1 Внутренние структуры данных

Далее обозначаем алфавит констант греческой буквой Σ, алфавит переменных — Ξ.

Базовые структуры данных алгоритма Ежа: переменная, константа и отрицательное условие.

- Поскольку множество переменных типа строка не меняется, то логично, что переменная представляется просто своим именем: например, x представляется как `(Var 'X')`.
- Константы могут являться результатами сжатия блоков, поэтому логично представлять их парами: имя и индекс. Например, `('D' 1)`.
- Индексы дополнительно имеют расшифровки: мультимножества компонент их длины. Например, если `('D' 1)` — константа, хранящая блок $D^{i_1+i_2+2}$, где D — константа исходного уравнения, то расшифровка будет:

  `('D' 1) is (('D' 0) (i1 1)(i2 1)(const 2))`.
- Отрицательное условие накладывается на переменную и имеет форму КНФ, где каждый литерал — это отрицательное утверждение, чем не может кончаться или начинаться переменная, или утверждение о непустоте значения переменной.

  Например,

```
(OR
    (not ('D' 1) ends (Var 'X'))
    (not ('A' 2) starts (Var 'Y'))
)
```

2 Базовые операции интерактивного режима

- Команда `(PairComp C1 C2)`. Применить сжатие пар:

  `<PairComp (/*Const1*/) (/*Const2*/) (/*EquationData*/)>`.

Константы не должны совпадать. Результатом сжатия пар станет нумерованный набор новых уравнений с условиями (таким образом, сжатие пар преобразует одно уравнение к множеству уравнений).

- Команда (`BlockComp C`). Применить сжатие блоков:

  `<BlockComp (/*Const*/) (/*EquationData*/)>`.

  Аналогично, результат — нумерованный набор новых уравнений с условиями.

- Команда (`Pick i`). Выбрать из нумерованного списка уравнение, соответствующее номеру: `<Pick /*Number*/ /*EquationSet*/>`.

  Действие позволяет перейти от множества уравнений к единственному.

- Команда (`Subst i1 (/* Multiset */)`). Осуществить подстановку мультимножества компонент, представленного в команде, вместо компоненты `i1`.

3 Возможные результаты вычисления шага

- Если в результате вычислений получилось уравнение вида $\varepsilon = \varepsilon$ или уравнение, которое сводится к нему после вычёркивания всех переменных, тогда объявить, что решение найдено.

- Если в результате вычислений получилось уравнение, содержащее только переменные, причём хотя бы одна переменная имеет условие непустоты, тогда объявить, что на ветке нет минимального решения (остались только неявные сжатия).

- Если команда противоречит условиям (например, требуется повторно сжимать блоки, с которых ничего начинаться не может, или требуется сжать блок, содержащий константу, не встречающуюся в уравнении) — выдать сообщение о некорректном шаге.

- В противном случае результатом вычислений станет одно или несколько состояний (т.е. уравнений в паре с условиями).

4 Построение новых констант в BlockComp и PairComp

Вопрос о том, ограничено ли множество используемых в процессе преобразования констант некоторой разумной (полиномиально зависящей от размера уравнения) величиной, пока открыт. Будем считать, что мы такое ограничение не ставим. Поэтому в дальнейшем нет смысла связывать новые имена констант со старыми, а лучше порождать их по очереди.

Чтобы код был написан полностью в функциональном стиле, достаточно хранить в общем состоянии последний используемый индекс блоков и последнюю использованную новую константу. Далее порождать на их основе очередную константу, например, пользуясь алфавитным сдвигом и комбинацией встроенных функций `<Explode s.X>` – `<Implode e.X>` и `<Numb e.N>` – `<Symb s.N>`.

- Функция `<Explode s.X>` разбирает многобуквенную константу на символы и возвращает строку: `Term -> 'Term'`.
- Функция `<Implode e.X>` собирает символы в многобуквенную константу (не любая последовательность символов может ей стать, выбирается самая длинная подпоследовательность значения `e.X`): `'Term 123' -> Term '123'`.
- Функция `<Symb s.X>` разбирает число на символы и возвращает строку: `123 -> '123'`. Функция `<Numb e.X>` действует обратным образом.

