Обобщение предунификации Уэ на абстрактный синтаксис второго порядка

Стариков Артем Игоревич
Научный руководитель: Кудасов Николай Дмитриевич

Университет Иннополис

Семинар META, 2 июля 2025 г.

Введение: логическое программирование

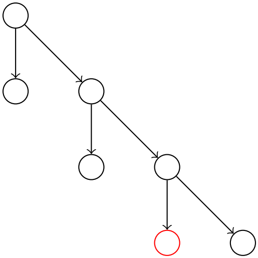
Вспомним логический язык программирования miniKanren. Программы на miniKanren состоят из реляционных функций:

```
(defrel (appendo xs ys zs)
    (conde
      [(== xs '()) (== ys zs)]
      [(fresh (x xs^ zs^)
         (== xs `(,x . ,xs^))
         (== zs `(,x . ,zs^))
         (appendo xs^ ys zs^))]))
```

Введение: поиск в логическом программировании

Реляционную программу можно запустить, предоставив ей известные данные и попросив заполнить неизвестные. miniKanren начнёт искать способ заполнить «дырки».

```
> (run* (q) (appendo '(1 2) '(3 4) q))
```



Введение: поиск в логическом программировании

Каждая вершина дерева — это наша логическая программа на определённом этапе её выполнения. Самое ключевое — это набор уравнений, собранных на текущий момент:

```
(== '(1 2) `(,x . ,xs^))
(== q `(,x . ,zs^))
(== xs^ `(,x^ . ,xs^^))
(== zs^ `(,x^ . ,zs^^))
(== xs^^ '())
(== '(3 4) zs^^)
```

Задача унификации

Другими словами, каждая вершина дерева поиска содержит *задачу унификации*:

$$
\begin{aligned}
(1,(2,())) &= (x, xs') \\
q &= (x, zs') \\
xs' &= (x', xs'') \\
zs' &= (x', zs'') \\
xs'' &= () \\
(3,(4,())) &= zs''
\end{aligned}
$$

Такие задачи унификации решаются не только в логических языках программирования, но и в системах типов различных языков программирования: например, в Hindley-Milner type system, который можно встретить в Haskell, Rust и других языках.

Задача унификации

Для текущего примера существует *унификатор* — подстановка, при которой выполняется каждое уравнение:

$$\begin{aligned} x &\mapsto 1 \\ xs' &\mapsto (2, ()) \\ zs' &\mapsto (2, (3, (4, ()))) \\ q &\mapsto (1, (2, (3, (4, ())))) \\ x' &\mapsto 2 \\ xs'' &\mapsto () \\ zs'' &\mapsto (3, (4, ())) \end{aligned}$$

Задача унификации высшего порядка

В отличие от задач унификации первого порядка (прошлый пример), в унификации высшего порядка переменные могут быть параметризованы:

$$F\ a\ b = c\ a$$
$$\lambda v.v\ X = G\ (\lambda u.u\ b)$$

Один из возможных унификаторов:

$$F \mapsto \lambda x.\lambda y.c\ x$$
$$G \mapsto \lambda f.\lambda v.v\ X$$

Задача унификации высшего порядка

В задачах унификации высшего порядка разделяются:

- универсальные переменные, a.k.a. связанные переменные;
- экзистенциальные переменные, значение которых нужно найти.

Дальше экзистенциальные переменные будем называть метапеременными, а универсальные — просто переменными.

Свойства унификации высшего порядка

Если задача унификации первого порядка имеет унификатор, то она имеет *наиболее общий* унификатор.

Задачи унификации высшего порядка могут иметь унификаторы, но не иметь наиболее общего унификатора:

$$F\ a\ a = a$$

$$F \mapsto \lambda x.\lambda y.x$$
$$F \mapsto \lambda x.\lambda y.y$$
$$F \mapsto \lambda x.\lambda y.a$$

Свойства унификации высшего порядка

Унификация первого порядка является разрешимой: для любой задачи можно за конечное время определить, имеет ли она унификатор.

