МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное
учреждение высшего образования
ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт математики, механики и компьютерных наук
имени И. И. Воровича

Направление подготовки
Прикладная математика и информатика

Кафедра информатики и вычислительного эксперимента

СИСТЕМА ВЫВОДА В ЛИНЕЙНОЙ ЛОГИКЕ С РЕАЛИЗАЦИЕЙ НА ЯЗЫКЕ HASKELL

Выпускная квалификационная работа
на степень бакалавра

Студента 4 курса
В. А. Панкова

Научный руководитель:
асс. каф. ИВЭ А. М. Пеленицын

Ростов-на-Дону
2015

Содержание

Введение

В данной работе был реализован логический вывод формул линейной логики на языке программирования Haskell. Язык Haskell позволил описать вывод на «естественном языке» без использования условных операторов, посредством указания каждой логической связке, каким образом её необходимо преобразовывать.

Ключевыми моментами в реализации программы являются

- Представление выражение в стиле *Data types à la carte* [1]
- Расширенное перекрытие классов типов (англ. Advanced Overlapping) [2]

Основным источником ссылок по теории линейной логики является учебное пособие «Линейная Логика» Марка Феннинга [3]

1. Предварительные сведения

1.1. Система вывода

Определение (Система вывода). *Система вывода [4] (англ. Deductive System, Inference System) определяется*

1. *Набором суждений*
2. *Набором переходов вида*

$$\frac{J_1 \dots J_n}{J}, \text{ где } J_k \text{ — суждения;}$$

J *— переменная;*

если $n = 0$*, то переход называют аксиомой.* $\diamondsuit$

Определение (Правило вывода). *Правилом вывода называется переход, в котором все суждения являются переменными.* ♢

Утверждение 1.1: *Правила вывода позволяются схематично описать все переходы.* □

Определение (Дерево доказательства). *Деревом доказательства называется граф, в котором узлы представлены переходами, а дуги — суждениями.* ♢

$$\dfrac{\dfrac{J_1 \qquad J_2}{J_3} \qquad \dfrac{}{J_4}}{J_5}$$

Рисунок 1 — Пример дерева доказательств

Определение (Теорема). *Теоремой (или доказательством) называется дерево доказательств, в котором все листья являются аксиомами.* ♢

1.2. Исчисление секвенций

Определение (Исчисление секвенций). *Исчисление секвенций [5] — это система вывода для логики предикатов (или подобной ей), в котором основным суждением является секвенция:*

$$A_1, \ldots, A_n \vdash B_1 \ldots B_m.$$

Подразумевается, что гипотезы $B_1 \ldots B_m$ становятся истинными, если истинны гипотезы $A_1, \ldots, A_n$. ♢

Базовыми правилами исчесления секвенций являетя *правило инициализации*, которое позволяет сопоставить *атомарные суждения*. Будем называть это правило *init*.

$$\frac{}{A \vdash A}\ \text{init}$$

Рисунок 2 — Правило *init*

Определение. *Суждение называется атомарным, если оно не состоит из других суждений.* ◊

1.3. Исчисление секвенций для линейной логики

1.3.1. Основы линейной логики

Линейная логика [6] основывается на ограничении использования гипотез: всякую гипотезу в процессе вывода разрешается использовать один раз. По-этому гипотезы в линейной логике принято называть ресурсами. Существуют два правила, которые контролируют выполнение этого свойства:

Правило введения гипотезы Имея ресурс A, можем доказать A

$$\frac{}{u : A \vdash A}\ u$$

Рисунок 3 — Правило введения гипотезы

Принцип подстановки Если $\Delta \vdash A$ и $\Delta', A \vdash C$ истинны, то $\Delta, \Delta' \vdash C$ тоже истинно.

Правило введения гипотезы гарантирует, что ресурсы будут использованы. *Принцип подстановки* гарантирует, что ресурс не будет использован более одного раза.

Эти свойства формируют *базовое утверждение в линейной логике* (англ. *Linear Hypothetical Judgment*). Оно называется *линейным гипотетическим утверждением*. Будем сокращенно называть его *ЛГС*.

Утверждение 1.2 (Линейной гипотетическое утверждение):
Имея ресурсы $A_1 \dots A_n$, можно получить ресурс G:

$$A_1 : u_1, A_2 : u_2, \ldots, A_n : u_n \vdash G$$ □

Рисунок 4 — Линейной гипотетическое утверждение

Переменные u_k обозначают имена ресурсов. В утверждение могут быть одинаковые ресурсы, однако их имена должны быть различными. Далее имена будут опускаться если в них не будет необходимости.

Помимо ресурсов, допускаются гипотезы, которые можно использовать неограниченное количество раз [7]. Такие гипотезы называются неограниченными или общезначимыми (англ. *Valid*).

