1 Теоретическая основа

Для описания структур данных используется *формальная* БНФ-грамматика. Все литералы имеют один из трёх префиксов: `s.`, `t.` или `e.`, соотвествующие *символам*, *термам* и *объектным выражениям* языка Рефал.

Помимо явных символьных слов (`AreEqual`, `Var` и т.п.) используются следующие терминалы:

- `s.NUMBER` — любая макроцифра;
- `s.CHAR` — любой символ;
- `s.WORD` — любое символьное слово.
- `e.ANY` — произвольное объектное выражение.

Символы `*` и `+` означают повторение литерала ноль и более и один и более раз соответственно.

Помимо `::=` используется инфиксный оператор `:`, означающий "выражение слева представляется как выражение справа".

1.1 Представление уравнения

Рассматриваются уравнения в словах с алфавитом переменных Ξ и алфавитом констант Σ. Применение алгоритма Jez'а предполагает хранение дополнительной информации об уравнении: какие константы являются результатом сжатия блоков, какие переменные не могут быть пусты и т.д. Поэтому *уравнение* представляется структурой данных `t.Eq`:

```
t.Eq ::= ((AreEqual (t.Elem*) (t.Elem*)) (t.Constr*) (t.Cond*)),
```

где выражения `(t.Elem*)` представляют левую и правую части уравнения, `(t.Constr*)` — ограничения на переменные, а `(t.Cond*)` содержит условия на константы.

Элемент `t.Elem` обобщённо представляет *константу* или *переменную*, соответствующие структурам данных `t.Const` и `t.Var`:

```
t.Elem ::= t.Var | t.Elem,
t.Var ::= (Var s.CHAR),
t.Const ::= (s.CHAR s.NUMBER).
```

Назовём **сокращением** уравнения удаление всех совпадающих префиксных и суффиксных элементов его левой и правой частей. В результате получаем **сокращённое** уравнение.

Два уравнения будем называть **эквивалентными**, если их сокращение производит уравнения с одинаковыми левыми и правыми частями.

1.2 Ограничения на переменные

Ограничения на переменные представляются структурой `t.Constr` и описываются в конъюнктивной нормальной форме. Литералами дизъюнкций являются *рестрикции* (отрицательные условия на переменные) типа `t.Restr`, подразделяющиеся на краевые `t.BoundRestr` и рестрикции на пустоту `t.EmptyRestr`:

```
t.Restr ::= t.BoundRestr | t.EmptyRestr.
```

Краевые рестрикции бывают *префиксными* `t.PrefixRestr` и *суффиксными* `t.SuffixRestr`. Они указывают, на какие константы не может начинаться или кончаться данная переменная.

```
t.BoundRestr ::= t.PrefixRestr | t.SuffixRestr,
t.PrefixRestr ::= (not t.Const starts t.Var),
t.SuffixRestr ::= (not t.Const ends t.Var).
```

Рестрикция на пустоту `t.EmptyRestr` сообщает, как следует, о невозможности обращения данной переменной в пустое слово ε:

```
t.EmptyRestr ::= (not empty t.Var).
```

В частности, будем называть переменную **непустой**, если для неё существует такая рестрикция.

Ограничения на переменные `t.Constr` могут содержать одну или две рестрикции, в соответствии с чем называются *тривиальными* `t.TrivialConstr` и *нетривиальными* `t.NonTrivialConstr`. В программе используется только четыре вида ограничений:

```
t.Constr ::= t.TrivialConstr | t.NonTrivialConstr
t.TrivialConstr ::= (OR t.Restr),
t.NonTrivialConstr ::= (OR t.SuffixRestr t.PrefixRestr).
```

Говоря далее ограничения мы, как правило, будем подразумевать именно тривиальные, если не оговорено противное, а также будем экстраполировать тип рестрикции на ограничение её содержащее: префиксное ограничение, ограничение на пустоту и т.д.