Унификация второго и высших порядков является полуразрешимой: если унификатор существует, то его можно найти за конечное время; если унификатор отсутствует, то алгоритм унификации может никогда не завершить свою работу.[1]

$$
\begin{aligned}
& \{X\ a = a\ (Y\ a), Y\ a = X\ a\} \\
\xrightarrow{X \mapsto \lambda a.a\ (H_{\mathbf{1}}\ a)}\ & \{H_1\ a = Y\ a, Y\ a = a\ (H_1\ a)\} \\
\xrightarrow{Y \mapsto \lambda a.a\ (H_{\mathbf{2}}\ a)}\ & \{H_1\ a = a\ (H_2\ a), H_2\ a = H_1\ a\}
\end{aligned}
$$

[1] Goldfarb 1981, “The undecidability of the second-order unification problem”.

Алгоритмы унификации высшего порядка

Существуют полные алгоритмы унификации высшего порядка. Первой такой алгоритм опубликовали Йенсен и Пьетржиковский[2] в 1976 году. В алгоритме определены пять правил для генерации возможных подстановок. Впрочем, алгоритм не отличается эффективностью. Например, следующее правило перебирает всевозможные способы объединить параметры метапеременной:

(2) *Iteration rules*: Let e be any free variable which occurs above γ, δ or let e be either f or g in case either of these is a free variable. If e has type $(\mathbf{t}_1, \ldots, \mathbf{t}_m \to \mathbf{o})$, let the u_i here be variables of types $\mathbf{t}_i$. Let $\vec{w}$ be a sequence (possibly empty) of variables of various arbitrary types. Let u_l be one of the prefix variables and suppose u_l has type $(\mathbf{r}_1, \ldots, \mathbf{r}_j \to \mathbf{o})$. Then the following *iteration substitution* is *required for* γ, δ: ξ described by

$$\{e \leftarrow \lambda u_1 \ldots u_m \,.\, h(u_1, \ldots, u_m, \lambda \vec{w} \,.\, u_l(f_1(u_1, \ldots, u_m, \vec{w}), \ldots, f_j(u_1, \ldots, u_m, \vec{w}))).\}$$

[2] Jensen and Pietrzykowski 1976, “Mechanizing ω-order type theory through unification”.

Алгоритмы унификации высшего порядка

В 1975 году Уэ опубликовал полуразрешимый алгоритм, который лишь определяет существование унификатора для задачи унификации[3]. Благодаря этому алгоритм имеет всего лишь два правила и оставляет нерешёнными так называемые flex-flex уравнения, в которых метапеременные присутствуют с обеих сторон:

$$\lambda x.M\ a\ b\ x = \lambda x.\lambda y.N\ x\ y.$$

Тем не менее, результатом алгоритма является унификатор для уравнений остальных типов и список оставшихся flex-flex уравнений. Поэтому алгоритм Уэ часто называют алгоритмом предунификации.

[3] Huet 1975, “A unification algorithm for typed λ-calculus”.

As we shall see it is sufficient to consider these two cases. The result Σ will be the union of the results obtained using the two rules. In each case the possible values of r are found by type considerations.

3.4.1.1. *Imitation rule.* We want to "imitate" e_2 by e_1, substituting for f a term with head @. If @ is some v_i with $i \in [n_1]$, it will not be possible to introduce directly the corresponding v_i by substitution for f, since it is protected by the binder of e_1. It is only possible to introduce it indirectly if it appears in one of the arguments of e_1, and then the rule of projection will cover this case. So we limit the application of the rule of imitation to the case where $@ \in \mathcal{C}$ or $@ = v_i$ with $n_1 < i \leqslant n_2$.