Определение (Общезначимые гипотезы, 1). *Гипотеза A называется общезначимой, если для её доказательства не требуется других гипотез:*

$$\cdot \vdash A$$

Символ $\cdot$ означает отсутствие ресурсов слева от знака секвенции. ♢

Определение (Общезначимые гипотезы, 2). *Гипотеза A называется общезначимой, если существует такой набор общезначимых гипотез Γ, что*

$$\Gamma \vdash A$$ ♢

Общее гипотетическое суждение с присутствием общезначимых гипотез переписывается следующим образом:

Утверждение 1.3 (ЛГС с общезначимыми гипотезами): *Имея общезначимые гипотезы $B_1, B_2, \ldots, B_m$ и ресурсы $A_1 \ldots A_n$, можно получить ресурс G:*

$$B_1 : v_1, B_2 : v_2, \ldots, B_m : v_m; A_1 : u_1, A_2 : u_2, \ldots, A_n : u_n \vdash G \qquad \square$$

Рисунок 5 — Линейной гипотетическое суждение с общезначимыми гипотезами

Для обозначения набора общезначимых гипотез будем использовать символ Γ, а для ресурсов - Δ. Тогда *ЛГС* сокращенно запишется так:

$$\Gamma; \Delta \vdash G$$

Правило введения гипотезы и *принцип подстановки* с присутствием общезначиных гипотез изменяются следующим образом:

Правило введения общезначимой гипотезы Имея общезначимую гипотезу A, можем доказать A

$$\frac{}{(\Gamma, u : A); \cdot \vdash A}\, u$$

Рисунок 6 — Правило введения общезначимой гипотезы

Принцип подстановки общезначимой гипотезы Если $\Gamma; \cdot \vdash A$ и $(\Gamma, u : A); \Delta \vdash C$ истинны, то и $\Gamma; \Delta \vdash C$ истинно.

1.3.2. Правила вывода в исчислении секвенций для линейной логики

Правило *init* имеет следующий вид в линейной логике [8]:

$$\frac{}{\Gamma; A \vdash A}\, \text{init}$$

Рисунок 7 — Правило init для линейной логики

Наличие общезначимых гипотез порождает дополнительное правило, позволяющее копировать гипотезы из Γ в Δ:

$$\frac{(\Gamma, v : A); (\Delta, u : A) \vdash C}{(\Gamma, v : A); \Delta \vdash C}\, copy_v$$

Рисунок 8 — Правило копирования общезначимых гипотез

Замечание: В данном примере указаны имена гипотез и ресурсов для конкретности. △

В исчислении секвенций к логическим связкам применяются *правые* и *левые* правила. Они аналогичны правилам *введения* и *удаления* в естественном выводе. Отличие заключается в том, что вывод, построенный с помощью *правых* и *левых* правил всегда будет нормальным.

Определение. *Вывод называется* нормальным*, если он был получен посредством применения правил введения к заключению и правил удаления к посылкам доказываемой секвенции.* ♢

В линейной логике есть одиннадцать логических связок (или формул).

1. *Одновременная конъюнкция* (англ. *Simultanious Conjuction*). Обозначается символом $\otimes$ и означает истинность двух суждений в один и тот же момент времени.

$$\frac{\Gamma; \Delta' \vdash A \qquad \Gamma; \Delta'' \vdash B}{\Gamma; \Delta', \Delta'' \vdash A \otimes B}\otimes R$$

(a) Правое

$$\frac{\Gamma; \Delta, A, B \vdash C}{\Gamma; \Delta, A \otimes B \vdash C}\otimes L$$

(b) Левое

Рисунок 9 — Правила для одновременной конъюнкции

2. *Альтернативная конъюнкция* (англ. *Alternative Conjuction*). Обозначается символом $\&$ и означает одновременную истинность двух суждений с возможностью выбора лишь одного из них.

$$\frac{\Gamma; \Delta \vdash A \qquad \Gamma; \Delta \vdash B}{\Gamma; \Delta \vdash A \& B} \,\&R$$

(a) Правое

$$\frac{\Gamma; \Delta, A \vdash C}{\Gamma; \Delta, A \& B \vdash C} \,\&L_1 \qquad\qquad \frac{\Gamma; \Delta, B \vdash C}{\Gamma; \Delta, A \& B \vdash C} \,\&L_2$$

(b) Левые

Рисунок 10 — Правила для альтернативной конъюнкции

3. *Линейная импликация* (англ. *Linear Implication*). Обозначается сиволом $\multimap$ и означает истинность следствия при истинности посылки.

$$\frac{\Gamma; \Delta, A \vdash B}{\Gamma; \Delta \vdash A \multimap B} \multimap R \qquad\qquad \frac{\Gamma; \Delta' \vdash A \qquad \Gamma; \Delta'', B \vdash C}{\Gamma; \Delta', \Delta'', A \multimap B \vdash C} \multimap L$$

(a) Правое (b) Левое

Рисунок 11 — Правила для линейной импликации

4. *Общезначимая импликация* (англ. *Unrestricted Implication*). Обозначается символом $\supset$ и означает возможность получения ресурса-следствия при истинности посылки. Причем посылка является общезначимой гипотезой.