Простейший код для порождения новых индексов блоков всего лишь увеличивает индекс константы. Константой здесь считаем литерал (многобуквенный терм без кавычек), который начинается на букву, а следом за ней обязательно идёт цифра. Например, `i19` (не путаем со строкой `'i19'`).

```
GetNewIndex {
 s.OldIndex
 , <Explode s.OldIndex> : s.Letter e.Digits
 , <Symb <Add <Numb e.Digits> 1>> : e.NewNumber
  = <Implode s.Letter e.NewNumber>;
}
```

Улучшением этого базового варианта может быть вариант, который меняет сначала букву (выбирая в строке возможных вариантов ближайшую справа от текущей), а если не задействованных букв уже не осталось — то цифру, возвращая указатель на букву в начало алфавита.

```
/* В этом варианте есть тонкость: начальным буквам стоит
   присвоить индексы с нулём, а порождённым - начиная с
   единицы, чтобы не путаться. Реализацию этой тонкости
   оставим за кадром ТЗ. */
GetNewIndexUpdated {
 s.OldIndex
 , <Explode s.OldIndex> : s.Letter e.Digits
 , <CharAplhabet> : {
    e.A1 s.Letter s.NextLetter e.A2 /* Можно выбрать следующую
   по очереди букву */
        = <Implode s.NextLetter e.Digits>;
    s.FirstLetter e.A1 s.Letter /* Буква уже последняя ---
   переходим на первую и увеличиваем число */
    , <Symb <Add <Numb e.Digits> 1>> : e.NewNumber
        = <Implode s.FirstLetter e.NewNumber>;
  };
}
```

Заметим, что эти же приёмы можно использовать для порождения не только индексов блоков, но и букв — то есть пар вида `(s.Name s.Number)`. Всё, что нужно будет поменять — это не делать первый вызов функции `<Explode s.OldIndex>` с последующей сборкой символов в число (и обратное ему преобразование), а просто сдвигать букву `s.Name` и увеличивать числовой индекс `s.Number`.

5 Нулевые значения переменных

В дальнейшем подразумевается следующее допущение. Все извлечённые блоки по умолчанию полагаются непустыми — пустыми их может сделать только явная подстановка `(SubstIndex ik ((const 0)))` (или композиция таких подстановок), за которую отвечает пользователь. Более того, все извлечённые блоки по умолчанию предполагаются неравными

— в противном случае от пользователя ожидается подстановка, которая явно утверждает обратное.

Однако в случае вызова <PairComp /* args */> может получиться, что потребуется автоматически разобрать и случай пустой подстановки в переменную, чтобы выявить все возможности сжатия перекрёстных или явных вхождений. Например, как в следующем уравнении (назовём его Уравнением 1):

```
(AreEqual
   ((Var 'X') ('B' 0) (Var 'Y') ('A' 0))
   ((Var 'Y') (Var 'Y') ('B' 0) ('A' 0) (Var 'X'))
)
(
   (OR (not empty (Var 'X')))
)
(/* Блок уравнений на компоненты пуст */)
```

Вызов <PairComp ('B' 0) ('A' 0) (/* Уравнение 1 */)> порождает только разбор случаев по переменной Y. Полагаем, что новой константой, сжимающей ('B' 0)('A' 0), является ('A' 1).

1. Случай, когда Y кончается на ('B' 0) и начинается на ('A' 0):

   ```
   (AreEqual
      ((Var 'X') ('A' 1) (Var 'Y') ('A' 1))
      (('A' 0) (Var 'Y') ('A' 1) (Var 'Y') ('B' 0)('A' 1) (Var 'X'))
   )
   (
      (OR (not empty (Var 'X')))
   )
   ((('A' 1) is (('A' 0) (const 1))(('B' 0) (const 1))))
   ```

2. Случай, когда Y кончается на ('B' 0), но не начинается на ('A' 0):

   ```
   (AreEqual
      ((Var 'X') ('B' 0) (Var 'Y') ('A' 1))
      ((Var 'Y') ('B' 0) (Var 'Y') ('B' 0)('A' 1) (Var 'X'))
   )
   (
   ```

```
   (OR (not empty (Var 'X')))
   (OR (not ('A' 0) starts (Var 'Y')))
)
((('A' 1) is (('A' 0) (const 1))(('B' 0) (const 1))))
```

3. Случай, когда `Y` не кончается на `('B' 0)`, но начинается на `('A' 0)`:

```
(AreEqual
   ((Var 'X') ('A' 1) (Var 'Y') ('A' 0))
   (('A' 0) (Var 'Y') ('A' 0) (Var 'Y') ('A' 1) (Var 'X'))
)
(
   (OR (not empty (Var 'X')))
   (OR (not ('B' 0) ends (Var 'Y')))
)
((('A' 1) is (('A' 0) (const 1))(('B' 0) (const 1))))
```

4. Случай, когда `Y` не кончается на `('B' 0)` и не начинается на `('A' 0)`:

```
(AreEqual
   ((Var 'X') ('B' 0) (Var 'Y') ('A' 0))
   ((Var 'Y') (Var 'Y') ('A' 1) (Var 'X'))
)
(
   (OR (not empty (Var 'X')))
   (OR (not ('A' 0) starts (Var 'Y')))
   (OR (not ('B' 0) ends (Var 'Y')))
)
((('A' 1) is (('A' 0) (const 1))(('B' 0) (const 1))))
```