(i) $n > 0$. This determines completely the heading of the term we must substitute for f. The rest of the term is filled in the most general way, by introducing new variables. Precisely, we return as unique solution in this case:

$$\sigma = \langle f, \lambda w_1 \dots w_{p_1} v_{n_1+1} \dots v_{n_2} \cdot @ (E_1, E_2, \dots, E_{p_2}) \rangle,$$

where $E_i = h_i(w_1, \dots, w_{p_1}, v_{n_1+1}, \dots, v_{n_2})$ $i \in [p_2]$, the h_i's being distinct variables of the appropriate type not in V. (Note: no conflict may arise, even if some v_k $(n_1 < k \leqslant n_2)$ appears free in e_1; also there is no risk of conflict of h_i with some w_j or v_k because they have different types.)

(ii) $n=0$. Considering the remark above, we must have here $@ \in \mathcal{C}$. The heading of the term substituted for f is going to be some $\lambda w_1 \dots w_k \cdot @$, with $0 \leqslant k \leqslant p_1$. However, any such k will not do, because we have a type condition on the remaining arguments of e_1. More precisely, we must be able to complete the term with a number of arguments $= p_2-(p_1-k) \geqslant 0$ which gives the first condition:

(1) $\qquad \max(0, p_1-p_2) \leqslant k \leqslant p_1.$

Also, the unchanged argument of e_1 must be type-compatible with the ones of e_2:

(2) $\qquad \alpha_j = \tau(e^2_{p_2-p_1+j}) \quad \forall j \, (k < j \leqslant p_1).$

Conversely, these conditions are sufficient for the substitution below to be legitimate. Therefore, for every k satisfying (1) and (2), we include in Σ

$$\sigma = \langle f, \lambda w_1 \dots w_k \cdot @ (E_1, E_2, \dots, E_{p_2-p_1+k}) \rangle,$$

where $E_i = h_i(w_1, \dots, w_k)$, $i \in [p_2-p_1+k]$, the h_i's being distinct variables of the appropriate type not in V. In this subcase, we have therefore at most $\min(p_1, p_2)+1$ solutions. Note that conditions (1) and (2) are always satisfied with $k = p_1$, which guarantees at least one solution. But we cannot restrict ourselves to this case unless we admit the η-reduction rule:

$$\mathcal{E}[\lambda u \cdot e(u)] = \mathcal{E}[e] \quad \text{if } u \notin \mathcal{F}(e).$$

For instance, consider $e_1 = f(B(A))$ and $e_2 = A(B(f))$, with $\tau(f) = \tau(A) = (\alpha \to \alpha)$, $\tau(B) = ((\alpha \to \alpha) \to \alpha)$. The unique unifier of e_1 and e_2 is $\{\langle f, A \rangle\}$, corresponding to $p_1 = p_2 = 1$, $k = 0$.

3.4.1.2. *Projection rule.* We want to project f on one of its arguments. The possible headings for the term we may substitute for f are:

(i)	$\lambda w_1 \dots w_k \cdot w_i$	$k \in [p_1]$, $\quad i \in [k]$,
(ii)	$\lambda w_1 \dots w_{p_1} v_{n_1+1} \dots v_{n_1+k} \cdot w_i$	$k \in [n]$, $\quad i \in [p_1]$,
(iii)	$\lambda w_1 \dots w_{p_1} v_{n_1+1} \dots v_{n_1+k} \cdot v_{n_1+i}$	$k \in [n]$, $\quad i \in [k]$.

However in case (iii), after substitution of the term for f, we would have to unify

$$\sigma e_1 = \lambda u_1 \dots u_{n_1} v'_{n_1+1} \dots v'_{n_1+k} \cdot v'_{n_1+i}(\dots)$$

with

$$\sigma e_2 = \lambda v_1 \dots v_{n_1} v_{n_1+1} \dots v_{n_2} \cdot @ (\dots),$$

and this will be rejected by SIMPL unless

$$k = n \text{ and } @ = v_{n_1+i},$$

which case we have already considered in 3.4.1.1(i), and so we can limit ourselves to cases (i) and (ii). We shall include in Σ the union of their solutions.

(i) Heading $\lambda w_1 \dots w_k \cdot w_i$

1) $\qquad k \in [p_1]$, $i \in [k]$, $\quad$ (i.e. $1 \leqslant i \leqslant k \leqslant p_1$).

