Санкт-Петербургский государственный университет

Кафедра системного программирования
Программная инженерия

Королихин Владимир Игоревич

Доказательство корректности функциональной реализации библиотек принтер-комбинаторов с выбором

Курсовая работа

Научный руководитель:
д.т.н., доцент Д. В. Кознов

Консультант:
к.ф.-м.н. А. В. Подкопаев

Санкт-Петербург
2020

Оглавление

Введение

Данная работа посвящена задаче преобразования данных из некоторого абстрактного вида, например, синтаксических деревьев программ, в текст. Такая задача называется *pretty printing*, а соответствующий инструмент *pretty printer* (далее *принтер*).

В качестве примера данных будем рассматривать синтаксические деревья программ. В сфере IT программы используются повсеместно, а потому демонстрировать работу принтеров на них удобно.

Один и тот же текст можно печатать по-разному. Рассмотрим процедуру bar на языке C#: на рис. 1 и 2 представлены три результата форматирования текста этой процедуры, и каждый соответствуют одному абстрактному синтаксическому дереву.

```
void bar(){ int count = 0;
  while (very_long_condition1() &&
  very_long_condition2()) { count++; } }
```

Рис. 1: Процедура bar: неформатированный код

```
void bar()
{
    int count = 0;
    while (very_long_condition1() && very_long_condition2())
    {
        count++;
    }
}
```

(a) Без учета ширины

```
void bar()
{
    int count = 0;
    while (very_long_condition1() &&
           very_long_condition2())
    {
        count++;
    }
}
```

(b) С учетом ширины

Рис. 2: Процедура bar: форматированный код

Текст процедуры bar, представленный на рис. 1, читать трудно, так как в нем явно не прослеживается структура программы. Следует отметить, что в рамках одного языка существует несколько способов текстового представления одной и той же программы (проще говоря, способов форматирования), например удовлетворяющих разным стандартам (такие стандарты существуют в большинстве крупных промышленных проектах). Например, могут разрешаться длинные языковые конструкции (рис. 2a), но недостатком такого представления является то, текст может не помещаться в ширину экрана. Форматирование с ограничением на ширину вывода иллюстрируется на рис. 2b. Таким образом, определили функцию *вариативности* в рассматриваемом принтере.

Также немаловажное значение играет и обозримость текста программы — при соблюдении прочих ограничений, оптимальное форматирование занимает минимальное количество строк.

В функциональных языках программирования классическим подходом к написанию принтеров являются принтер-комбинаторы [6]. Базовыми считаются библиотеки принтер-комбинаторов Джона Хьюза [1] и Филиппа Вадлера [4]. Однако эти библиотеки обладают существенным недостатком: они имеют небольшую выразительную мощность, а потому подаваемые на вход данные обрабатываются слишком единообразно, что не позволяет выразить разные стандарты форматирования.

В данной работе рассматривается способ задания принтеров с помощью *принтер-комбинаторов с выбором*. Они позволяют создавать принципиально разные варианты форматирования для каждого поддерева программы. Базовая реализация такого принтера была представлена в работе Пабло Азеро и Дойце Свирстра [2]. Однако предложенной ими библиотеке, алгоритм в худшем случае имеет экспоненциальную сложность. Существенно отличаются от них алгоритмы Антона Подкопаева [5] и Жана-Филиппа Бернарди [6], которые полиномиальны.

Принтеры можно использовать в языковых процессорах (компиляторах, интерпретаторах, средствах интегрированной разработки (IDE)) для автоматического поддержания стандарта форматирования, для визуализации дерева синтаксического разбора и т.д.

Для принтеров важным является верификация — формальное доказательство их корректности. Важно, чтобы представляемый текст удовлетворял определенным свойствам, не изменял семантику передаваемых данных. Доказательством корректности полиномиальных принтеров с выбором еще никто не занимался. Это является упущением, так как данный класс принтеров решает задачу представления текста довольно быстро. Его можно использовать как часть верифицированных языковых процессоров, в частности, компиляторов, подобных CompCert.

Целью данной работы является доказать корректность библиотеки [6] относительно базовой реализации [2], а также механизировать доказательство на языке Coq. Библиотека корректна, если результат ее выполнения является не хуже результата базовой реализации. Coq – это интерактивное программное средство доказательств теорем. Он позволяет естественным образом описывать привычные нам математические утверждения и доказывать их с помощью встроенных тактик.