$$\frac{(\Gamma, A); \Delta \vdash B}{\Gamma; \Delta \vdash A \supset B} \supset R \qquad\qquad \frac{\Gamma; \cdot \vdash A \qquad \Gamma; \Delta, B \vdash C}{\Gamma; \Delta, A \supset B \vdash C} \supset L$$

(a) Правое (b) Левое

Рисунок 12 — Правила для общезначимой импликации

5. *Дизъюнкция* (англ. *Disjunction*). Обозначается символом $\oplus$ и означает *исключающее или* двух суждений.

$$\frac{\Gamma; \Delta \vdash A}{\Gamma; \Delta \vdash A \oplus B} \oplus R_1 \qquad \frac{\Gamma; \Delta \vdash B}{\Gamma; \Delta \vdash A \oplus B} \oplus R_2$$

(a) Правые

$$\frac{\Gamma; \Delta, A \vdash C \qquad \Gamma; \Delta, B \vdash C}{\Gamma; \Delta, A \oplus B \vdash C} \oplus L$$

(b) Левое

Рисунок 13 — Правила для дизъюнкции

Замечание: Я называю эту логическую связку *дизъюнкцией* несмотря на то, что она определена как *исключающее или*, ввиду того, что такому подходу следуют в пособии .!.!. △

6. *Коерция* (англ. *Coercion*). Является унарной логической связкой. Обозначается символом ! и означает принадлежность ресурса к общезначимым гипотезам.

$$\frac{\Gamma; \cdot \vdash A}{\Gamma; \cdot \vdash !A} !R$$

(a) Правое

$$\frac{(\Gamma, A); \Delta \vdash C}{\Gamma; (\Delta, !A) \vdash C} !L$$

(b) Левое

Рисунок 14 — Правила для коерции

7. *Потолок* (англ. *Top*). Обозначается символом $\top$. Является константой, для доказательства которой подходят любые ресурсы. Более конкретно, суждение $\top$ поглощает все ресурсы.

$$\frac{}{\Gamma; \Delta \vdash \top} \top R$$

(a) Правое

Левое правило отсутствует, так как $\top$ не хранит информацию о том, какие ресурсы были поглощены для его доказательства.

Рисунок 15 — Правила для $\top$

8. *Единица* (англ. *Unit*). Обозначается символом 1 и означает суждение, для доказательства которого не требуются ресурсы.

$$\frac{}{\Gamma; \cdot \vdash 1}\,1R$$

(a) Правое

$$\frac{\Gamma; \Delta \vdash C}{\Gamma; \Delta, 1 \vdash C}\,1L$$

(b) Левое

Рисунок 16 — Правила для 1

9. *Невозможность* (англ. *Impossibility*). Также называют нулём. Обозначается символом 0. Является суждением, которое нельзя построить или достичь. Определяет невозможность дальнейшего вывода.

Правое правило отсутствует: это суждение нельзя получить в качестве заключения вывода.

$$\frac{}{\Gamma; \Delta, 0 \vdash C}\,0L$$

(a) Левое

Рисунок 17 — Правила для 0

10. *Квантор всеобщности* (англ. *Universal Quantifier*).

$$\frac{\Gamma; \Delta \vdash [a/x]A}{\Gamma; \Delta \vdash \forall x.A}\,\forall R^a$$

(a) Правое

$$\frac{\Gamma; \Delta, [t/x]A \vdash C}{\Gamma; \Delta, \forall x.A \vdash C}\,\forall L$$

(b) Левое

Замечание: Индекс a означает, что имя которое подставляется по x в A, должно быть новым именем в A. $\triangle$

Рисунок 18 — Правила для $\forall$

11. *Квантор существования* (англ. *Existential Quantifier*).

$$\frac{\Gamma; \Delta \vdash [t/x]A}{\Gamma; \Delta \vdash \exists x.A}\,\exists R$$

(a) Правое

$$\frac{\Gamma; \Delta, [a/x]A \vdash C}{\Gamma; \Delta, \exists x.A \vdash C}\,\exists L^a$$

(b) Левое

Рисунок 19 — Правила для $\exists$

2. Алгоритм вывода доказательств фокусировкой

Все в этом разделе [9].

Определение (Небратимое правило вывода). *Правило вывода называют необратимым (англ. non invertible), если его применение к секвенции не создаёт новых секвентов таких, что доказательство одного может зависеть от ресурсов, которые останутся после доказательства другого.* ♢

Определение (Обратимое правило вывода). *Правило вывода называют обратимым (англ. invertible), если оно не является необратимым. А именно, если применение правила к секвенции создаёт новые секвенции, то они не зависят друг от друга.* ♢

Определение (Частично правило вывода). *Правило вывода называют частично обратимым (или слабо обратимым) (англ. weakly invertible), если оно является обратимым при определенном наборе ресурсов.* ♢

Определение (Асинхронная логическая связка). *Логическая связка называется правой / левой асинхронной, если у неё есть правое / левое обратимое правило.* ♢