Here we have a supplementary condition on the type of w_i, so that the term we construct be of the same type as f. Precisely, there must exist $m \geqslant 0$ such that:

(2) $\qquad \alpha_i = (\gamma_1, \gamma_2, \dots, \gamma_m, \alpha_{k+1}, \dots, \alpha_{q_1} \to \beta)$

for some $\gamma_1, \dots, \gamma_m \in T$. (This condition is satisfied by $m = 0$ if $k = q_1$ and $\alpha_i = \beta$.)

For each i and k satisfying (1) and (2), we take as solution in Σ:

$$\sigma = \langle f, \lambda w_1 \dots w_k \cdot w_i(E_1, E_2, \dots, E_m) \rangle$$

where $E_j = h_j(w_1, \dots, w_k)$, $j \in [m]$, the h_j's being distinct variables of the appropriate type not in V.

Note that, given i and k, m is completely determined from (2). This gives us at most $p_1(p_1+1)/2$ solutions.

(ii) Heading $\lambda w_1 \dots w_{p_1} v_{n_1+1} \dots v_{n_1+k} \cdot w_i$. This case is very similar to the previous one, changes are just notational.

(1) $\qquad k \in [n]$, $i \in [p_1]$

and similarly:

(2) $\qquad \alpha_i = (\gamma_1, \gamma_2, \dots, \gamma_m, \alpha_{p_1+k+1}, \dots, \alpha_{q_1} \to \beta).$

(As above, this condition is satisfied by $m = 0$ if $p_1+k = q_1$ and $\alpha_i = \beta$.)

The solutions in this case are all the:

$$\sigma = \langle f, \lambda w_1 \dots w_{p_1} v_{n_1+1} \dots v_{n_1+k} \cdot w_i(E_1, E_2, \dots, E_m) \rangle$$

where $E_j = h_j(w_1, \dots, w_{p_1}, v_{n_1+1}, \dots, v_{n_1+k})$, $j \in [m]$, the h_j's being distinct variables of the appropriate type not in V. Here we have at most $n \times p_1$ solutions.

Алгоритмы унификации высшего порядка

В 1991 году Миллер представил разрешимый алгоритм Pattern Unification[4]. Он накладывает ограничения на уравнения: все аргументы метапеременной должны быть различными связанными переменными. Алгоритм способен решать уравнения вида

$$\lambda x.\lambda y.M\ x\ y = \lambda a.\lambda b.\lambda c.N\ c\ b\ a,$$

но не вида

$$\lambda x.\lambda y.M\ (x\ y) = \lambda a.\lambda b.\lambda c.N\ c\ b\ a.$$

Functions-as-Constructors[5] расширяет Pattern Unification, сделав накладываемое ограничение менее строгим.

[4] MILLER 1991, "A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification".
[5] Libal and Miller 2016, "Functions-as-constructors higher-order unification".

Алгоритмы унификации высшего порядка

В 2020 году Вукмирович и др.[6] показали эффективный полный алгоритм унификации высшего порядка. Авторы статьи взяли за основу алгоритм Йенсена и Пьетржиковского и расширили его *оракулами* — другими алгоритмами унификации, которым делегируется решение некоторых уравнений. Среди оракулов есть алгоритм предунификации и Functions-as-Constructors.

[6] Vukmirović, Bentkamp, and Nummelin 2021, "Efficient full higher-order unification".

Абстрактный синтаксис второго порядка

Универсальная переменная обозначается как x.

Метапеременная M в синтаксисе второго порядка всегда применена к полному списку аргументов: $M[\overline{t}]$.

Оператор $\mathsf{F}(\overline{\overline{x}.t})$ — это конструкция в объектном языке. Оператор может содержать другие подвыражения, каждое из них может привязывать новые переменные.

- Лямбда-исчисление можно представить двумя операторами $\mathsf{lam}(x.t)$ и $\mathsf{app}(t, t)$.

