

1 Теоретическая основа

Для описания структур данных используется *формальная* БНФ-грамматика. Все литералы имеют один из трёх префиксов: `s.`, `t.` или `e.`, соответствующие *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'a 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 кончатся 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, и **пустой** в противном случае.

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 $t.\text{Cond}$ содержит сжимаемые блоки и соответствующую этому сжатию константу. Вообще, блок $t.\text{Block}$ обозначает степень константы и представляется в виде

$$t.\text{Block} ::= (t.\text{Const } t.\text{Exp} * (\text{const } s.\text{NUMBER})),$$

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

$$t.\text{Cond} ::= (t.\text{Const is } t.\text{Block}+).$$

В программе используется только два типа условий: *парное условие* $t.\text{PairCond}$ и *условие на блок* $t.\text{BlockCond}$:

$$\begin{aligned} t.\text{PairCond} &::= (t.\text{Const is } (t.\text{Const } (\text{const } 1)) (t.\text{Const } (\text{const } 1))), \\ t.\text{BlockCond} &::= (t.\text{Const is } t.\text{Block}). \end{aligned}$$

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 $\alpha \in \text{First}(\gamma)$, если существует цепочка условий на константы такая, что

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

где $\beta_i \in \Sigma, i = 1, 2, \dots, n-1$, и $B_j \subset \Sigma, j = 1, 2, \dots, n$. Симметрично определяется множество *Last*-элементов константы.

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

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 $\alpha \in \text{First}(\beta) \cup \beta$. В противном случае рестрикция называется **независимой**.

Иногда нам потребуется модифицировать нетривиальные ограничения. Например, мы хотим выполнить подстановку $X \rightarrow X\alpha$, $X \in \Xi$, $\alpha \in \Sigma$, 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.

Нормальные уравнения представляют особый интерес для нас. Именно такие уравнения возвращают 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 возвращает

- **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 выполняет 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 — 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 замённых условий). Остаётся лишь нормализовать полученное уравнение.