут пересекаться друг с другом.

- Вычисление заканчивается тогда, когда не остается редуцируемых выражений.

Провести редукцию выражения в Isabelle/HOL можно при помощи команды **value**.
Рассмотрим примеры:

Выражение	Результат вычислений	Комментарий
value "2 + 2"	"1 + 1 + (1 + 1)" :: "'a"	Константа 2 на самом деле определена через сумму двух единиц, вычисление выражения 2+2 приводит к выводу на экран "1+1+(1+1)", дальнейшей редукции не происходит.
value "1 * 3"	"1 * (1 + 1 + 1)" :: "'a"	Константа 3 определена через сумму трех единиц. Вычисление 1*3 приводит к выводу на экран "1 * (1 + 1 + 1)", дальнейшей редукции не происходит.

Натуральные числа и натуральный ряд

Натуральные числа (тип **nat**) построены индуктивно по двум простым принципам:

- Ноль является натуральным числом*
- Число, следующее за натуральным, также является натуральными

*Обычно в натуральный ряд *не* включается ноль и N = {1,2,...}, а $N^0 = Z^+ = \{0,1,2,...\}$ определяется как множество неотрицательных целых.

Посмотрим на (упрощенное) определение натуральных чисел в Isabelle.

Ввод	Комментарий
datatype nat = Zero | Suc nat	!Ключевое слово **datatype** используется для определения *алгебраических типов данных* (определенных через другие типы, включая самих себя). Определение nat говорит о том, что объект данного типа может иметь значение Zero либо значение, сконструированное из другого значения типа nat добавлением к нему слева конструктора Suc. Пример: Zero = 0 Suc Zero = 1 Suc (Suc Zero) = 2, и так далее О конструкторе Suc можно думать как об операции прибавления единицы, в нашем примере: Zero = 0 Suc Zero = 1 + 0 = 1

	Suc (Suc Zero) = 1 + (1 + 0) = 2

Продвинутый материал — мотивация индуктивного определения nat через 0 и Suc.

Натуральные числа определены таким образом, чтобы для них можно было относительно просто задать основные арифметические операции такие как сложение, умножение и так далее, а также доказать основные алгебраические свойства данных операций.

Ввод	Комментарий
primrec add :: "nat ⇒ nat ⇒ nat" where "add x 0 = x" | "add x (Suc n) = Suc (add x n)"	Здесь вы видите как определена операция сложения над натуральными числами. Вспомним как мы определили натуральное число: • Ноль — натуральное число • Число следующее за натуральным также натуральное
Для задания операции сложения нам нужно рассмотреть эти два случая. 1) "add x 0 = x" — если прибавить к любому числу ноль, получится это же число: 1 + 0 = 1 42 + 0 = 42 2) "add x (Suc n) = Suc (add x n)" — здесь немного сложнее, но тоже достаточно просто. Вспомним что такое сложение — x + n означает взять x и прибавить к нему n единиц. Пусть, например x = 2, n = 3, тогда x + n = 2 + 3 = 2 + (1 + 1 + 1) = 5. Нужно отметить, что x мы оставляем без изменений, поэтому при вызове функции сложения x передаем как есть. В тоже самое время, нам нужно сосчитать, сколько раз мы прибавили 1, чего можно добиться, используя n как счетчик. Также часто используют метафору «перетаскивания» единицы справа налево на каждом вызове. Рассмотрим пример на конкретных значениях: 2 + 3 add 2 3 = add 2 (2 + 1) = 1 + (add 2 2) add 2 2 = add 2 (1 + 1) = 1 + (add 2 1) add 2 1 = add 2 (0 + 1) = 1 + (add 2 0) add 2 0 = 2 Если свернуть все промежуточные значения, мы получим «1 + 1 + 1 + 2», что тоже что и «2 + 1 + 1 + 1» и «2 + 3» и «5». !Резонный вопрос может заключаться в том, действительно ли мы нашли сумму 2 и 3? Ведь вывод на экран результата в виде «1 + 1 + 1 + (1 + 1)» вряд ли удовлетворит кого-нибудь. Но на самом деле, все это разные представления числа 5. Чтобы выразить результат в десятичной системе счисления, нужно просто написать соответствующую функцию.	

Лямбда-исчисление

Высокоуровненые функциональные программы (написанные на Haskell, ML и так далее) транслируются в нотацию лямбда-счисления, которая далее транслируется в конкретную реализацию (байткод, ассемблер и так далее). Грубо говоря, лямбда-исчисление является своеобразным «ассемблером»для функциональных языков. Мы также можем в явном виде использовать лямбда-функции в коде высокоуровневых программ там, где это необходимо.

