Расширенный алгоритм обобщённого сопоставления в компиляторе Рефала-5λ

Владислав Е. Пичугин
МГТУ имени Н.Э. Баумана, г. Москва

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

8 июня 2021

Введение

В общем случае алгоритм сопоставления имеет дело с уравнениями вида `E : P`, где `E` и `P` могут быть любыми образцовыми выражениями.

Решением уравнения является набор подстановок в `E` – **сужений** и подстановок в `P` – **присваиваний**, такой, что уравнение обращается в тождество.

Можно выделить три ситуации, когда уравнение `E : P` разрешимо:

- `E` – произвольное выражение, `P` – L-выражение,
- `E` – объектное выражение, `P` – произвольный образец,
- `E` – e-переменная `e.X`, `P` – произвольный образец, решение в этом случае – сужение `e.X → P` и пустой набор присваиваний.

Что если решения нет?

Для уравнения `E : P` **динамическим обобщением** будем называть пару из параметризованного выражения `E′` и подстановки `S` такую, что `E = E′ / S` и уравнение `E′ : P` имеет полное решение.

Динамическое обобщение открывает возможности для расширенной специализации функций.

Таким образом, требовалось разработать и реализовать в компиляторе алгоритм, который для данного уравнения `E : P` или находит полное решение, или находит приемлемое динамическое обобщение и строит полное решение для него.

Обзор алгоритма

Для дальнейшего осуществления динамического обобщения нужно добавить в исходное уравнение метки координат. Например,

$$_1(_2\ \mathtt{s.A}\ _3\ \mathtt{e.B}\ _4)\ _5\ (_6\ \mathtt{e.B}\ _7\ \mathtt{s.A}\ _8)_9\ :\ (\mathtt{e.X})\ (\mathtt{e.X})$$

В процессе решения будут рассматриваться два вида уравнений: клэши (или асимметричные клэши)

$$_m\ \mathtt{E}\ _n\ :\ \mathtt{P}$$

и симметричные клэши

$$_k\ \mathtt{E1}\ _l\ =\ _m\ \mathtt{E2}\ _n$$

Обзор алгоритма

Алгоритм выполняется в два этапа:

- Сопоставление выражения с образцом без учёта повторных переменных (асимметричные клэши).
- Разрешение повторных переменных (симметричные клэши).

Обзор алгоритма

Состояние алгоритма на первом этапе содержит следующие значения:

- текущий набор сужений `Ct`
- систему клэшей
- текущий набор присваиваний `As`, при этом присваивания являются мультисловарём

Обзор алгоритма

Состояние алгоритма на втором этапе содержит:

- текущий набор сужений `Ct`
- систему симметричных клэшей
- текущий набор присваиваний `As`, который уже является обычным словарём

Обзор алгоритма

В процессе решения алгоритм ветвится – строится упорядоченный набор ветвей.

Каждая ветвь может завершиться одной из трёх ситуаций:

- успешное решение – даёт пару `(Ct, As)` полного решения
- отсутствие решения – данная ветвь решений не имеет
- запрос на обобщение – указывает координаты обобщаемого участка в `E`

Если хотя бы одна из ветвей решения вернула запрос на обобщение – `E` обобщается. Если разные ветви предлагают обобщить разные участки аргумента, выбирается один из вариантов, он применяется и делается новая попытка решения.

Ветки с отсутствием решений усекаются в процессе решения. Может так оказаться, что все ветки оказались усечены. Это значит, что решений нет.

Преобразования системы клэшей

- Применение сужений ко всему состоянию решателя
- Упрощение координат

Сопоставления с L-образцами

$_m\, T\, _n : t.X \mapsto\; _m\, T\, _n \leftarrow t.X$

$_m\, (E)\, _n : (P) \mapsto\; _m\, E\, _n : P$

$_m\, (E)\, _n : Psym \mapsto$ нет решений

$_m\, s.X\, _n : X \mapsto s.X \rightarrow X$

$_m\, X\, _n : X \mapsto$ стираем

$_m\, T\, _n\, E : Pt\ P \mapsto\; _m\, T\, _n : Pt\ \&\&\ _n\, E : P$

$_m\, e.X\ E\, _n : Pt\ P \mapsto e.X \rightarrow t.NEW1\ e.NEW2\ ||\ e.X \rightarrow \varepsilon$

$_m\, e.X\ E\, _n : \varepsilon \mapsto e.X \rightarrow \varepsilon$

Сопоставление с открытыми переменными

Либо клэши закончились и нужно переходить ко второму этапу алгоритма, либо все клэши имеют вид `E : e.X P e.Y`

Нужно рассмотреть различные разбиения `E` на две части:

- Если `E` начинается на терм, то точка разбиения располагается перед первым термом.
- Точки разбиения добавляются между двумя смежными термами.
- Точки разбиения находятся «внутри» e-параметров.
- Если `E` заканчивается на терм, то точка разбиения добавляется в конец.