Определение (Синхронная логическая связка). *Логическая связка называется правой / левой синхронной, если у неё есть правое / левое частично обратимое или необратимое правило.* ♢

Правые асинхронные :	$A \multimap B, A\&B, \top, A \supset B, \forall x.A$
Левые асинхронные :	$A \otimes B, 1, A \oplus B, 0, !A, \exists x.A$
Правые синхронные :	$A \otimes B, 1, A \oplus B, 0, !A, \exists x.A$
Левые синхронные :	$A \multimap B, A\&B, \top, A \supset B, \forall x.A$

Рисунок 20 — Классификация логических связок

Алгоритм фокусировки делится на три фазы:

1. *Инверсия*. Сначала к цели, затем к посылкам доказываемой секвенции применяются обратимые правила.

2. *Решение*. Как только цель и затем посылки перестают быть асинхронными (нельзя применить обратимые правила), принимается решение о том, фокусироваться ли на цели или на посылках.

3. *Фокус*. Фаза фокуса означает фокусировку на цели или на посылке доказываемой секвенции. Фокусировка означает применение частично обратимых и необратимых правил — раскрытие синхронных логических связок.

Алгоритм фокусировки схематично описывается правилами вывода. При этом фазы алгоритма определяются новыми логическими связками:

1. $A \Uparrow$ - инверсия ресурса A;

2. $A \Downarrow$ - фокусировка на ресурсе A;

2.1. Правая инверсия

Некоторые правила разбивают цель на подзадачи, часть из которых попадает в левую часть новой секвенции. В качестве примера рассмотрим правило для *линейной импликации*:

$$\frac{\Gamma; \Delta, A \vdash B}{\Gamma; \Delta \vdash A \multimap B} \multimap R$$

Ресурс A попадает в набор Δ. Значит, в ходе естественного вывода, A должно было быть получено *правилом удаления* и затем использовано *правилом введения* чтобы построить $A \multimap B$. Чтобы отразить этот факт, в процессе левой инверсии ресурс A должен быть обработан в соответствующем порядке. Для этого вводится понятие *упорядоченного ЛГС* (англ. *Ordered Hypothetical Judgment*).

Определение (Упорядоченное ЛГС). *Удорядоченным ЛГС называется секвенция с дополнительным набором упорядоченных линейных ресурсов Ω в левой части:*

$$\Gamma; \Delta; \Omega \vdash G$$

$\diamondsuit$

Замечание: Ввиду этого, для доказательства произвольной секвенции $\Gamma; \Delta \vdash A$, её необходимо преобразовать в $\Gamma; \cdot; \Delta \vdash A$ для того, чтобы начать фазу инверсии. $\triangle$

Следующие правила описывают алгоритм правой инверсии:

$$\frac{\Gamma; \Delta; \Omega \vdash A \Uparrow \quad \Gamma; \Delta; \Omega \vdash B \Uparrow}{\Gamma; \Delta; \Omega \vdash A \& B \Uparrow} \& R \qquad \frac{\Gamma; \Delta; \Omega, A \vdash B \Uparrow}{\Gamma; \Delta; \Omega \vdash A \multimap B \Uparrow} \multimap R$$

$$\frac{(\Gamma, A); \Delta; \Omega \vdash B \Uparrow}{\Gamma; \Delta; \Omega \vdash A \supset B \Uparrow} \supset R \qquad \frac{}{\Gamma; \Delta; \Omega \vdash \top \Uparrow} \top R$$

$$\frac{\Gamma; \Delta; \Omega \vdash [a/x]A \Uparrow}{\Gamma; \Delta; \Omega \vdash \forall x.A \Uparrow} \forall R^a$$

Рисунок 21 — Правила вывода для правой инверсии

Когда цель доказываемой секвенции перестанет быть правой асинхронной, осуществляется переход к левой инверсии, где будут раскрываться упорядоченные ресурсы. Это контролируется дополнительным правилом:

$$\frac{\Gamma; \Delta; \Omega \vdash C \Uparrow \qquad C \text{ не является правой асинхронной}}{\Gamma; \Delta; \Omega \vdash C \Uparrow} \Uparrow R$$

Рисунок 22 — Правило перехода от правой к левой инверсии

2.2. Левая инверсия

В процессе левой инверсии обрабатываются ресурсы из упорядоченного набора Ω посредством применения левых обратимых правил. В случае, если следующий по порядку ресурс в Ω не является левым асинхронным, он просто переносится в набор Δ.