1. Постановка задачи

Цель данной работы заключается в составлении требований и доказательстве корректности библиотеки [6] на языке Coq. Библиотека корректна, если результат ее выполнения является не хуже результата базовой реализации. Базовую реализацию предполагается взять из [2]. Для достижения необходимого результата были сформулированы следующие задачи:

- Изучить предметную область: провести обзор существующих способов доказательств утверждений на языке Coq, освоить данный язык программирования.
- Реализовать на языке Coq авторскую библиотеку [6] и тривиальный алгоритм принтер-комбинаторов с выбором [2].
- Исследовать свойства библиотек и определить требования, накладываемые на принтер.
- Механизировать доказательство корректности библиотек в Coq.
- Предоставить верифицированную версию библиотеки [6] на языке Haskell.

2. Обзор литературы и существующих решений

Первые принтер-комбинаторы с выбором были предложены в работе [2]. Предложенная там же реализация, хотя и основывается на «ленивых» вычислениях, имеет в худшем случае экспоненциальную сложность. В [3] приводятся некоторые оптимизации, однако и они не дают существенного прироста к производительности. В [6] представлен полиномиальный алгоритм, потому именно он был выбран в нашей работе.

2.1. Общая модель принтер-комбинаторов с выбором

Пусть необходимо напечатать код, составленный из блоков, изображенных на рис. 3. Это можно сделать с помощью трех комбинаторов, реализованных в библиотеке принтера.

(a) (b)

Рис. 3: Пример раскладок

Горизонтальная композиция (*beside* — рис. 4) позволяет поместить одну раскладку рядом с другой, при этом сдвигая вторую раскладку на длину последней строчки первой. Вертикальная композиция (*above* — рис. 5a) помещает одну раскладку под другой. Горизонтальная композиция со сдвигом ($fill\ \ 4$ — рис. 5b) использует в качестве параметра помимо раскладок еще и число, на которое нужно сдвинуть все строчки, кроме первой во второй раскладке.

Отметим, что комбинатор $fill$ в реализации работ [2] и [6] не участвует. Необходимость $fill$ рассмотрена в [5], а потому было решено добавить его в библиотеки для последующего анализа.

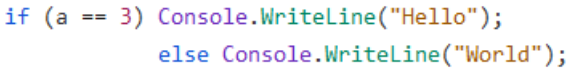
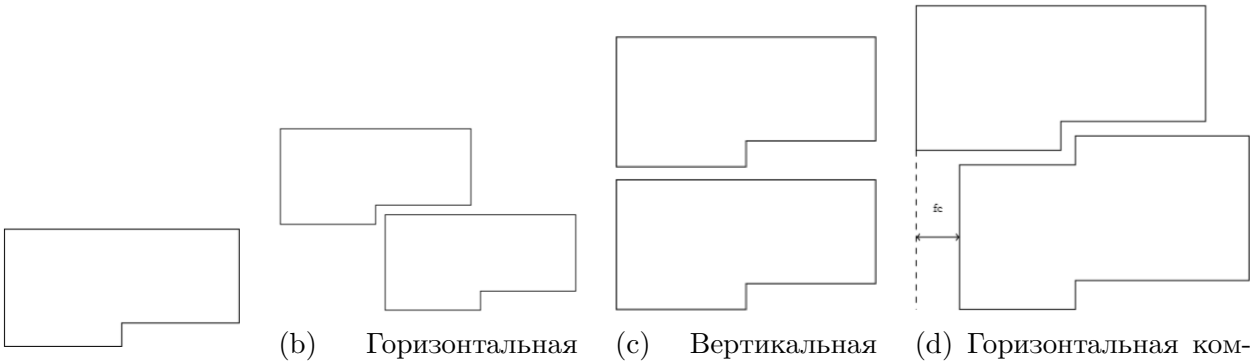


Рис. 4: Применение комбинатора beside

(a) Применение комбинатора above (b) Применение комбинатора fill 4

Рис. 5: Применение комбинаторов



(a) Блок текста (b) Горизонтальная композиция (beside) (c) Вертикальная композиция (above) (d) Горизонтальная композиция со сдвигом (fill)

Рис. 6: Работа с блоками текста *Format*

Таким образом, комбинаторы работают с блоками текста и формируют из них новые, согласно вышеизложенным правилам. Схематично такие блоки и соответствующие комбинаторы изображены на рис. 6. Данные блоки (рис. 6a) мы будем называть *раскладкой* и обозначать $Format$.