5. Случай, когда `Y` пуст:

```
(AreEqual
   ((Var 'X') ('A' 1))
   (('A' 1) (Var 'X'))
)
(
   (OR (not empty (Var 'X')))
```

```
)
(/* Блок уравнений на компоненты пуст */)
```

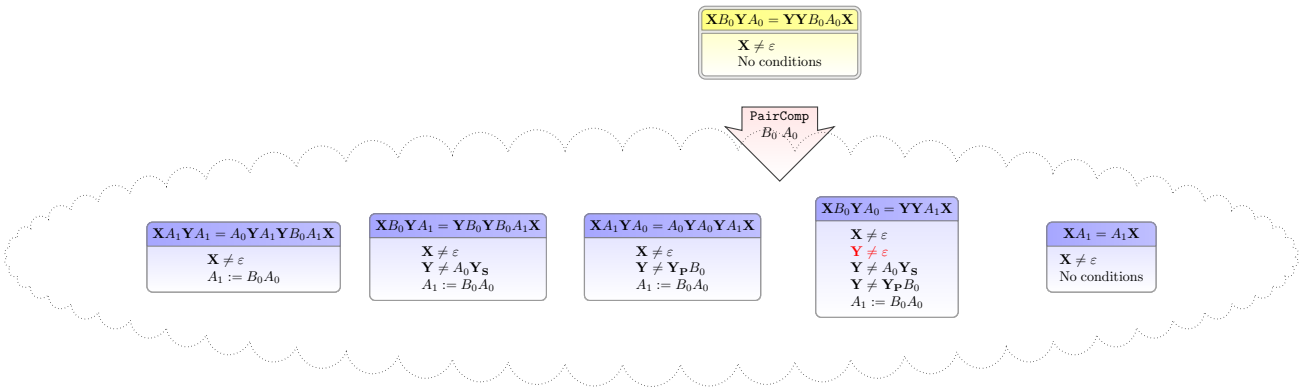

Рис. 1: Пустая подстановка при сжатии пар

Все подстановки показаны на Рисунке 1.

Последний случай принципиально отличается от всех предыдущих и не может быть сведён к ним. Кажется, что он похож на случай 4, но в случае 4 есть очевидное противоречие, если рассмотреть его образ в линейной целочисленной арифметике по модулю A_1, поскольку предполагается, что буквы B_0 и A_0, явно входящие в уравнение, уже точно не принадлежат никакому вхождению B_0A_0. В случае же 5 противоречий нет, и он является единственным, приводящим к решению (после подстановки сжатия блоков A_1).

6 Извлечение пар в PairComp

Перекрёстные и явные пары в `PairComp`(γ_1,γ_2) могут появляться вследствие следующих подстановок.

- элементарная подстановка $X \mapsto X\gamma_1$, $X \mapsto \gamma_2 X$;
- композиция двух элементарных подстановок (например, при извлечении $\gamma_1\gamma_2$) из границы между X и Y;
- пустая подстановка $X \mapsto \varepsilon$ (см. выше).

Скажем, что элементарная подстановка существенная, если в результате её применения (без иных действий) появляется новая явная пара $\gamma_1\gamma_2$.

Скажем, что композиция τ_i существенная, если в результате её применения появляется явная пара, которая не появляется при применении элементов τ_i по отдельности.

Пусть $\tau = \sigma_2 \circ \sigma_1$ существенна (все подстановки независимые, порядок может быть любым, существенны лишь сами σ_i). Возможны следующие случаи:

- σ_1 и σ_2 тоже существенны. Тогда относительно σ_i придётся перебрать все четыре варианта комбинаций.

- σ_1 существенна, σ_2 нет. Тогда вариант применения только σ_2 и вариант применения ни одной из подстановок эквиваленты (и описываются отрицательной рестрикцией на σ_1), и нет смысла выделять оба. Остаются только три варианта из четырёх.

- ни одна из компонент не существенна. Тогда случаи их применения по отдельности оба эквивалентны отсутствию подстановок (и соответствуют отрицательной рестрикции, представляющей собой дизъюнкцию), и остаётся только два варианта.

После построения базовых вариантов по каждой из композиций, а также базовых вариантов по существенным элементарным подстановкам, не вошедшим ни в одну композицию, строятся прямые произведения. При этом некоторые из произведений будут отброшены как повторные, а некоторые — как противоречивые; кроме того, могут происходить нетривиальные преобразования навешиваемых отрицательных условий (упрощение 2КНФ). Всю логику 2-КНФ здесь реализовывать нет необходимости, потому что все элементы дизъюнкций, состоящих больше, чем из одного дизъюнкта, отрицательные. Поэтому достаточно «протолкнуть» (т.е. подставить) все положительные значения подстановок в дизъюнкты (для упрощения и нахождения противоречий); и выбросить 2-дизъюнкты, которые поглощаются 1-дизъюнктами.

