Прогонка и встраивание функций в языке РЕФАЛ5-λ

Ситников К.А.
ИУ9-82, МГТУ им Н.Э.Баумана

Второе совместное рабочее совещание ИПС имени А.К Айламазяна РАН и МГТУ имени Н.Э.Баумана по функциональному языку программирования Рефал

11 июня 2019 года

Ограниченный РЕФАЛ

1) исключено использование t-переменных;

2) исключено использование открытых и повторных e-переменных выражения.

L-выражения

1) любое подвыражение этого выражения не должно содержать более одной e-переменной, не входящей в скобки;
2) любая e-переменная выражения не должна входить в него более одного раза.

Расширение L-выражений

Расширение L-выражений

1) любое подвыражение этого выражения не должно содержать более одной e-переменной, не входящей в скобки;
2) e-переменные и t-переменные не должны входить в выражение более одного раза.

Алгоритм проектирования выражений

Пусть имеются классы выражений:

$$E_t, E_l$$

Требуется найти:

$$E_t \cap E_l$$

Алгоритм проектирования выражений

$$E_l \cap E_t = C_1^t // E_t \cup .. \cup C_r^t // E_t$$

$$A_i^l // E_l = C_i^t // E_t$$

$$C_i^t = \{[v_1 \rightarrow L_1] .. [v_k \rightarrow L_k]\}$$

$$A_j^l = \{(L_{k+1} \leftarrow v_{k+1}) .. (L_m \leftarrow v_m)\}$$

$$[v_i \rightarrow L_i] - \textit{сужение}$$

$$(L_{k+1} \leftarrow v_{k+1}) - \textit{присваивание}$$

Алгоритм проектирования выражений

Конфигурация алгоритма:

$$\left[\begin{array}{l} <C_{1,}, A_{1,}, Q_1> \\ \quad \ldots \\ <C_n, A_n, Q_n> \end{array}\right.$$

Начальная конфигурация:

$$\left[<\emptyset, \emptyset, \{E_t : E_l\}>\right.$$

Алгоритм проектирования выражений

1) Берем первую тройку с непустым множеством уравнений

2) Из нее берем первое уравнение

3) Пытаемся его решить

4) В соответствии с решением преобразуем конфигурацию алгоритма

5) Выполняем до тех пор, пока во всех тройках не останется уравнений

6) Выполняем постобработку решения

Алгоритм проектирования выражений

Решение уравнений $T_t : T_l$ вида:

T_t - терм

T_l - терм

Терм:

1) s,t – переменные

2) структурные скобки

3) ADT-скобки

4) символы и указатели на функции

Алгоритм проектирвания выражений

$t.x:S, S-символ \rightarrow [t.x \rightarrow T]$

$s.x:S, S-символ \rightarrow [s.x \rightarrow S]$

$s.x:s.y \rightarrow (s.x \leftarrow s.y)$

$S:s.x, S-символ \rightarrow (S \leftarrow s.X)$

$S:S \rightarrow \emptyset$

Алгоритм проектирования выражений

$s.x:T$

T не символ
или
s-переменная

НЕТ РЕШЕНИЙ,
Удаляем тройку

$<C,A,Q>$

из конфигурации

Алгоритм проектирования выражений

$(E_k):(E_r)$ → $E_k:E_r$

$[E_k]:[E_r]$ → $E_k:E_r$

$t.X:s.Y$ → $[t.X \to s.Y]$

$T:t.X,T-терм$ → $(t.X \to S)$

Алгоритм проектирования выражений

$$t.X:(E_r) \quad \Rightarrow \quad [t.X \rightarrow (e.New)]$$

$$e.New:E_r$$

В остальных случаях – **НЕТ РЕШЕНИЙ**, удаляем тройку из конфигурации

Алгоритм проектирования выражений

Стоит отметететить, что добавление сужения $[v \rightarrow E]$
В тройку $< A, C, Q >$ должно сопровождаться ее модификацией,
Все переменные v всех уравнений тройки должны быть заменены на E

Таким же образом должна быть изменена левая часть всех присваиваний

Алгоритм проектирования выражений

$$T_t E_t : T_l E_l$$

$$E_t T_t : E_l T_l$$

$$\begin{cases} T_t : T_l \\ E_t : E_l \end{cases}$$

$$\begin{cases} T_t : T_l \\ E_t : E_l \end{cases}$$

Алгоритм проектирования выражений

$$e.x\,E_t : T_l\,E_l \Rightarrow \begin{cases} \begin{cases} [e.x \rightarrow \epsilon] \\ E_t : T_l\,E_l \end{cases} \\ \begin{cases} [e.x \rightarrow t.New1\;e.New2] \\ t.New1 : T_l \\ e.New2 : E_l \end{cases} \end{cases}$$

Аналогично обрабатываем ситуацию для

$$E_t\,e.x : E_l\,T_l$$

Алгоритм проектирования выражений

$E_t : e.x \rightarrow (E_t \leftarrow e.x)$

$e.1 .. e.n : \epsilon \rightarrow [e.1 \rightarrow \epsilon] .. [e.n \rightarrow \epsilon]$