1.3 Условия на константы

Условие на константу `t.Cond` содержит сжимаемые блоки и соответствующую этому сжатию константу. Вообще, *блок* `t.Block` обозначает степень константы и представляется в виде

```
t.Block ::= (t.Const t.Exp* (const s.NUMBER)),
```

где `t.Const` — сжимаемая константа, `(const s.NUMBER)` — обязательный константный показатель, а `t.Exp ::= (s.WORD s.NUMBER)` — переменный показатель степени. Таким образом, условие на константу представляется в виде

```
t.Cond ::= (t.Const is t.Block+).
```

В программе используется только два типа условий: *парное условие* `t.PairCond` и *условие на блок* `t.BlockCond`:

```
t.PairCond ::= (t.Const is (t.Const (const 1)) (t.Const (const 1))),
t.BlockCond ::= (t.Const is t.Block).
```

1.4 Краевые элементы

Мы хотим применять `Pair`- и `Block`-сжатие не только к тривиальным константам, но и тем, что уже являются результатом сжатия в блоки. Для корректной обработки ограничений необходимо аккуратно отслеживать, на какие константы не может начинаться и кончаться та или иная переменная.

Пусть $\alpha, \gamma \in \Sigma$. Скажем, что $\alpha \in First(\gamma)$, если существует цепочка условий на константы такая, что

$$\gamma = \beta_1^{i_1} B_1, \beta_1 = \beta_2^{i_2} B_2, \ldots, \beta_{n-1} = \alpha^{i_n} B_n,$$

где $\beta_i \in \Sigma, i = 1, 2, \ldots, n-1$, и $B_j \subset \Sigma, j = 1, 2, \ldots, n$. Симметрично определяется множество $Last$-элементов константы. Вообще, делая далее какое-либо утверждение о $First$-элементах константы будем считать, что оно симетрично выполняется и для $Last$-множества.

Иногда удобно использовать обобщение введённых выше понятий: будем говорить, что константа α является **краевым** элементом константы γ, если $\alpha \in First(\gamma)$ или $\alpha \in Last(\gamma)$. Обозначение: $\alpha \in Bound(\gamma)$.

1.5 Слабые и сильные ограничения

В соответствие с введённым выше понятием краевого элемента, для данной переменной X и константы γ мы можем ввести *иерархические отношения* на множествах префиксных и суффиксных ограничений X с константами $First(\gamma)$ и $Last(\gamma)$ соответственно.

Пусть даны два, например, префиксных ограничения с константами α и β (для суффиксных — аналогично). Если $\alpha \in First(\beta)$, то α-ограничение является **более сильным**. В то же время β-ограничение есть **более слабое** по сравнению с первым. Имеет смысл хранить в уравнении лишь самые сильные ограничения.

Пусть, например, уравнение содержит ограничения

```
t.Constr1: (OR (not ('A' 1) ends (Var 'X')),
t.Constr2: (OR (not ('C' 0) ends (Var 'X'))
```

и условие

```
t.Cond: (('A' 1) is (('B' 0) (const 1)) (('C' 0) (const 1))).
```

Тогда в уравнении можно оставить только `t.Constr2`, так как оно сильнее ограничения `t.Constr1`

1.6 Избыточные рестрикции и условия

В результате некоторых действий краевые рестрикции могут стать неактуальными (с рестрикциями на пустоту такого не происходит), вследствие чего их можно удалить из уравнения. Будем называть (краевую) рестрикцию **избыточной** в двух случаях:

- участвующей в ней переменной нет в уравнении;
- участвующей в ней константы нет в уравнении, а также в правых частях условий на константы.

Условие на константу также может стать **избыточным**. Это происходит в следующих случаях:

- константа в левой части условия не участвует в уравнении и не является *Bound*-элементом какой-либо другой константы;
- условие имеет парный тип, обеих констант его правой части нет в уравнении и на них нет каких-либо других условий.

1.7 Обработка нетривиальных ограничений

Нетривиальные ограничения на переменные также могут подвергаться обработке. Пусть уравнение содержит

```
t.NonTrivialConstr: (OR t.SuffixRestr t.PrefixRestr).
```

Если в уравнении найдётся более сильное в сравнении с `(OR t.SuffixRestr)` или `(OR t.PrefixRestr)` краевое ограничение, то `t.NonTrivialConstr` **выполняется тривиально**, и его можно удалить. Это же происходит и в случае, когда хотя бы одна из рестрикций `t.SuffixRestr` и `t.PrefixRestr` избыточна.

Будем называть префиксную рестрикцию с константой $\alpha \in \Sigma$ **зависимой** относительно некоторого $\beta \in \Sigma$, если $\alpha \in First(\beta) \cup \beta$. В противном случае рестрикция называется **независимой**.