Рассмотрим, например, уравнение `bXYa = XbZY`, к которому применяется `PairComp(ab)`. Выделяется существенная подстановка σ_1, $\mathtt{X}\sigma_1 = \mathtt{Xa}$, а также элементы композиций σ_2 и σ_3: $\mathtt{Y}\sigma_2 = \mathtt{bY}$, $\mathtt{Z}\sigma_3 = \mathtt{Za}$. По композиции $\sigma_2 \circ \sigma_1$ получается три случая.

- (1-1) X=Xa&Y=bY;
- (1-2) X=Xa&Y≠bY;
- (1-3) X≠Xa

По композиции $\sigma_2 \circ \sigma_3$ случая всего два.

- (2-1) Z=Za&Y=bY;
- (2-2) Z≠Xa∨Y≠bY.

Рассмотрим все их произведения.

- (1-1)×(2-1) X=Xa&Y=bY&Z=Za;
- (1-1)×(2-2) X=Xa&Y=bY&Z≠Xa (здесь дизъюнкция вынуждает выполниться первый дизъюнкт, т.к. второй вступает в противоречие с положительным условием);
- (1-2)×(2-1) противоречие;
- (1-2)×(2-2) X=Xa&Y≠bY. А здесь дизъюнкция тривиально выполнена, поэтому добавлять её не нужно.
- (1-3)×(2-1) X≠Xa&Z=Za&Y=bY;
- (1-3)×(2-2) X≠Xa&(Z≠Xa∨Y≠bY).

Осталось пять вариантов (Рисунок 2).

6.1 Существенные пустые подстановки

Скажем, что подстановка η :Y↦ ε существенная, если в результате её применения становится возможно построить новую явную или перекрёстную пару.

Скажем, что композиция пустых подстановок θ существенная, если в результате её применения становится возможно построить новую явную или перекрёстную пару, которая не строится при применении любого подмножества подстановок, участвующих в композиции.

Определим критерии существенности пустых подстановок и композиции пустых подстановок при сжатии пары $\gamma_1\gamma_2$.

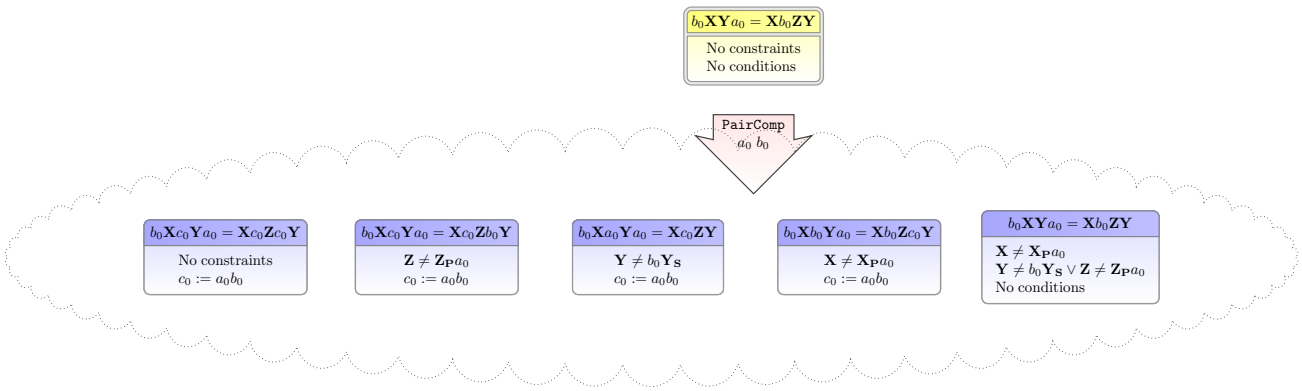

Рис. 2: Порождение случаев при сжатии пар

- Пусть часть уравнения имеет вид $\Phi_1\gamma_1\Psi\gamma_2\Phi_2$, где Ψ непуста и принадлежит алфавиту переменных Ξ. Тогда композиция подстановок в Ψ, обращающая Ψ в ε, считается существенной. Пример: в уравнении `YbZaYZb` = `XaYWYZ` при сжатии `ab` композиция $\theta_1 = \eta_1 \circ \eta_2$, где η_1 : `Y` $\mapsto \varepsilon$, η_2 : `Z` $\mapsto \varepsilon$, существенна, поскольку порождает новую явную пару, выделенную красным.