Также имеется примитив *indent*, позволяющий сдвигать раскладку на заданное количество позиций вправо с помощью добавления необходимого количества пробельных символов, и примитив *line*, который конструирует раскладку по переданной строке.

Описанные примитивы, имеют следующие сигнатуры:

$line : string \to Format$

$indent : \mathbb{N}_0 \to Format \to Format$

$add\_beside : Format \to Format \to Format$

$add\_above : Format \to Format \to Format$

$add\_fill : \mathbb{N}_0 \to Format \to Format \to Format$

Здесь важно подчеркнуть, что данные функции в библиотеках [2], [5] и [6] реализованы одинаково. Каждая библиотека реализует один общий интерфейс, методы которого используют эти примитивы.

Данный интерфейс обобщает понятие *документа*. Здесь документ можно рассматривать как множество $Format$-элементов и для него вводятся аналогичные примитивы. Документ представляет собой индуктивный тип, каждому конструктору которого соответствует своя операция из реализующей интерфейс библиотеки.

```
Inductive Doc : Type :=
  | Text   (s: string)
  | Indent (t: nat) (d: Doc)
  | Beside (d: Doc) (d: Doc)
  | Above  (d: Doc) (d: Doc)
  | Fill   (d: Doc) (d: Doc) (s: nat)
  | Choice (d: Doc) (d: Doc).
```

Комбинатор *choice*, который еще до этого не рассматривался представляет собой объединение множеств раскладок, переданных в качестве аргументов. Благодаря этому достигается необходимая вариативность раскладок, так как комбинатор конструирует общий список из различных представлений текста, не выбирая какой-то конкретный вариант.

Интерфейс, предоставляемый документом, удобен для последующего анализа работы библиотек, так как позволяет абстрагироваться от конкретной реализации. На вход каждой подаются только правила, которые нужно применить.

2.2. Хранение раскладок

Опишем подробнее структуру раскладки. Для простоты можно представлять ее себе как кортеж из пяти элементов.

$$\langle height, first\_line\_width, middle\_width, last\_line\_width, to\_text\rangle \in \mathbb{N} \times \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0 \times F$$

где $F : \mathbb{N}_0 \to string \to string$. По каждой раскладке можно получить ее текстовое представление с помощью функции to_text, где первым аргументом выступает сдвиг раскладки. Первые четыре поля структуры характеризуют ее геометрические размеры.

Заметим, что некоторые раскладки нам могут подходить больше с точки зрения читаемости, даже если они имеют одинаковую высоту. Не умаляя общности, будем называть раскладку a *меньше* b, если:

$$\begin{aligned} &height(a) \leqslant height(b) \ \wedge \\ &first\_line\_width(a) \leqslant first\_line\_width(b) \ \wedge \\ &middle\_width(a) \leqslant middle\_width(b) \ \wedge \\ &last\_line\_width(a) \leqslant last\_line\_width(b) \end{aligned}$$

и обозначать $a \preccurlyeq b$. Очевидно, что $\preccurlyeq$ является отношением частичного порядка на раскладках.

В дальнейшем (см. 3.2) мы модифицируем частичный порядок для поддержки комбинатора $fill$. Важной особенностью данного комбинатора является то, что в своей реализации он использует информацию о $first\_line\_width$, которая для других комбинаторов не требуется.

Теперь определим множество раскладок, которые будут удовлетворять требованиям оптимального форматирования.