Иногда нам потребуется модифицировать нетривиальные ограничения. Например, мы хотим выполнить подстановку $X \to X\alpha, X \in \Xi, \alpha \in \Sigma$, при наличии такого ограничения с зависимой относительно α рестрикцией. Мы **вынуждаем** выполнение второй рестрикции, превращая тем самым нетривиальное ограничение в тривиальное.

1.8 Нормальное уравнение

Будем называть уравнение **нормальным**, если оно несократимо, не содержит слабых и тривиально выполняющихся ограничений и избыточных рестрикций и условий. Также потребуем, чтобы в нормальном уравнении все ограничения были отсортированы, равно как и показатели степеней `t.Exp` условий на блок. В практической реализации используется быстрая сортировка, принимающая функцию-компаратор. Определив такие комапараторы для ограничений и показателей степеней появляется возможность эффективно сортировать указанные элементы.

Нормальные уравнения представляют особый интерес для нас. Именно такие уравнения возращают и ожидают получить на вход функции `Pick`, `SubstIndex`, `PairComp` и `BlockComp`.

2 Функция `Pick`

Скажем, что решение уравнения **неминимальное**, если его левая и правая части состоят из непустых переменных и только них.

Функция `Pick` принимает макроцифру `s.NUMBER` и уравнения `t.Eq+`. Выбрав уравнение под номером `s.NUMBER`, подставляет все его пустые переменные в ε. Функция возращает

- `Success`, если новое уравнение эквивалентно уравнению $\varepsilon = \varepsilon$;
- `NotMinimal`, если решение такого уравнения неминимальное;
- новое нормальное уравнение в остальных случаях.

Здесь и далее часто требуется выполнять *нерекурсивные подстановки* в выражение. Определим для этого специальный формат `t.Subst`:

```
t.Subst ::= (assign (e.ANY) (e.ANY)),
```

где первое выражение `e.ANY` — заменяемое, а второе — новое.

`Pick` имеет довольно простую реализацию: из множества всех переменных уравнения вычитаются непустые. Для оставшихся переменных генерируются и выполняются подстановки в ε. Если получившееся уравнение эквивалентно $\varepsilon = \varepsilon$, возвращаем, как говорилось, `Success`. Иначе получаем множество констант нового уравнения и возвращаем `NotMinimal`, если оно пусто, и получившееся уравнение с удалёнными (в связи с пустотой) избыточными рестрикциями в противном случае.

3 SubstIndex

Функция `SubstIndex` принимает индекс `s.Index: s.WORD`, показатели `e.Exps: t.Exp+`, уравнение `t.Eq` и выполянет подстановку показателей на место данного индекса.

Для каждого условия на блок с показателем (`s.Index s.NUMBER`) заменим этот показатель на множество `e.Exps`, домноженное на постоянную `s.NUMBER`. Обычно после этого шага порядок показателей нарушается — их необходимо отсортировать.

В результате подстановки мог получиться блок вида

```
(t.Const (const 0))
```

Такой блок *коллапсирует*, а вместе с ним коллапсирует и содержавшее его условие (вообще, подстановка показателей возможна только в условия на блок, которые всегда содержат степень единственного сжатого элемента). Нам полезно на месте генерировать подстановки констант из левых частей таких условий в ε: позже мы осуществим эти подстановки в левой и правой частях уравнения, чтобы избавиться от ненужных констант.

Также мы могли получить условия с разными левыми, но одинаковыми правыми частями:

```
t.Cond1: (t.Const1 is (t.Const e.Exps (const s.NUMBER))),
t.Cond2: (t.Const2 is (t.Const e.Exps (const s.NUMBER))).
```

В таком случае мы сохраняем лишь одно условие (в реализации — с лексикографически наименьшей константой), например, `t.Cond1`, и генерируем замену константы второго условия `t.Const2` на `t.Const1`.

Выполняем в уравнении все сгенерированные подстановки (коллапсировавших констант в пустое слово и констант заменённых условий). Остаётся лишь нормализовать полученное уравнение.