- Пусть часть уравнения имеет вид $\Phi_1\gamma_1\Psi X\Phi_2$, где X — переменная, а Ψ непуста и принадлежит алфавиту $\Xi \setminus \{X\}$. Тогда композиция подстановок в Ψ, обращающая Ψ в ε, считается существенной. Пример: в уравнении `YbZaYZb` = `XaYWYZ` при сжатии `ab` композиция $\theta_2 = \eta_1 \circ \eta_3$, где η_1 : `Y` $\mapsto \varepsilon$, η_3 : `W` $\mapsto \varepsilon$, существенна, поскольку порождает новую перекрёстную пару, если положить `Z` $\mapsto$ `bZ`. Аналогичным свойством обладает и подстановка η_1 сама по себе: она порождает уравнение `bZaZb = XaWZ`, где появляется новое соседство буквы `a` с переменными `Z` и `Y` справа от неё.

- Критерий, симметричный предыдущему. Пусть часть уравнения имеет вид $\Phi_1 X\Psi\gamma_2\Phi_2$, где X — переменная, а Ψ непуста и принадлежит алфавиту $\Xi \setminus \{X\}$. Тогда композиция подстановок в Ψ, обращающая Ψ в ε, считается существенной. В том же уравнении `YbZaYZb` = `XaYWYZ` при сжатии `ab` подстановка η_2 существенна в этом смысле.

- Последний случай: новое соседство между переменными. Пусть часть уравнения имеет вид $\Phi_1 X_1\Psi X_2\Phi_2$, где X_i — переменные (возмож-

но, равные), а Ψ непуста и принадлежит алфавиту $\Xi \setminus \{X_1, X_2\}$. Тогда композиция подстановок в Ψ, обращающая Ψ в ε, считается существенной. В уравнении $\texttt{YbZaYZb} = \texttt{XaYWYZ}$ при сжатии $\texttt{ab}$ существенными по этому критерию будут подстановки η_1 (порождает смежную пару, выделенную красным) и η_3 (порождает смежную пару, выделенную синим).

Кажется, что вышеперечисленные случаи порождают огромное число композиций, в которых, в том числе, будут отрицательные условия дизъюнктами не в 2-КНФ. Однако можно заметить[1], что, в отличие от случая непустых подстановок, если композиция θ является существенной, то все подстановки в ней являются существенными. Поэтому имеет смысл максимизировать фрагменты Ψ, обладающие требуемыми свойствами, чтобы выделить из них все необходимые для разбора случаев пустые подстановки, и дальше — комбинировать их между собой всеми способами и применять к результатам соответствующих подстановок уже только перебор по непустым подстановкам. Это приведёт к определённого рода избыточности: например, комбинация $\eta_1 \circ \eta_2 \circ \eta_3$ порождает уравнение $\texttt{bab} = \texttt{Xa}$, которое является частным случаем уравнения $\texttt{bab} = \texttt{XaW}$ при условии, что $\texttt{W}$ не начинается с $\texttt{b}$. Кажется, такая избыточность излечима, если пользоваться леммой и запускать перебор существенно пустых подстановок по одной, рекурсивно[2].

7 Модификации в блоке рестрикций

Скажем, что рестрикция избыточная, если в ней участвует константа, которой нет в уравнении, а также в блоке уравнений на индексы в правых частях. После модификации уравнения, нужно удалить избыточные рестрикции. С остальными предлагается действовать следующим образом.

- Если в блоке была рестрикция $X \neq \alpha_1 X$, то в случае, если $\alpha_1 \in \mathrm{First}(\gamma_2)$ (см. следующий раздел), подстановка $X \mapsto \gamma_2 X$ считается

[1] И потом доказать — это TODO остаётся за мной, но если вы хотите, то можете попробовать.

[2] При этом на все переменные, которые не обращаются в пустое слово на данном рекурсивном шаге, нужно объявлять непустыми в дополнительной рестрикции. Такая рестрикция выделена красным на Рисунке 1.

невозможной — больше того, добавлять дополнительное условие $X \neq \gamma_2 X$ на данную ветку уже нет необходимости. В противном случае подстановка $X \mapsto \gamma_2 X$ уничтожает рестрикцию $X \neq \alpha_1 X$.

- Аналогичное условие выполняется для суффиксных рестрикций.
- Рестрикция $X \neq \varepsilon$ уничтожается при любой подстановке в X ($X \mapsto \gamma_2 X$, $X \mapsto X\gamma_1$, а также их комбинации).
- Если $\beta = \gamma_2^s \Psi$, и добавляется рестрикция $X \neq \gamma_2 X$, тогда имеет смысл также добавить рестрикцию $X \neq \beta X$. Симметричное условие выполнено для суффиксов. (oudated, храним только самые сильные отрицательные рестрикции, в данном случае это $X \neq \gamma_2 X$)

8 Извлечение блоков

В этом варианте по каждой переменной без учёта рестрикций ровно два случая при операции `BlockComp` α:

- сжатие в единственный блок $X \mapsto \alpha^i$;
- извлечение двух блоков слева и справа, $X \mapsto \alpha^{i_1} X \alpha^{i_2}$, с дополнительными условиями $X \neq \varepsilon$, $X \neq \alpha X$, $X \neq X\alpha$.