Для множества раскладок S, и ширины w *лучшими* раскладками $best(S, w)$ мы будем называть такие, что:

$$\forall s \in best(S, w).total\_width(s) \leq w \wedge \nexists s' \in S.total\_width(s') \leq w \wedge height(s') < height(s)$$

2.3. Общий алгоритм работы принтер-комбинаторов с выбором

Опишем более подробно работу библиотек [5] и [6]. Как было сказано ранее, на вход принтерам подается некоторый экземпляр *Doc*. Его можно представлять себе в виде дерева, в листьях которого находятся конструкторы документа $Text$. Напомним, что комбинатор *choice* создает множество раскладок, а потому нам недостаточно хранить по одной раскладке на каждый узел дерева. На данном этапе можно считать, что такое множество представляет собой обычный список. Разбор промежуточных узлов дерева, принимающих в качестве аргументов два списка раскладок, заключается в применении соответствующих комбинаторов к спискам по принципу "каждый с каждым". Таким образом, работа библиотек состоит в комбинации списков, с некоторыми последующими оптимизациями. В результате разбора *Doc* получаем список раскладок с сохранением вариативности.

На каждую раскладку накладывается одно важное ограничение – *максимальная ширина*. Из результирующего списка удаляются неподходящие по ширине раскладки. Из оставшихся выбирается раскладка с минимальной высотой и снова производится удаление, но уже тех, которые не совпадают по найденной высоте.

2.4. Тривиальная реализация принтер-комбинаторов с выбором

В работе [3] множество вариантов, соответствующее экземпляру Doc, представляется ленивым списком всех возможных раскладок, удовлетворяющих ограничению на максимальную ширину. Этот список отсортирован в порядке "ухудшения раскладок". Таким образом, на конечном шаге потребуется взять первую раскладку. Она и будет оптимальной.

В нашей реализации на Coq, в котором нет встроенной ленивости, мы решили отказаться от этой оптимизации, потому что данная реализация интересна нам только как референсная. Как следствие, она должна быть реализована как можно проще. Для этого раскладки хранятся в том порядке, в котором были получены, а на последнем шаге выбираются те, которые имеют минимальную высоту и попадают под ограничение ширины.

2.5. Фильтрация с помощью поиска множества Парето

В реализации Жана-Филиппа Бернарди [6] все вычисления также проводятся на списках, но отличается она тем, что удаляет часть раскладок после применения комбинаторов. Ранее упоминалось (см. 2.1), что на множестве раскладок можно ввести

частичный порядок $\preccurlyeq$. Определим *множество Парето* для списка X следующим образом:

$\{x \in X \mid \nexists y \in X . x \neq y \wedge y \preccurlyeq x\}$.

Процедуру удаления раскладок по такому правилу назовем *фильтрацией по Парето*.

Таким образом, список, состоящий из элементов множества Парето содержит в себе меньшее (или такое же, в случае, если все элементы несравнимы) количество раскладок, что уменьшает количество операций на более высоких узлах дерева.

2.6. Верификация библиотек принтер-комбинаторов с выбором

Принципиально новыми подходами в решении задачи оптимальной печати принтер-комбинаторов с выбором являются алгоритмы, изложенные в статьях [5] и [6]. Однако в них описана лишь алгоритмическая сложность и приведены тесты производительности для каждой из библиотек. На данный момент ни одна из них формально не верифицирована и попыток сделать не представлено.

3. Свойства принтер-комбинаторов с выбором

В этой секции вводятся формальные утверждения, которые являются основой доказательств. Они были приняты после тщательного анализа различных вариантов раскладок и работы комбинаторов. Сложность заключалась в том, что в результате применения какого-либо комбинатора, раскладки преобразовывались к таким видам, которые по своим геометрическим характеристикам не могут быть рассмотрены как общий случай результата работы комбинатора.

Напомним (см. раздел 2.3), что результатом работы каждой из библиотек является список раскладок, минимальных по высоте и помещающихся в заданную ширину w.

3.1. Корректность раскладок

Как отмечалось ранее, каждая раскладка имеет четыре геометрические характеристики: высоту, ширину первой и последних строчек, максимальную ширину центральной строчки. Рассмотрим пример раскладки высоты 4 (рис. 7). Для наглядности каждый символ в раскладке будем представлять прямоугольным блоком. Ее характеристики ширины следующие: $first\_line\_width = 1, middle\_width = 3, last\_line\_width = 1$

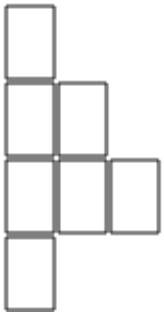


Рис. 7: Пример раскладки

В случае, если раскладка имеет высоту 1, то все характеристики ширины считаем равными. Если же высота 2, то считаем, что $middle\_width = first\_line\_width$. Высота раскладки 0 считается недопустимой, т.к. даже пустая строка имеет высоту 1.