Рассмотрим определение функции инкремента **λn. n + 1** по аналогии с языком Си++

*	Заголовок	Тело функции	Вызов
Isabelle/ML	λn.	.n + 1	(λn. n + 1) 2
C++11	[] (int n)	{return n + 1;}	[](int n){return n+1;}(2)

Для того, чтобы произвести вычисления мы можем воспользоваться **командой** value

Ввод	Вывод	Комментарий
value "(λn. n + 1) 2::nat"	"Suc (Suc (Suc 0))" :: "nat"	Результат вычислений, как и ожидалось равен трем.

Обсуждение

Вы ознакомились с базовыми возможностями Isabelle/HOL а именно:

- Получили начальные сведения о системе типов, правилах вывода типов.
- Ознакомились с процессом вычислений в Isabelle/ML
- Ознакомились с типом **nat**, задающим натуральные числа
- Разобрали пример использования лямбда-функций

Как выбрать вариант?

Ваш вариант формируется бинарным кодом. Выбираете код из списка ниже (номер кода в списке соответствует вашему номеру в журнале). Далее выбираете для каждого задания один из псевдо-вариантов, проходя бинарный код *слева-направо*:

1. 111100
2. 110101
3. 010101
4. 100111
5. 000001
6. 100010
7. 110101
8. 101001
9. 000101
10. 000110
11. 001010

12. 000110
13. 110110
14. 101011
15. 010111
16. 011100
17. 110000
18. 000001
19. 111001
20. 010011

Задания

Базовые задания

1. Для указанных выражений нужно написать термы, при помощи команды **term**, указать их тип, сопроводив нужным комментарием #1.1.Типы и термы|outline
2. Для указанных выражений нужно проверить, являются ли они хорошо типизированными, если нет, то пояснить почему #1.1.Типы и термы|outline
3. Найти указанных значений выражений при помощи команды **value** #1.2.Вычисления в Isabelle/ML|outline
4. Представить выражение в виде дерева #1.2.Вычисления в Isabelle/ML|outline
5. Записать указанные числа через 0 и Suc, вычислить значения составленных из них выражений при помощи **value** #1.3.Натуральные числа и натуральный ряд|outline
6. Записать описанные функции в виде лямбда-функций, вычислить их значения для указанных аргументов #1.4.Лямбда-исчисление|outline

Задание для указанных в вариантах выражений	**Псевдо-вариант 0**	**Псевдо-вариант 1**
1) Объявить термы при помощи команды **term**, указать их тип, сопроводить ответ нужными комментариями #1.1.Типы и термы|outline	переменная x типа bool; логическое отрицание x.	переменная x типа nat; 2 + 2, где оба числа — nat.
2) Проверить, являются ли данные выражения хорошо типизированными, если нет,	логическое отрицание нуля; деление переменной x на ноль.	логическое И нуля и единицы; декремент переменной x.

то почему? Имеют ли они смысл? #1.1.Типы и термы| outline		
3) Найти значения выражений при помощи команды **value**. Что служит результатом, почему? Каков тип результата? Можно ли продолжить вычисления? #1.2.Вычисления в Isabelle/ML|outline	Разность 42 и 28; Тоже выражение, но теперь обе константы имеют тип **nat**.	Сумма 13 и 7; Тоже выражение, но теперь обе константы имеют тип **nat**;
4) Представить выражения в виде дерева. Можно ли провести вычисления параллельно? Если да, раскрасьте поддеревья с независимыми подвыражениями в разные цвета. Как бы вы описали алгоритм выбора поддеревьев для параллельных вычислений? #1.2.Вычисления в Isabelle/ML|outline	((x + 2) * (x - 2)) / 3; f 16.	(17 - (p + 42)) / (q - 28); g 20.
5) Записать указанные числа через 0 и Suc, вычислить значения составленных из них выражений при помощи **value** #1.3.Натуральные числа и натуральный ряд|outline	a = 2, b = 3: • a + b • a * b	a = 6, b = 2: • a - b • a + b
6) Записать выражения в виде лямбда-функций, вычислить их значения для указанных аргументов при помощи **value**. Что служит результатами, почему? Каков тип результатов? #1.4.Лямбда-исчисление|	f (x) = x * 2 • f (3), где 3 — nat • f (w) • f (u-1)	g(y) = y + y • g (2), где 2 — nat • g (p) • g (r+1)

outline		

Источники и материалы для дополнительного чтения

- Richard S. Bird, Philip Wadler. Introduction to Functional Programming, 1st edition, 1998
- Tobias Nipkow, Lawrence C. Paulson, Markus Wenzel. A Proof Assistant for Higher-Order Logic, 2016
- Аксиомы Пеано, здесь же вы найдене рекурсивное определение суммы для натуральных чисел, Википедия - https://en.wikipedia.org/wiki/Peano_axioms