Напомним, что в дальнейшем все различные блоки (собранные из идущих подряд блоков α) переводятся в различные новые буквы. Например, при сжатии блоков a_0 выражение $a_0 a_0^{i_1} X a_0^{i_1} a_0^{i_2} X a_0^{i_2} a_0 a_0$ индуцирует замены $a_1 := a_0^{i_1+1}$, $b_1 := a_0^{i_1+i_2}$, $c_1 := a_0^{i_2+2}$. В итоге после сжатия выражение будет представлено как $a_1 X b_1 X c_1$, а указанные выше подстановки попадут в блок уравнений на индексы.

Хотя новые буквы различны по умолчанию (если только уравнения на индексы не совпадут буквально), операция подстановки в переменные-индексы в дальнейшем может сделать их равными.

В отрицательных условиях допускаем использование исходной константы (в нашем примере a_0). В принципе, можно было бы заменить эти условия множествами условий на a_1, b_1, c_1, но во-первых, это породит в перспективе (похоже что) экспоненциальный рост числа рестрикций, а во-вторых, возникает тонкость с возможностью вырождения индекса в

нулевой, что влечёт появление абсурдной рестрикции «не начинается с ε».

Можно было бы возразить, что такая рестрикция может всё равно неявно появиться, если сжатие уже не первое (сжали блок букв a_0, получили a_1; затем сжали блок букв b_0, получили b_1; сжали пару a_1b_1, получили c_1, но если a_1 и b_1 оба были нулевыми степенями букв, то извлечение блоков букв c_1 заведомо абсурдно). Второй похожий вопрос состоит в том, что если a_1 и b_1 оба являются блоками a_0, то допускать ли возможность, что пары a_1b_0 и b_1b_0 должны сжиматься в одну и ту же константу.

Решается этот вопрос следующим соглашением.

- Если в блок сжимается составная константа α, то все её компоненты заведомо предполагаются непустыми. Если хотелось проверить пустую подстановку, то пользователь должен был объявить об этом до сжатия, анонсировав соответствующую подстановку индексов.
- Если составная константа α является элементом сжатой пары, то она не совпадает ни с одной из других составных констант.

На практике эти соглашения просто означают, что никакие дополнительные проверки на равенство и пустоту сжатых блоков при дальнейших операциях сжатия не нужны.

8.1 Обработка рестрикций в BlockComp

Скажем, что $\alpha_0 \in \mathrm{First}(\alpha)$, если существует цепочка условий на индексы вида $\alpha = \gamma_1^{i_1}\Phi_1$, $\gamma_1 = \gamma_2^{i_2}\Phi_2$, $\ldots$, $\gamma_n = \alpha_0^{i_{n+1}}\Phi_{n+1}$.

Симметрично определим понятие $\mathrm{Last}(\alpha)$. В дальнейшем будем оперировать понятием First, по умолчанию считая, что для двойственного ему все действия и утверждения будут симметричны.

Заметим, что если $\alpha_1 \in \mathrm{First}(\alpha)$ и $\alpha_2 \in \mathrm{First}(\alpha)$, тогда верна дизъюнкция $\alpha_1 \in \mathrm{First}(\alpha_2) \vee \alpha_2 \in \mathrm{First}(\alpha_1)$, поскольку отношение «быть первой буквой» строго иерархично.

Это условие не рекомендуется понимать в смысле «α_0 является префиксом α», поскольку, например, если $a_1 := a_0^2$, $a_2 := a_0^3$ (что возможно при сжатии выражения $XbaabXbaaa$), то строго говоря, a_1 формирует слово-префикс a_2, однако неверно, что $a_1 \in \mathrm{First}(a_2)$. Разница здесь в том, что при выделении максимальных блоков подразумевается, что блоки точно имеют границы, не позволяющие их смешивать, поэтому там,

где встретилась a_1, за ней не может следовать никакая буква, содержащая a_0 в First-множестве, и поэтому наборы отделённых 2-блоков a_0 и отделённых 3-блоков a_0 могут рассматриваться как принципиально различные буквы. Это одна из ключевых черт алгоритма Ежа, в том числе, гарантирующая то, что он может порождать только линейные диофантовы уравнения на длины блоков.

Рестрикции учитываются следующим образом.