Раскладки, удовлетворяющие приведенным выше ограничениям, будем называть *корректными*.

Такие ограничения вводятся с целью корректного удаления не оптимальных раскладок. Заметим, что это никак не влияет на реализацию, т.к. все раскладки, создаваемые *line* удовлетворяют ограничениям корректности.

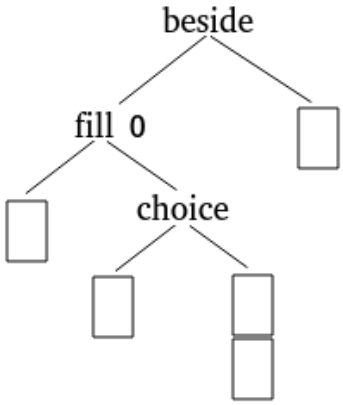


Рис. 8: Пример документа, с использованием комбинатора fill

Рис. 9: Список, после разбора дерева

3.2. Модификация частичного порядка

Анализ результатов, полученных при использовании отношения $\preccurlyeq$ на различных структурах документа показал, что в некоторых случаях комбинатор $fill$ может создавать раскладки, удалять которые нельзя. Иначе, в худшем случае результирующий список может оказаться пустым. Данное соображение хорошо иллюстрирует следующий пример.

Рассмотрим структуру документа, изображенную на рис. 8. Для наглядности она представлена в виде дерева, в листьях которого, согласно определению *Doc* находятся раскладки. В промежуточных узлах — комбинаторы, которые необходимо применить. Список, полученный после разбора данного дерева изображен на рис. 9. Синим цветом обозначены две раскладки после применения комбинатора $fill$. Заметим, что если реализация использует отношение $\preccurlyeq$ для раскладок, то после применения *choice* раскладка высоты 2 будет удалена, т.к. удовлетворяет неравенству $\leq$ по всем геометрическим характеристикам. В итоге получится список всего из одной раскладки ширины 3 (на рис. 9 она показана слева). Однако при ограничении на максимальную ширину в 2 символа получим, что такой результат нам не подходит, в то время как, если бы мы не удаляли раскладку, один вариант остался.

Вследствие этого, было принято решение модифицировать отношение частичного порядка следующим образом: раскладка a *меньше с учетом высоты* раскладки b, если:

$$a \preccurlyeq_m b = \begin{bmatrix} a \preccurlyeq b, \ height(a) = 1 \wedge height(b) = 1 \\ a \preccurlyeq b, \ height(a) > 1 \wedge height(b) > 1 \end{bmatrix}$$

3.3. Корректность комбинаторов

Следующая наша цель определить свойства, которые позволят в дальнейшем проводить доказательство в более удобной форме, а также абстрагироваться от конкретной реализации комбинаторов. В случае расширения библиотеки новыми комбинаторами потребуется лишь доказать эти свойства. Под $\underline{\mathrm{f}}$ понимается любой из трех комбинаторов *add_beside*, *add_fill*, *add_above*.

Для начала отметим, что комбинаторы в процессе своей работы должны создавать только корректные раскладки. Действительно, корректность раскладок гарантирует полное соответствие геометрическим характеристикам будущего напечатанного текста, а также влияет на результат обработки и сравнения раскладок с другими.

Если две раскладки связаны отношением меньше, то и результат применения комбинаторов должен давать нестрогое неравенство некоторых компонент. Для пояснения данного утверждения еще раз модифицируем наше отношение $\preccurlyeq_m$. Отметим, что данная модификация никак не затрагивает реализацию и используется только в доказательстве.

Для раскладок a и b зададим отношение:

$$a \preccurlyeq'_m b = \begin{cases} a \preccurlyeq' b, \ height(a) = 2 \\ a \preccurlyeq_m b, \ height(a) \neq 2 \end{cases}$$

где $\preccurlyeq'$ то же отношение, что и $\preccurlyeq$, но без сравнения по *middle_width*.