Следующие правила описывают процесс левой инверсии:

$$\frac{\Gamma; \Delta; \Omega, A, B \Uparrow\vdash C}{\Gamma; \Delta; \Omega, A \otimes B \Uparrow\vdash C} \otimes L \qquad \frac{\Gamma; \Delta; \Omega, A \Uparrow\vdash C \qquad \Gamma; \Delta; \Omega, B \Uparrow\vdash C}{\Gamma; \Delta, A \oplus B \Uparrow\vdash C} \oplus L$$

$$\frac{(\Gamma, A); \Delta; \Omega \Uparrow\vdash C}{\Gamma; \Delta; (\Omega, !A) \Uparrow\vdash C} !L \qquad \frac{\Gamma; \Delta; \Omega \Uparrow\vdash C}{\Gamma; \Delta; \Omega, 1 \Uparrow\vdash C} 1L$$

$$\frac{}{\Gamma; \Delta; \Omega, 0 \Uparrow\vdash C} 0L \qquad \frac{\Gamma; \Delta; \Omega, [a/x]A \Uparrow\vdash C}{\Gamma; \Delta; \Omega, \exists x.A \Uparrow\vdash C} \exists L^a$$

Рисунок 23 — Правила вывода для левой инверсии

В наборе Ω могут находиться ресурсы, не являющиеся левыми асинхронными. В этом случае они должны быть перенесены в набор Δ, чтобы затем быть обработанными в *фазе фокуса*. Это контролирует следующие правило:

$$\frac{\Gamma; \Delta, A; \Omega \Uparrow\vdash C \qquad A \text{ не является левой асинхронной}}{\Gamma; \Delta; \Omega, A \Uparrow\vdash C} \Uparrow L$$

Рисунок 24 — Правило переноса ресурсов из Ω в Δ.

Фаза левой инверсии завершается тогда, когда набор Ω окажется пустым.

2.3. Решение

Замечание: Ввиду того, что для проведения фокусировки не нужно следить за порядком, набор Ω отбрасывается от секвенции.

Переход от инверсии к фокусировке контролируют следующие правила:

$$\frac{\Gamma; \Delta \vdash C \Downarrow \qquad C \text{ не является атомарным}}{\Gamma; \Delta; \cdot \Uparrow \vdash C} \text{decideR}$$

(a) Правило перехода к фокусировке на цели

$$\frac{\Gamma; \Delta, A \Downarrow \vdash C}{\Gamma; \Delta, A; \cdot \Uparrow \vdash C} \text{decideL} \qquad \frac{\Gamma; \Delta, A \Downarrow \vdash C}{(\Gamma, A); \Delta; \cdot \Uparrow \vdash C} \text{decideL!}$$

(b) Правила перехода к фокусировке на гипотезах

Рисунок 25 — Правила перехода от инверсии к фокусировке

2.4. Правый фокус

Правила, описывающие алгоритм правого фокуса:

$$\frac{\Gamma; \Delta' \vdash A \Downarrow \qquad \Gamma; \Delta'' \vdash B \Downarrow}{\Gamma; \Delta', \Delta'' \vdash A \otimes B \Downarrow} \otimes R \qquad \frac{\Gamma; \cdot; \cdot \vdash A \Uparrow}{\Gamma; \cdot \vdash !A \Downarrow} !R$$

$$\frac{\Gamma; \Delta \vdash A \Downarrow}{\Gamma; \Delta \vdash A \oplus B \Downarrow} \oplus R_1 \qquad \frac{\Gamma; \Delta \vdash B \Downarrow}{\Gamma; \Delta \vdash A \oplus B \Downarrow} \oplus R_2$$

$$\frac{}{\Gamma; \cdot \vdash 1 \Downarrow} 1R \qquad \frac{\Gamma; \Delta \vdash [t/x]A \Downarrow}{\Gamma; \Delta \vdash \exists x.A \Downarrow} \exists R$$

Нет правого правила для 0

Рисунок 26 — Правила вывода для правой фокусировки

Замечание: Правое правило для коерции является частично обратимым. По-этому, сразу после раскрытия связки $!A$ приступаем к удалению возможных асинхронных связок в A посредством применения инверсии к его секвенции. $\triangle$

Замечание: В случае со связкой 1 мы завершает доказательство (отсутствие ресурсов доказывает 1). $\triangle$

2.5. Левый фокус

Правила вывода, описывающие левый фокус:

$$\frac{\Gamma; \Delta, A \Downarrow\vdash C}{\Gamma; \Delta, A \& B \Downarrow\vdash C}\,\&L_1 \qquad \frac{\Gamma; \Delta, B \Downarrow\vdash C}{\Gamma; \Delta, A \& B \Downarrow\vdash C}\,\&L_2$$

$$\frac{\Gamma; \Delta'', B \Downarrow\vdash C \qquad \Gamma; \Delta' \vdash A \Downarrow}{\Gamma; \Delta', \Delta'', A \multimap B \Downarrow\vdash C}\,\multimap L$$

Нет левого правила для $\top$.

$$\frac{\Gamma; \Delta, B \Downarrow\vdash C \qquad \Gamma; \cdot; \cdot \vdash A \Uparrow}{\Gamma; \Delta, A \supset B \Downarrow\vdash C}\,\supset L \qquad \frac{\Gamma; \Delta, [t/x]A \Downarrow\vdash C}{\Gamma; \Delta, \forall x.A \Downarrow\vdash C}\,\forall L$$