- Если сжимаются блоки α, и существует рестрикция вида $X \neq \alpha_0 X$, где $\alpha_0 \in \mathrm{First}(\alpha)$, тогда отображение «сжатие в блок» $X \mapsto \alpha^i$ заменяется на пустое: $X \mapsto \varepsilon$; отображение «извлечение блоков слева и справа» $X \mapsto \alpha^{i_1} X \alpha^{i_2}$ заменяется на отображение «извлечение только правого блока» $X \mapsto X\alpha^{i_2}$ (и рестрикция $X \neq \alpha_0 X$ остаётся, поскольку она сильнее, чем налагаемая сжатием новая рестрикция $X \neq \alpha X$).

- Если рестрикция $X \neq \beta X$ независима от α (то есть $\beta \notin \mathrm{First}(\alpha)$), тогда при сжатии в единый блок она полностью пропадает, а при извлечении блоков необходимо расщепить его на два случая: $X \mapsto X\alpha^{i_2}$ с сохранением рестрикции $X \neq \beta X$ (а также добавлением $X \neq \alpha X$, как обычно), и $X \mapsto \alpha^{i_1} X \alpha^{i_2}$ с удалением рестрикции $X \neq \beta X$. Это необходимо, потому что в противном случае потеряется условие, что если префиксный блок пуст, то за ним следует точно не буква β или производные от неё.

- Рестрикции на последние буквы обрабатываются симметричным образом.

- Рестрикция $X \neq \varepsilon$ влияет только на сжатие в единый блок α^i: в этом случае она уничтожается с порождением подстановки $i := i'+1$ (i' — свежий индекс), которая немедленно применяется. Если сжимаются блоки-префиксы и блоки-суффиксы, то $X \neq \varepsilon$ «формально уничтожается» с появлением точно такой же рестрикции, являющейся результатом уже конкретно этого сжатия.

- В заключение, рестрикции на избыточные константы (в смысле предыдущего раздела) следует удалить из условий.

9 Пример

Покажем пример работы цепочки операций на следующем примере. Обработка уравнений на индексы и рестрикций оптимизирована согласно последнему разделу об операции `BlockComp`.

Исходное уравнение:

```
(AreEqual
 ((Var 'X') ('A' 0) ('A' 0))
 (('B' 0) (Var 'Y'))
)
(/* Блок условий пуст */)
(/* Блок уравнений на компоненты пуст */)
```

- Применяем `(BlockComp ('A' 0))`. Получаем набор уравнений:

  1. Случай, когда все переменные коллапсируют в блоки:

     ```
     (AreEqual
      (('A' 1))
      (('B' 0) ('A' 2))
     )
     (/* Блок условий пуст */)
     (
      (('A' 1) is ('A' (i1 1) (const 2)))
      (('A' 2) is ('A' (i2 1) (const 0)))
     )
     ```

  2. В блок коллапсирует только `X`:

     ```
     (AreEqual
      (('A' 1))
      (('B' 0) ('A' 2) (Var 'Y') ('A' 3))
     )
     (
       (OR (not empty (Var 'Y')))
       (OR (not ('A' 0) ends (Var 'Y'))
       (OR (not ('A' 0) starts (Var 'Y'))
     )
     /* Здесь использованы те же свежие индексы и имена
     ↪  констант, что и в предыдущем случае.
     ```

```
Это допустимо, поскольку ветки независимы - и поэтому
↪  им можно передавать один и тот же последний
↪  неиспользованный индекс. */
(
 (('A' 1) is (('A' 0) (i1 1) (const 2)))
 (('A' 2) is (('A' 0) (i2 1) (const 0)))
 (('A' 3) is (('A' 0) (i3 1) (const 0)))
)
```

3. В блок коллапсирует только `Y`:

```
(AreEqual
 (('A' 1) (Var 'X') ('A' 2))
 (('B' 0) ('A' 3))
)
(
  (OR (not empty (Var 'X')))
  (OR (not ('A' 0) ends (Var 'X'))
  (OR (not ('A' 0) starts (Var 'X'))
)
(
 (('A' 1) is (('A' 0) (i1 1) (const 0)))
 (('A' 2) is (('A' 0) (i2 1) (const 2)))
 (('A' 3) is (('A' 0) (i3 1) (const 0)))
)
```

4. Ничего не коллапсирует:

```
(AreEqual
 (('A' 1) (Var 'X') ('A' 2))
 (('B' 0) ('A' 3) (Var 'Y') ('A' 4))
)
(
  (OR (not empty (Var 'X')))
  (OR (not empty (Var 'Y')))
  (OR (not ('A' 0) ends (Var 'X'))
  (OR (not ('A' 0) starts (Var 'X'))
  (OR (not ('A' 0) ends (Var 'Y'))
  (OR (not ('A' 0) starts (Var 'Y'))
)
 (('A' 1) is (('A' 0) (i1 1) (const 0)))
```

```
 (('A' 2) is (('A' 0) (i2 1) (const 2)))
 (('A' 3) is (('A' 0) (i3 1) (const 0)))
 (('A' 4) is (('A' 0) (i4 1) (const 0)))
)
```