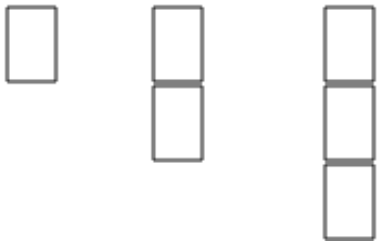


Рис. 10: Раскладки a, b и c соответственно

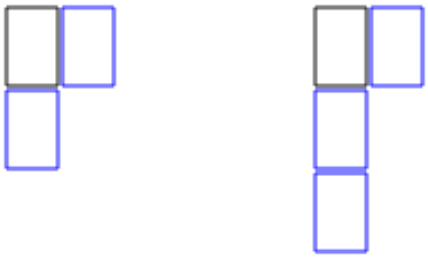


Рис. 11: Результат $fill \ \ 0 \ \ a \ \ b$ и $fill \ \ 0 \ \ a \ \ c$ соответственно

Покажем для чего потребовалась модификация, т.е. почему для $\preccurlyeq_m$ свойство 3 не выполняется. Рассмотрим раскладки на рис. 10 и применим к ним $fill$ следующим образом: $fill \ \ 0 \ \ a \ \ b$ и $fill \ \ 0 \ \ a \ \ c$.

В результате получим две новые раскладки, изображенные на рис. 11. Т.к. они обе корректны, то $middle\_width(fill\ \ 0\ \ a\ \ b) \not\preccurlyeq middle\_width(fill\ \ 0\ \ a\ \ c)$. Однако такое соотношение не удовлетворяет свойству 3, если бы на месте $\preccurlyeq'_m$ использовался $\preccurlyeq_m$.

Таким образом, корректность комбинаторов, необходимая для успешного доказательства библиотек выражается с помощью следующих свойств:

1) Сохраняется корректность раскладки: если раскладки a и b корректны, то раскладка $\underline{f}\ \ a\ \ b$ также корректна.

2) Сохраняется ширина раскладки: для любых корректных раскладок a и b из того, что $width(\underline{f}\ \ a\ \ b) \leq w$ следует $width(a) \leq w$ и $width(b) \leq w$.

3) Для любой четверки корректных раскладок a, b, c, d, если $a \preccurlyeq'_m b$ и $c \preccurlyeq'_m d$, тогда верно $(\underline{f}\ \ a\ \ c) \preccurlyeq'_m (\underline{f}\ \ b\ \ d)$.

4. Корректность библиотек принтер-комбинаторов с выбором

В данной секции формально определяется корректность библиотек и вводятся основные теоремы, которые были доказаны в текущей работе. Перед тем как это сделать введем некоторые обозначения:

***Определение* 4.1.** *Функция* $evaluator : \mathbb{N}_0 \to Doc \to list\ \ Format$ *принимает ограничение на максимальную ширину раскладок и документ, согласно которому применяются комбинаторы. Результатом является список раскладок.*

***Определение* 4.2.** *Функция* $pretty\_list : evaluator \to \mathbb{N}_0 \to Doc \to list\ \ Format$ *принимает некоторую абстрактную функцию, с сигнатурой из определения 4.1, ограничение на максимальную ширину раскладок и документ. После исполнения* $evaluator$*, производится удаление раскладок с шириной, большей допустимой. В получившемся списке ищется раскладка минимальной высоты и удаляются раскладки с неравной высотой.*

Таким образом, у каждой библиотеки имеется функция $pretty\_list$ и некий $evaluator$, реализация которого соответствует либо алгоритму из 2.4, либо 2.5. Для первого алгоритма обозначим его как $evaluatorTrivival$, для второго $evaluatorPareto$. Результат функции $pretty\_list$ является результатом работы принтеров из [2] и [6] соответственно.

4.1. Понятие корректности реализаций

В обоих библиотеках, для каждого узла дерева имеется некоторый список раскладок. Часть раскладок из данного списка может удаляться благодаря введению отношения $\preccurlyeq$, поэтому первым делом необходимо показать, что высота раскладок в новом списке будет совпадать с высотой раскладок в тривиальной реализации и, что список новых раскладок будет содержаться в тривиальном списке. Также важно проверить, что получившийся список не оказался пустым.

Первоначальной задачей было показать равенство двух списков, т.к. из этого факта следует эквивалентность двух реализаций. Данное утверждение можно сформулировать в виде теоремы:

***Теорема* 4.1.** *Для любых* $width \in \mathbb{N}_0$ *и* Doc *выполняется*

$$pretty\_list\ \ evaluatorPareto\ \ width\ \ doc \equiv pretty\_list\ \ evaluatorTrivial\ \ width\ \ doc$$

Однако в процессе доказательства был найден контрпример, опровергающий данную гипотезу. Рассмотрим документ Doc, изображенный на рис. 12.