Рисунок 27 — Правила вывода для левой фокусировки

2.6. Завершение фокуса

Фаза фокуса завершается тогда, когда суждение, на котором происходит фокусировка, перестает быть синхронным. Следующие правила осуществляют контроль при переходе от фокусировки к инверсии или завершают доказательство.

$$\frac{}{\Gamma; P \Downarrow\vdash P}\,\text{init}$$

$$\frac{\Gamma; \Delta, A \Uparrow\vdash C \qquad A \text{ не атомарное и не левое синхронное}}{\Gamma; \Delta, A \Downarrow\vdash C}\,\Downarrow L$$

$$\frac{\Gamma; \Delta; \cdot \vdash A \Uparrow}{\Gamma; \Delta \vdash A \Downarrow}\,\Downarrow R$$

Рисунок 28 — Правила завершения фокуса

3. Реализация вывода в линейной логике

3.1. Реализация логических связок

Логические выражения были реализованы с помощью техники известной как *Data types à la carte* [1]. Эта техника является решением *проблемы выражения* (англ. *Expression Problem*) в языке Haskell, поставленной Филом Уодлером в 1998 году [1]:

Определение (Проблема выражения). *Проблема реализации типа данных с возможностью добавления в него новых конструкторов и функций без необходимости в перекомпиляции существующего кода и с поддержкой статической типизации.* ♢

Data types à la carte реализуют один тип, реализовав его составляющие по-отдельности, соединив их некоторым супер-типом. Рассмотрим эту технику на примере реализации выражений линейной логики.

1. Все логические связки определяются по-отдельности.

Листинг 3.1 Определение конструкторов формул

```
-- Одновременная конъюнкция
data SimConj f = SimConj f f

-- Единица
data Unit f = Unit

...
```

   Все конструкторы имеют параметр. Он будет одинаковым для всех внутри выражения.

2. Определяется супер-тип, связывающий конструторы в выражение.

Листинг 3.2 Супер-тип для конструкторов

```
newtype Expr f = In { out :: f (Expr f) }
```

Expr тоже параметризован. Его параметр передастся всем входящим в него конструкторам.

3. Определяется копроизведение типов.

Листинг 3.3 Определение копроизведения

```
data (f :+: g) a = Inl (f a) | Inr (g a)
```

Копроизведение конструкторов выступает в качестве параметра f в *Expr*.

В результате получается механизм, позволяющий строить выражения, указав, какие конструкторы формул в него входят:

Листинг 3.4 Пример логического выражения

```
someExpr :: Expr (SimConj :+: Unit)
someExpr = In (Inl (SimConj
              (In (Inr Unit))
              (In (Inr Unit))))
```

Заметно, что приходится явно достраивать конструкторы до их копроизведения, приписывая к ним конструкторы *Inl* и *Inr*. С одиннадцатью конструкторами всех логических связок линейной логики делать это было бы затруднительно. Однако, этот процесс можно автоматизировать следующим образом:

Листинг 3.5 Реализация автоматизации вставки конструкторов *Inl* и *Inr*.

```
class (Functor sub, Functor sup) => sub :<: sup where
  inj :: sub a -> sup a

instance Functor f => f :<: f where
  inj = id
instance (Functor f, Functor g) => f :<: (f :+: g) where
  inj = Inl
instance (Functor f, Functor g, Functor h, f :<: g) =>
  f :<: (h :+: g) where
  inj = Inr . inj

inject :: (g :<: f) => g (Expr f) -> Expr f
inject = In . inj
```

Замечание: Ограничение *Functor* не является здесь необходимым, однако оно нужно для полной реализации *Data types à la carte*. △

Непосредственная автоматизация реализуется в использовании *умных конструкторов* (англ. *Smart Constructors*), которые собирают необходимую логическую формулу, вставляя необходимые *Inl* и *Inr*:

Листинг 3.6 Реализация умных конструкторов.

```
(&:) :: (SimConj :<: f) => Expr f -> Expr f -> Expr f
(&:) a = inject . SimConj a

unit :: (Unit :<: f) => Expr f
unit = inject Unit
```

Умные конструкторы позволяют переписать пример (ссылка на пример) следующим образом:

Листинг 3.7 Реализация примера с использованием умных конструкторов.

```
someExpr :: Expr (SimConj :+: Unit)
someExpr = unit &: unit
```

3.2. Реализация алгоритма вывода доказатльств фокусировкой

3.2.1. Реализация инверсии

Реализация логических связок с помощью *Data types à la carte* позволяет классифицировать их с помощью классов типов, определяя функции, работающие только на связках, обладающих общим свойством. В частности, это позволяет реализовать правые обратимые правила на асинхронных связках ; необратимые и частично обратимые на синхронных.