Обобщённый алгоритм предунификации

Мой обобщённый алгоритм предунификации основан на алгоритме Уэ. В отличие от алгоритма Уэ, в моём алгоритме есть два новых шага.
Алгоритм начинает работу с одной задачи унификации и выполняет следующие шаги:

1. *Нормализовать* все уравнения;
2. *Декомпозировать* уравнения типа rigid-rigid;
3. Если есть уравнение типа flexible-rigid,
   3.1 сгенерировать возможные подстановки с помощью правил *имитации*, *проекции* и *введения*;
   3.2 применить каждую подстановку к задаче в отдельной ветви поиска;
   3.3 перейти к шагу 1.
4. Иначе мы нашли унификатор.

Гибкие и жёсткие термы

Любое ненормализуемое выражение можно классифицировать как *гибкое* или *жёсткое*:

- Переменная x — всегда жёсткая;
- Метапеременная $M[\overline{t}]$ — всегда гибкая;
- Оператор $\mathsf{F}(\overline{\overline{x}.t})$ гибок тогда и только тогда когда гибкой является любая из его голов. Голова оператора — это подвыражение, от значения которого оператор может замениться на другое выражение при нормализации.

Например,

- Лямбда-абстракция $\lambda x.t$ и пара (t, u) всегда жёсткие;
- Применение функции $f\ x$ гибкое если f гибкое. Доступ к полю пары $\pi_1\ t$ гибкое если t гибкое.

Декомпозиция

Декомпозиция разбивает сложные уравнения на более простые. Декомпозиция полностью структурна:

$$C \cup \{\forall \overline{x}.\ x = x\} \Rightarrow C$$

$$C \cup \{\forall \overline{x}.\ \mathsf{F}(\overline{\overline{y}.t}) = \mathsf{F}(\overline{\overline{y}.u})\} \Rightarrow C \cup \{\forall \overline{x}, \overline{y_i}.\ t_i = u_i\}^{1 \le i \le |\overline{t}|}$$

Уравнения такого вида невозможно унифицировать, а потому они завершают поиск:

- $\forall \overline{x}.\ x = y$
- $\forall \overline{x}.\ x = \mathsf{F}(\overline{\overline{y}.u})$
- $\forall \overline{x}.\ \mathsf{F}(\overline{\overline{y}.t}) = \mathsf{G}(\overline{\overline{z}.u})$

Если любая часть уравнения — это метапеременная, то такое уравнение остаётся неизменным.

Правило имитации

Для уравнения вида

$$\forall \overline{x}.\ M[\,\overline{t}\,] = \mathsf{F}(\overline{\overline{y}.u}),$$

правило имитации заменяет метапеременную на выражение, имеющее ту же структуру, что и выражение справа:

$$M[\,\overline{z}\,] \mapsto \mathit{imitate}(\mathsf{F}(\overline{\overline{y}.u}), \overline{z}).$$

imitate на операторах возвращает тот же оператор, где рекурсивно имитируются головы оператора, а остальные подвыражения заменяются на свежие метапеременные. Переменные не могут быть имитированы.

- Для $\forall x.\ M[x] = \lambda y.x\ y$, правило имитации генерирует $M[x] \mapsto \lambda y.H[x, y]$.

Правило проекции

Для уравнения вида

$$\forall \overline{x}.\ M[\,\overline{t}\,] = u,$$

правило проекции заменяет метапеременную на каждый из её параметров. Если тип параметра не совпадает, то правило пытается свести его к нужному типу[7].

$$M[\,\overline{z}\,] \mapsto \mathit{reduce}(z_i, \tau(z_i), \tau(u), \overline{z}) \quad \forall i : 1 \leq i \leq |\overline{z}|,$$

- Для $\forall f : \alpha \to \tau, x : \tau, y : \alpha.\ M[f, x, y] = f\ \ y$, правило проекции генерирует две подстановки: $M[f, x, y] \mapsto f\ \ H[f, x, y]$ и $M[f, x, y] \mapsto x$.