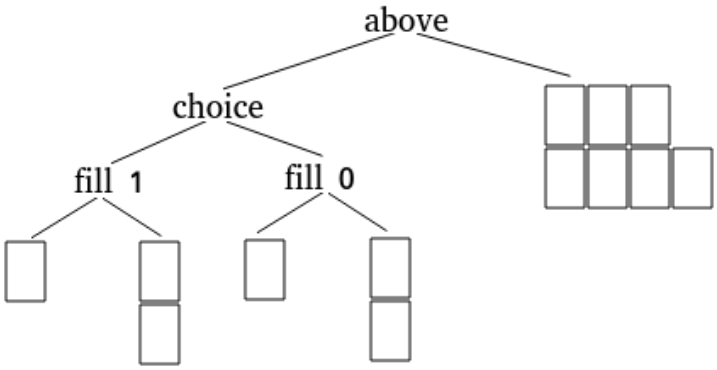


Рис. 12: Контрпример

(a) (b)

Рис. 13: Результат документа из контрпримера

В случае тривиальной реализации получаем раскладки, изображенные на рис. 12. Синим цветом показаны раскладки после применения *choice*. Заметим, что синяя раскладка на рис. 13b меньше той, что на рис. 13a, а потому последняя подлежит удалению, что приводит в итоге к отсутствию раскладки 13a в результирующем списке для оптимизированных реализаций. Таким образом утверждение о равенстве списков нарушается.

В связи с этим было решено заменить равенство на нестрогое включение множества раскладок оптимизированной версии в тривиальную. По причине того, что пустое множество элементов содержится в любом другом, а такой случай нас не устраивает, нужно проверять его отдельно.

4.2. Доказательство корректности реализации с фильтрацией по Парето

Покажем корректность, доказав следующую теорему.

***Теорема* 4.2.** *Для любых* $width \in \mathbb{N}_0$ *и* Doc *выполняется*

$$pretty\_list\ \ evaluatorPareto\ \ width\ \ doc \subseteq pretty\_list\ \ evaluatorTrivial\ \ width\ \ doc$$

Корректность была доказана на основе следующих наблюдений:

1) На каждом шаге алгоритма список, полученный с использованием филь-

трации по Парето содержится в списке тривиальной версии. Действительно, в обоих реализациях раскладки никак не модифицируются, а лишь комбинируются друг с другом.

2) В результирующих списках для обоих реализаций минимальные высоты раскладок, попадающих по ограничения на максимальную ширину совпадают.

Для доказательства первого пункта была выдвинута теорема:

***Теорема* 4.3.** *Для любых* $width \in \mathbb{N}_0$ *и Doc верно, что* $evaluatorPareto\ \ width\ \ doc\ \subseteq evaluatorTrivial\ \ width\ \ doc$.

Она доказывалась индукцией по структуре документа. Имея в качестве индукционного предположения 2 пары списков, для которых выполняется утверждение, можно показать, что комбинация "каждый с каждым" будет также ему удовлетворять.

Для доказательства пункта 2 были введены понятия корректности раскладок и комбинаторов. Заметим, что если для любого элемента b из списка тривиальной реализации найдется элемент a из оптимизированной версии, такой что $a \preccurlyeq'_m b$, то мы получим требуемое. Действительно, т.к. введенное отношение гарантирует $height(a) \leq height(b)$, а включение одного списка в другой следует из пункта 1, что дает необходимое равенство высот для списков.

Осталось показалось выдвинутое предположение о наличии раскладки с меньшей высотой. Оно следует из леммы о контексте.

***Лемма* 4.1** (О контексте)**.** *Для любых* $a \preccurlyeq'_m b$ *и Doc верно, что* $C[a] \preccurlyeq'_m C[b]$.

Под контекстом понимаются более высокие узлы дерева. Лемма позволяет удалять некоторые раскладки, реализуя в каком-то смысле жадность и возможность делать локальный выбор на каждом шаге разбора документа.