Рассмотрим класс правых асинхронных логических связок:

```
class RightAsync c where
  rightInvRule ::
    c LinearFormula -> (Int, OrdAnt) -> [(Int, OrdSeq)]
```

Функция *rightInvRule* реализует правило вывода. Первые два аргумента образуют секвенцию-цель, записанную наоборот как $C \dashv \Gamma; \Delta; \Omega$; результат функции — секвенции-посылки правила. Можно рассматривать эту функцию как правило, действующее снизу вверх.

Число, стоящее первой в паре с упорядоченным антецедентом, является уникальным номером. Этот номер используется для генерации имен имен термов для подстановки в кванторы. У каждой секвенции в процессе вывода имеется свой уникальный номер.

Листинг 3.8 Некоторые экземпляры класса *RightAsync*

```
instance RightAsync ResImpl where
  rightInvRule (ResImpl a b) (uniq, (vals, ress, ords)) =
    [(uniq, (vals, ress, a:ords) :>> b)]

instance RightAsync AltConj where
  rightInvRule (AltConj a b) (uniq, ant) =
    [(uniq * 2, ant :>> a), (uniq * 2 + 1, ant :>> b)]

instance RightAsync ForAll where
  rightInvRule (ForAll f) (uniq, ant) =
    let newConst = Func ("Constant" ++ show uniq) []
      in [(uniq * 2, ant :>> f newConst)]
```

Как видно, функция *rightInvRule* фактически повторяет запись соответствующего правила вывода.

Помимо возможности классификации логических связок, использование *Data types à la carte* вводит ограничения на работу с ними. А именно, весь вывод приходится проводить исключительно внутри классов типов. Поэтому контролирующая часть алгоритма правой инверсии реализуется специальным классом.

Класс *RightInversion* контролирует применение правил вывода и останавливает работу, как только цель обрабатываемой секвенции не является правой асинхронной:

```
class RightInversion f where
  rightInversion ::
    f LinearFormula -> (Int, OrdAnt) -> [(Int, OrdSeq)]
```

Его наивная реализация

```
instance (RightAsync f) => RightInversion f where
  rightInversion f ...
```

Листинг 3.9 Использование продвинутого перекрытия для отличия логических связок.

```
-- Реализация переносится на вспомогательную функцию
instance (RightAsyncPred a flag, RightInversionImpl flag a) =>
  RightInversion a where
  rightInversion = rightInversionImpl (undefined :: flag)

class RightInversionImpl flag a where
  rightInversionImpl ::
    flag -> a LinearFormula -> (Int, OrdAnt) -> [(Int, OrdSeq)]

-- Предикат, отличающий правые асинхронные связки
class RightAsyncPred (a :: * -> *) flag | a -> flag

-- Применение правил к правым асинхронным связкам
instance (RightAsync f) => RightInversionImpl HTrue f where
  rightInversionImpl _ f x =
    rightInvRule f x >>= applyRightInversion

-- Останов для всех остальных
instance (f :<: LinearInput) => RightInversionImpl HFalse f where
  rightInversionImpl _ f (uniq, ant) = [(uniq, ant :>> inject f)]
```

```
instance RightInversion f where
  rightInversion f ...
```

приводит к перекрытию. Поэтому требуется вводить дополнительные данные, чтобы можно было отличить правые асинхронные связки от остальных. Это делается с помощью техники *продвинутого перекрытия* [2].

Алгоритм левой инверсии реализуется аналогичным образом, по-этому подробности его реализация тут приводиться не будет.

Алгоритм инверсии собирается из алгоритмов правых и левых инверсий:

```
inversion :: (Int, OrdSeq) -> [(Int, LinSeq)]
inversion x =
```

```
do (uniq, ant :>> goal) <- applyRightInversion x
   (uniq, ant') <- applyLeftInversion (uniq, ant)
   return (uniq, ant' :=> goal)
```

3.2.2. Реализация фокуса

Правила, применяемые в процессе фокусировки, не являются обратимыми. Это значит, что, при вычислении доказательства с использованием правила

$$\frac{\Gamma; \Delta' \vdash A \Downarrow \qquad \Gamma; \Delta'' \vdash B \Downarrow}{\Gamma; \Delta', \Delta'' \vdash A \otimes B \Downarrow} \otimes R$$

неясно, как много ресурсов от $\Delta = (\Delta', \Delta'')$ отдать для доказательства левой ветки, а сколько — для правой. Это вызывает недетерминизм в распеределении ресурсов [10].

Определение (Управление ресурсами). *Техника, занимающаяся распределением ресурсов между подзадачами, называется* управлением ресурсами *(англ.* Resource Management*).* ◊

Замечание: Для *управления ресурсами* был выбран следующий подход: использовать все ресурсы для доказательства одной подзадачи; затем использовать оставшиеся для доказательства следующей. △

Проблема управления ресурсами вызывает аналогию с процессами: для доказательства следующей подзадачи нужно подождать, пока не будет доказана первая. Если первая — процесс p_1, а вторая — процесс p_2, то p_2 находится в ожидании сообщения от p_1. Сообщением являются оставшиеся после доказательства ресурсы.