[7] Kudasov 2023, "Generalising Huet-style Projections in E-unification for Second-Order Abstract Syntax".

Правило введения

Для уравнений вида

$$\forall \overline{x}.\ \mathsf{F}(\overline{\overline{y}.t}) = u,$$

правило введения ищет метапеременные в позиции головы в левой части и заменяет их на выражения, которые приведут к нормализации выражения.

- Для $\forall x.\ M[]\ x = x$, правило введения генерирует $M[] \mapsto \lambda y.H[y]$.

Реализация

Я реализовал свой обобщённый алгоритм на Haskell с использованием Free Foil. Так как сигнатуры с пользовательскими операторами недостаточно для работы правил, я определил класс типов `HuetPreunifiable`:

```
class HuetPreunifiable typ metavar binder sig where
  normalize :: ...
  imitate :: ...
  project :: ...
  introduce :: ...
  isFlexible :: ...
```

Пользователю алгоритмы нужно предоставить экземляр этого класса для своей сигнатуры операторов.

Реализация

В моей реализации есть функция `solve`, которая ищет унификаторы для задачи унификации:

```
>>> constraint = parse "∀ a, b. F[a, b] X[a, b] = a b"
>>> mapM_ print $ solve [Problem metas [constraint]]
{ F[x0, x1] ↦ λx2. x0 x1, X[x0, x1] ↦ X[x0, x1] }
{ F[x0, x1] ↦ x0, X[x0, x1] ↦ x1 }
{ F[x0, x1] ↦ λx2. x2, X[x0, x1] ↦ x0 x1 }
{ F[x0, x1] ↦ λx2. x0 x2, X[x0, x1] ↦ x1 }
```

Результаты

1. Я разработал обобщённую версию алгоритма предунификации Уэ;
2. реализовал алгоритм на Haskell с помощью Free Foil;
3. проверил работоспособность алгоритма с помощью набора тестов.

Реализация алгоритма доступна на Github.[8]

Вместе с коллегами по диплому Федором Ивановым и Дамиром Афлятоновым я показал свою работу на семинаре по системам типов WITS 2025, и будем её показывать на семинаре по унификации UNIF 2025 (waiting for author notification).

[8] https://github.com/fedor-ivn/free-foil-hou/blob/main/src/Language/Lambda/Huet.hs

Список используемой литературы I

Goldfarb, Warren D. (1981). "The undecidability of the second-order unification problem". In: *Theoretical Computer Science* 13.2, pp. 225–230. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(81)90040-2.

Huet, G.P. (1975). "A unification algorithm for typed λ-calculus". In: *Theoretical Computer Science* 1.1, pp. 27–57. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(75)90011-0.

Jensen, Don C and Tomasz Pietrzykowski (1976). "Mechanizing ω-order type theory through unification". In: *Theoretical Computer Science* 3.2, pp. 123–171.

Kudasov, Nikolai (July 2023). "Generalising Huet-style Projections in E-unification for Second-Order Abstract Syntax". In: *UNIF 2023 - 37th International Workshop on Unification*. Veena Ravishankar and Christophe Ringeissen. Rome, Italy. URL: https://inria.hal.science/hal-04128229.

Список используемой литературы II

Libal, Tomer and Dale Miller (2016). "Functions-as-constructors higher-order unification". In: *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, pp. 1–17.

MILLER, DALE (Sept. 1991). "A Logic Programming Language with Lambda-Abstraction, Function Variables, and Simple Unification". In: *Journal of Logic and Computation* 1.4, pp. 497–536. ISSN: 0955-792X. DOI: 10.1093/logcom/1.4.497. eprint: https://academic.oup.com/logcom/article-pdf/1/4/497/3817142/1-4-497.pdf. URL: https://doi.org/10.1093/logcom/1.4.497.

Vukmirović, Petar, Alexander Bentkamp, and Visa Nummelin (2021). "Efficient full higher-order unification". In: *Logical Methods in Computer Science* 17.