- Выбираем четвёртый случай: `(Pick 4)`. Осуществляем ряд подстановок индексов:

  ```
  (Subst i1 ((const 0)))
  ```

  ```
  (Subst i3 ((const 0)))
  ```

  ```
  (Subst i4 ((i2 1) (const 2)))
  ```

  Интерактивный режим сам сократит одинаковые суффиксы и удалит ненужные зависимости, ведь констант `('A' k)` после этого в уравнении не останется:

  ```
  (AreEqual
   ((Var 'X'))
   (('B' 0) (Var 'Y'))
  )
  (/* Тут ничего нет, кроме условий на непустоту:
      все упоминания констант, связанных с 'A',
      исчезли из уравнения, и нет никаких связей этих
      констант ни с чем, кроме как с собой, в блоке
      индексов, значит, и условия на них уже не нужны */
    (OR (not empty (Var 'X')))
    (OR (not empty (Var 'Y')))
   )
  (/* Тут тоже ничего нет --- всё сократилось */)
  ```

- Сжимаем блоки `('B' 0)`. Опять получаем четыре случая.

  1. Случай, когда все переменные коллапсируют в блоки:

     ```
     (AreEqual
      (('B' 1))
      (('B' 2))
     )
     (/* Блок условий пуст */)
     ```

```
 /* Условие на непустоту добавило 1 в уравнения для
 ↪  обеих констант */
(
 (('B' 1) is ('B' (i5 1) (const 1)))
 (('B' 2) is ('B' (i6 1) (const 2)))
)
/* Счёт индексов двинулся дальше. Формально, мы могли
 ↪  бы без проблем переиспользовать i1-i4, поскольку
 ↪  их уже нет в уравнении, но это усложнило бы логику
 ↪  работы функции порождения свежих констант и
 ↪  индексов, поэтому проще взять свежие. */
```

2. В блок коллапсирует только `X`:

```
(AreEqual
 (('B' 1))
 (('B' 2) (Var 'Y') ('B' 3))
)
(
  (OR (not empty (Var 'Y')))
  (OR (not ('B' 0) ends (Var 'Y'))
  (OR (not ('B' 0) starts (Var 'Y'))
)
/* Условие на непустоту добавило 1 в уравнение для
 ↪  блока по X */
(
 (('B' 1) is (('B' 0) (i5 1) (const 1)))
 (('B' 2) is (('B' 0) (i6 1) (const 1)))
 (('B' 3) is (('B' 0) (i7 1) (const 0)))
)
```

3. В блок коллапсирует только `Y`:

```
(AreEqual
 (('B' 1) (Var 'X') ('B' 2))
 (('B' 3))
)
/* Условие на непустоту добавило 1 в уравнение для
 ↪  блока по Y */
(
  (OR (not empty (Var 'X')))
```

```
  (OR (not ('B' 0) ends (Var 'X'))
  (OR (not ('B' 0) starts (Var 'X'))
)
(
 (('B' 1) is (('B' 0) (i5 1) (const 0)))
 (('B' 2) is (('B' 0) (i6 1) (const 0)))
 (('B' 3) is (('B' 0) (i7 1) (const 2)))
)
```

4. Ничего не коллапсирует:

```
(AreEqual
 (('B' 1) (Var 'X') ('B' 2))
 (('B' 3) (Var 'Y') ('B' 4))
)
(
  (OR (not empty (Var 'X')))
  (OR (not empty (Var 'Y')))
  (OR (not ('B' 0) ends (Var 'X'))
  (OR (not ('B' 0) starts (Var 'X'))
  (OR (not ('B' 0) ends (Var 'Y'))
  (OR (not ('B' 0) starts (Var 'Y'))
)
 (('B' 1) is (('B' 0) (i5 1) (const 0)))
 (('B' 2) is (('B' 0) (i6 1) (const 0)))
 (('B' 3) is (('B' 0) (i7 1) (const 1)))
 (('B' 4) is (('B' 0) (i8 1) (const 0)))
)
```

- Пусть далее рассматривается опять ветка 4: `(Pick 4)`, и аналогичные рассмотренным ранее подстановки:

  ```
  (Subst i5 ((i7 1) (const 1)))
  ```

  ```
  (Subst i6 ((i8 1) (const 0)))
  ```

  После такого преобразования останется только уравнение

  ```
  (AreEqual ((Var 'X')) ((Var 'Y')))
  ```

  с условиями непустоты, и интерактивный режим должен сообщить, что минимальное решение на этой ветке не существует (т.к. остались только переменные).

- Если бы рассматривалась первая ветка, то следующая подстановка:
  `(Subst i5 ((i6 1) (const 1)))`
  привела бы к сообщению о нахождении решения.