Сопоставление с открытыми переменными

```
E1 _m E2* T3.T4 E5* _n E6 : e.X P e.Y ↦ E1 _m E2* T3 _n ← e.X,
            ↑                            _m T4 E5* _n E6 : P e.Y

E1 e.X _m e.Y E2 : e.L P e.R  ↦  e.X → e.NEW1! t.NEW2 e.NEW3,
    ↑                              E1 e.NEW1! _m ← e.L,
                                 _n t.NEW2 e.NEW3 _m e.Y E2 : P e.R
                 где E1 = E3 _n E4*
```

Сопоставление с открытыми переменными

Имеется нюанс работы с открытыми переменными. Если в системе есть несколько клэшей на открытые переменные, то первый клэш может иметь слева произвольное выражение, а остальные должны иметь тривиальную левую часть, т.е. являться клэшами вида

$$_{m}\,\mathtt{e.X}\,_{n} : \mathtt{e.L\ P\ e.R}$$

$$\varepsilon : \mathtt{e.L\ P\ e.R}$$

Клэши, не удовлетворяющие этому условию приводят к запросу на динамическое обобщение соответствующего участка.

Решение симметричных клэшей

Симметричные клэши возникают в результате кратных вхождений переменных в правую часть исходного уравнения.

Пусть имеется переменная `v.X` и в процессе решения было получено несколько присваиваний:

```
E1 ← v.X, …, Ek ← v.X, …, Em ← v.X
```

Из этих присваиваний в качестве результата нужно оставить одно, а для остальных необходимо построить уравнения `Ei = Ej` (каждый с каждым).

Решение симметричных клэшей

В процессе решения в системе симметричных клэшей могут возникать тавтологии. Для них введено следующее правило:

```
E1 = E2  ↦  стираем, если CLEAR(E1) ≡ CLEAR(E2)
```

Решение симметричных клэшей

${}_a$ e.X ${}_b$ = ${}_c$ t.Y ${}_d$ $\mapsto$ e.X → t.NEW, t.Y → t.NEW

${}_a$ t.X ${}_b$ = ${}_c$ {{ &F e.X }} ${}_d$ $\mapsto$ обобщаем {c-d}

${}_a$ t.X ${}_b$ = ${}_c$ X ${}_d$ $\mapsto$ t.X → X

${}_a$ (E) ${}_b$ = ${}_c$ Sym ${}_d$ $\mapsto$ решений нет

${}_a$ T1 E1 = ${}_b$ T2 E2 $\mapsto$ ${}_c$ T1 ${}_d$ = ${}_e$ T2 ${}_f$ && ${}_g$ E1′ = ${}_h$ E2′

где ${}_c$ T1 ${}_d$, ${}_g$ E1′ := TERM_LEFT(${}_a$ T1 E1)

${}_e$ T2 ${}_f$, ${}_h$ E2′ := TERM_LEFT(${}_b$ T2 E2)

${}_a$ T ${}_b$ E1 = ${}_c$ e.X E2 $\mapsto$ e.X → t.NEW1 e.NEW2 || e.X → ε

Оптимизация специализации

Пусть имеется вызов `<F ARG>`. Преобразованием специализации назовём замену этого вызова на вызов `<F′ ARG′>` и построение новой функции `F′` на основе `F` такое, что тело функции `F′` учитывает статически известную информацию из исходного аргумента `ARG`, а новый аргумент `ARG′` эту статически известную информацию не содержит.

Оптимизация специализации

Особенности текущей реализации:

- Для специализируемых функций необходимо явно задавать входной формат и обозначать в нём статические и динамические параметры. Например, `$SPEC Map s.FUNC e.arg`
- Специализация ведётся только по статическим параметрам
- Экземпляры специализированных функций получают суффиксы `@n`: `Map@1, Map@2` и т.д.

Оптимизация специализации

Очевидное направление развития – специализация без шаблона.

```
F {
  … <S ARG> …
}

S {
  Pat1 Tail1;
  …
  PatN TailN;
}
```

```
F {
  … <S′ wrap(vars(ARG′)) / Sg> …
}

S′ {
  …
  wrap(vars(ARG′)) / Cij  Taili / Aij;
  …
}
```

- Решаются уравнения `ARG : Pati`
- Возможно динамическое обобщение `ARG ≡ ARG′ / Sg`

Таким образом, любой вызов функции `S` можно будет специализировать, возможно с обобщением.

Оптимизация специализации

```
… <Rot 'A' e.X e.Y> …

$SPEC Rot;

Rot {
  e.1 s.2 = s.2 e.1;
}
```

```
… <Rot@1 (e.X) e.Y> …

$SPEC Rot;

Rot@1 {
  (/*ε*/) /*ε*/ = 'A';

  (e.1 s.2) /*ε*/
    = s.2 'A' e.1;

  (e.1) e.2 s.3
    = s.3 'A' e.1 e.2;
}
```

Спасибо за внимание!