В следствии этого, все доказываемые секвенции в процессе фокусировки отождествляются с процессами. Функции, реализующие правила вывода, возвращают результат, содержащий значение типа *Channel*.

Листинг 3.10 Реализация типа *Channel* для связи между процессами.

```
data Channel =
  Channel { wait :: [LinearFormula]
          , get  :: [(Int, LinSeq)] }
          -- где wait - сообщение, пересылаемое другим процессам;
          --     get  - листья в дереве доказательств, полученные
          --            применением правила init.

-- получения сообщения от процесса
receive :: Channel -> ([LinearFormula], Channel)
receive (Channel ws gs) = (ws, Channel [] gs)

-- слияние каналов
merge :: [Channel] -> Channel
merge chan =
  let ws = map wait chan
      gs = map get chan
      in Channel (mconcat ws) (mconcat gs)
```

```
type Result = Either ErrorMsg Channel

data ErrorMsg =
  ErrorMsg
    String        -- сообщение ошибки
    (Int, LinSeq) -- уникальный номер и секвенция,
                  -- на которой она произошла.
```

Выявление провала доказательства может быть осуществлено только во время фазы фокуса. Ввиду этого, результат применения правила должен поддерживать контроль ошибок. Все правила вывода фазы фокуса возвращают значение следующего типа:

Замечание: Достаточно возвращать сообщение об ошибке в случае провала. Решение возвращать номер и секвенцию было принято произвольно. △

Реализации правил вывода в правом и левом фокусе аналогична реализациям правой и левой инверсий. Отличие заключается в осуществлении управления ресурсами и контроля ошибок.

Листинг 3.11 Пример реализации управления ресурсами для одновременной конъюнкции

```
instance RightSyncFocus SimConj where
rightSyncFocus (SimConj a b) (uniq, (vals, ress)) =
  do leftChan <- applyRightFocus (uniq * 2, (vals, ress) :=> a)
     let (leftRess, leftChan') = receive leftChan
     rightChan <- applyRightFocus
                    (uniq * 2 + 1, (vals, leftRess) :=> b)
     return $ merge [leftChan', rightChan]
```

Листинг 3.12 Пример контроля ошибок.

```
instance RightSyncFocus Disj where
rightSyncFocus (Disj a b) (uniq, ant) =
  (applyRightFocus (uniq, ant :=> a))
    `mplus` (applyRightFocus (uniq, ant :=> b))
```

Пример контроля ошибок. Правое правило для дизъюнкции создает две потенциальные подзадачи. Одна из них должна быть доказана, но не ясно, какая. Функция mplus позволяет выбрать первую, доказательство которой успешно завершилось.

После левой фокусировки, принятие решения может осуществиться с помощью правил *init* и $\Downarrow L$. По-этому нужно различать левые синхронные, атомарные и все остальные ресурсы. Это делается с помощью улучшенно техники AdvancedOverlapping. тут ссылку на листинг

В реализации класса, контролирующего применение правил левого фокуса, используется несколько улучшенная версия AdvancedOverlapping, позволяющая отличать атомарные, левые синхронные и все остальные логические связки. Это делается посредством использования двух предикатов вместо одного:

```
class LeftFocus f where
  leftFocus :: f LinearFormula -> (Int, LinSeq) -> Result
```

```
instance ( AtomPred a aflag
         , LeftFocusPred a lflag
         , LeftFocusImpl aflag lflag a) => LeftFocus a where
  leftFocus = leftFocusImpl (undefined :: aflag) (undefined
     :: lflag)
```

3.2.3. Реализация правил принятия решения

Принятние решения о том, фокусироваться ли на цели или на посылках определяется контролируется классом *Decision*:

```
class Decision c where
  decision :: c LinearFormula -> (Int, LinAnt) -> Result
```

В его реализации тоже используется AdvancedOverlapping для того, чтобы отличать атомарные логические связки от остальных.

Заключение

1. Разработана система вывода в линейной логике.
2. Исследованы и применены следующие возможности Haskell:
   - Data types a ‘ la carte
   - Перекрытие классов

Список литературы

1. *Swierstra W.* Data types à la carte // FUNCTIONAL PEARL.
2. HaskellWiki. — URL: `https://wiki.haskell.org/GHC/AdvancedOverlap` (дата обр. 15.11.2015).
3. *Pfenning F.* Linear Logic. — 2002.

4. nLab. — URL: http://ncatlab.org/nlab/show/deductive+system (дата обр. 15.11.2015).
5. nLab. — URL: http://ncatlab.org/nlab/show/sequent+calculus (дата обр. 15.11.2015).
6. *Pfenning F.* Linear Logic. — 2002. — С. 3—4.
7. *Pfenning F.* Linear Logic. — 2002. — С. 19—21.
8. *Pfenning F.* Linear Logic. — 2002. — С. 24—25.
9. *Pfenning F.* Linear Logic. — 2002. — С. 72—76.
10. *Pfenning F.* Linear Logic. — 2002. — С. 72.