4 PairComp

4.1 Вхождения сжимаемой пары

Рассмотрим возможности появления пары $\alpha\beta \in \Sigma^+$ в уравнении. Пусть одна из его частей представляется как

$$\Phi\alpha\beta\Psi, \quad \Phi, \Psi \in (\Sigma \cup \Xi)^*. \tag{1}$$

Здесь получаем **явное** вхождение исходной пары. Если часть уравнения имеет вид

$$\Phi X\beta\Psi, \quad X \in \Xi, \quad \Phi, \Psi \in (\Sigma \cup \Xi)^*, \tag{2}$$

мы можем рассмотреть случай, когда решение X оканчивается на α, т.е. $X = X\alpha$. Выполнив подстановку $X \to X\alpha$ (если это возможно) получим новое явное вхождение пары $\alpha\beta$. Поэтому нам интересна ситуация (2). Будем называть такое вхождение пары $\alpha\beta$ **перекрёстным**.

Случай
$$\Phi\alpha Y\Psi, \quad Y \in \Xi, \quad \Phi, \Psi \in (\Sigma \cup \Xi)^* \tag{3}$$
симметричен предыдущему: имеем перекрёстное вхождение пары $\alpha\beta$ и можем получить явное, выполнив подстановку $Y \to \beta Y$.

Наконец, часть уравнения может представляться в виде
$$\Phi XY\Psi, \quad X, Y \in \Xi, \quad \Phi, \Psi \in (\Sigma \cup \Xi) * . \tag{4}$$
В таком случае новое явное вхождение появляется с выполненными одновременно подстановками $X \to X\alpha$ и $Y \to \beta Y$.

Пара $\alpha\beta$ также может входить в уравнение **неявно**, не являясь началом или концом решения, а находясь полностью внтури него. Такая ситуация в силу неминимальности решения нам неинтересна.

4.2 Существенные подстановки

Обратим внимание на случаи (2) – (4) предыдущего пункта. *Такие и только такие* ситуации могут порождать в уравнении новые явные вхождения пары $\alpha\beta$. Способствующие этому подстановки называются **существенными**.

Существенные подстановки подразделяются на **элементарные** и **композитные**, включающие одновременно одну и две подстановки соответственно. Например, подстановки пунктов (2) и (3) являются существенными элементарными, а в (4) — существенной композитной.

Далее нам пригодится понятие **специальной** подстановки: таковыми будем называть существенные элементарные подстановки, являющиеся частью какой-либо существенной композитной подстановки.

4.3 Пустые подстановки

Все описанные выше подстановки являются **непустыми**. **Пустые** подстановки, однако, тоже могут порождать новые явные пары. Пусть часть уравнения представляется в виде
$$\Phi\alpha\Omega\beta\Psi, \quad \Omega \in \Xi^+, \quad \Phi, \Psi \in (\Sigma \cup \Xi)^*, \tag{5}$$

где хотя бы одна переменная в Ω может быть пуста. Пусть такой переменной будет $W \in \Xi$. Если максимальный блок W^i, $i \in \mathbb{N}$, следует непосредственно за α, обратив переменную в ε мы получим или новую явную пару $\alpha\beta$, или новую перекрёстную пару при соседстве α с новой переменной, следующей прямо за W^i. Получаем симметричную ситуацию, если W^i непосредственно предшествует β. Если же W^i находится внутри Ω, подстановка в пустое слово даст новое соседство двух переменных, следовательно — новое перекрёстное вхождение, где явная пара может быть порождена непустой существенной композитной подстановкой.

Мы видим, что во всех случаях имеет смысл совершать подстановку $W \to \varepsilon$. Аналогичными случаями являются

$$\Phi X \Omega \beta \Psi, \quad X \in \Xi, \quad \Omega \in (\Xi / X)^+, \quad \Phi, \Psi \in (\Sigma \cup \Xi)^*, \tag{6}$$

$$\Phi \alpha \Omega Y \Psi, \quad Y \in \Xi, \quad \Omega \in (\Xi / Y)^+, \quad \Phi, \Psi \in (\Sigma \cup \Xi)^*, \tag{7}$$

$$\Phi X \Omega Y \Psi, \quad X, Y \in \Xi, \quad \Omega \in (\Xi / X / Y)^+, \quad \Phi, \Psi \in (\Sigma \cup \Xi)^*, \tag{8}$$

Важно, чтобы среди Ω была хотя бы одна переменная, для которой возможна пустая подстановка.