$\epsilon : T_l E_l \rightarrow$ **НЕТ РЕШЕНИЙ**

Постобработка решений

$$(s.A \leftarrow v),(X \leftarrow v) => [s.A \rightarrow X]$$

$$(s.A \leftarrow v),(s.B \leftarrow v) => [s.B \rightarrow s.A], \text{удаляем}(s.B \leftarrow v)$$

$$(X \leftarrow v),(X \leftarrow v) => \text{удаляем одно из}(X \leftarrow v)$$

$$(X \leftarrow v),(Y \leftarrow v), X \neq Y => \text{нет решений}$$

$$(E_1 \leftarrow t/e.X),(E_2 \leftarrow t/e.X) => \text{undefined}$$

Постобработка решений

1) Если имеем хоть одно решение *undefined* – все решение будет undefined

2) Если решение пусто, то сопоставление считается неудачным

3) Во всех остальных случаях возвращаем набор пар вида <Сужения, Присваивания>

Эквивалентное преобразование функций

$$F=\{L_{F1}=R_L<G\,a_0>R_R,..,L_{Fn}=R_{Fn}\}$$

$$G=\{L_{G1}=R_{G1},..,L_{Gm}=R_{Gm}\}$$

$$a_0: L_{G1}=<C_1, A_1>,..,<C_k, A_k>$$

$$F=\left\{\begin{array}{l} C_1 // L_{F1}=C_1 // R_l A_1 // R_{G1} C_1 // R_R \\ ... \\ C_k // L_{F1}=C_k // R_l A_k // R_{G1} C_k // R_R \\ L_{F1}=R_l<G*1\,a_0>R_R \\ ... \\ L_{Fn}=R_{Fn} \end{array}\right.$$

$$G*1=\{L_{G2}=R_{G2,}..,L_{Gm}=R_{Gm}\}$$

Прогонка и встраивание

```
<Solve t.Eq> :=
    Success t.Solution* |
    Failure |
    Undefined
```

Прогонка

```
F {
    Expr1 = R1 <G
Args> R2;
    ..
    ExprM = RM;
}

G {

    Left1 = Right1;
    Left2 = Right2;
    …
    LeftN = RightN;
}

$DRIVE G;
```

Результат решения – `Success`
`(()(A))`

```
F {
    Expr1 = R1 A //
Right1 R2;
    ..
}
```

Замечание: `Expr1` может быть не L-выражением

Прогонка

Результат решения: `Success ((C1) (A1)) .. ((CK) (AK))`

```
F {
    Expr1 = R1 <G
Args> R2;
    ..
    ExprM = RM;
}

G {

    Left1 = Right1;
    Left2 = Right2;
    …
    LeftN = RightN;
}

$DRIVE G;
```

```
F {
    C1//Expr1 = C1//R1 A1//Right1 C1//R2;
    ..
    CK//Expr1 = CK//R1 AK//Right1 CK//R2;
    Expr1 = R1 <G*1 Args> R2;
    ..
    ExprM = RM;
}

G*1 {
    Left2 = Right2;
    …
    LeftN = RightN;
}

$DRIVE G*1;
```

Прогонка

Результат решения уравнения - `Failure`

```
F {
    Expr1 = R1 <G
Args> R2;
    ..
    ExprM = RM;
}

G {

    Left1 = Right1;
    Left2 = Right2;
    …
    LeftN = RightN;
}

$DRIVE G;
```

```
F {
    Expr1 = R1 <G*1
Args> R2;
    ..
    ExprM = RM;
}

G*1 {
    Left2 = Right2;
    …
    LeftN = RightN;
}

$DRIVE G;
```

Прогонка

Результат решения уравнения - `Undefined`

```
F {
    Expr1 = R1 <G
Args> R2;
    ..
    ExprM = RM;
}

G {

    Left1 = Right1;
    Left2 = Right2;
    …
    LeftN = RightN;
}

$DRIVE G;
```

```
F {
    Expr1 = R1 <Cold G
Args> R2;
    ..
    ExprM = RM;
}

G {
    Left1 = Right1;
    Left2 = Right2;
    …
    LeftN = RightN;
}

$DRIVE G;
```

Изменение семантики функций

```
F {
    e.A s.X e.B = <G s.X>;
    e.A = Z;
}

DRIVE G;

G {
    A = 1;
    B = 2;
}
```

```
G*2 {

}

F {
    e.A A e.B = 1;
    e.A B e.B = 2;
    e.A s.X e.B = <G*2 s.X>
    e.A = Z;
}
```

Встраивание

1) Исключаются ситуации неоднозначного решения

2) Используется ключевое слово `$INLINE`

Пример прогонки

```
$DRIVE TEST;

TEST {
  (e.A) F e.B (e.C F) = e.A e.B e.C;
}

TARGET {
   e.X = <TEST (e.X) e.X (e.X)>

}
```

```
TEST {
  (e.A#1) F e.B#1 (e.C#1 F) = e.A#1 e.B#1 e.C#1;
}

TARGET {
  F = F;

  F e.0#0 F = F e.0#0 F e.0#0 F F e.0#0;

  e.X#1 = <TEST*1 (e.X#1) e.X#1 (e.X#1)>;
}

TEST*1 {
}
```

Пример встраивания

```
$INLINE N;

N {
    s.u s.u = True;
}

J {

     s.2 =  <N s.2 s.2>
}
```

```
$INLINE N;

N {
    s.u s.u = True;
}


J {
  s.2#1 = True;
}
```

Дальнейшее развитие

1) Обработка ad-hoc решений (Undefined)

2) Прогонка для левых частей выражения с присваиваниями, условиями и блоками

Спасибо за внимание!