Докажем это, пользуясь индукцией по структуре документа. Очевидно, что на этапе конструирования раскладок ($line$) лемма выполняется. В случае с комбинаторами имеем 3 свойство корректности, позволяющее провести индукционный переход. Заметим, что при фильтрации по Парето отношение $\preccurlyeq_m$ сильнее $\preccurlyeq'_m$ в том смысле, что для $\preccurlyeq'_m$ не требуется проверка $middle\_width$ для раскладок высоты 2. Таким образом, если при использовании 3 свойства мы получаем раскладку, которая удаляется по Парето, то в отфильтрованном списке будет находиться элемент меньший удаленного. Но если для двух раскладок a и b выполняется $a \preccurlyeq_m b$, то $a \preccurlyeq'_m b$ гарантируется.

Вернемся к проверке списка раскладок на пустоту. Фактически наличие элементов следует из предыдущего доказательства. В самом деле, если для любой раскладки из тривиального списка найдется меньшая с учетом высоты раскладка из оптимизированного списка, то оптимизированный список гарантированно не пуст.

Таким образом, теорема 4.2 доказана.

5. Экстракция кода

С помощью стандартных команд извлечения кода на Coq, используемых для создания верифицированных и относительно эффективных функциональных программ, была получена библиотека на языке Haskell.

Модель транслируется на другой функциональный язык, при этом стираются зависимые типы, использующиеся в Coq. Однако такой подход не влияет на результат, а потому программа остается корректной. В то же время, на данном этапе мы принимаем "на веру" корректность такой экстракции внутри самого Coq.

6. Заключение

В рамках данной работы были достигнуты следующие результаты:

- На языке Coq реализованы авторские библиотеки из [2] и [6].
- Определены требования, необходимые для корректной работы библиотек.
- Поддержан комбинатор $fill$, отсутствующий в авторских библиотеках, а также уточнен частичный порядок для доказательства леммы о контексте.
- Выполнена механизация доказательств корректности библиотек в Coq:
  - Свойства частичного порядка (480 строк)
  - Корректность комбинаторов и раскладок (652 строки)
  - Связь минимальной высоты с леммой о контексте (1174 строки)
  - Свойства множества Парето и доказательство корректности библиотеки (2356 строк)
- Выполнена экстракция верифицированного кода библиотеки [6] с языка Coq на язык Haskell.

Программная разработка расположена на веб-сервисе для хостинга IT-проектов GitHub `https://github.com/vokor/PPCombinatorsProof`.

Существует несколько направлений для развития данной работы. Стоит доказать корректность библиотеки Антона Подкопаева [5] полиномиальных принтер-комбинаторов с выбором. Оптимизация в данной работе заключается в оптимальном хранении раскладок, что представляет интерес для исследования. Также в планах реализовать принтер-комбинаторную библиотеку с выбором на языке C и доказать ее корректность с помощью логики Хоара.

7. Благодарности

Выражаю благодарность своему научному консультанту Подкопаеву Антону Викторовичу к.ф.-м.н., руководителю группы слабых моделей памяти в JetBrains Research за предложенную тему, а также ценные советы в ходе исследований и реализации принятых решений.

Также хочется поблагодарить за возможность присутствия на учебно-практическом семинаре по семантике Хоара (в Coq). Приобретенные знания пригодятся в последующей работе над проектом.

Список литературы

[1] John Hughes. “The design of a pretty-printing library”. В: *International School on Advanced Functional Programming*. Springer. 1995, с. 53—96.

[2] Pablo Azero и Doaitse Swierstra. *Optimal pretty-printing combinators*. 1998.

[3] S Doaitse Swierstra, Pablo R Azero Alcocer и Joao Saraiva. “Designing and implementing combinator languages”. В: *International School on Advanced Functional Programming*. Springer. 1998, с. 150—206.

[4] Philip Wadler. “A prettier printer”. В: *The Fun of Programming, Cornerstones of Computing* (2003), с. 223—243.

[5] Anton Podkopaev и Dmitri Boulytchev. “Polynomial-Time Optimal Pretty-Printing Combinators with Choice”. В: *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*. Springer. 2014, с. 257—265.

[6] Jean-Philippe Bernardy. “A pretty but not greedy printer (functional pearl)”. В: *Proceedings of the ACM on Programming Languages* 1.ICFP (2017), с. 1—21.