КЗ-свойства языков. Проверка и вывод простейших полиморфных типов

Теория формальных языков

2021 г.

Возникновение понятия типа

Изначально возник в трудах Б.Рассела, который заметил, что в наивной теории множеств существует парадокс:

Парадокс Рассела

$$\Omega = \{A \mid A \notin A\} \Rightarrow (\Omega \in \Omega \Leftrightarrow \Omega \notin \Omega)$$

Понятие типа ограничивает возможные операции над его сущностями $\Rightarrow$ исключает парадоксы (неожиданное/неприемлемое поведение программ).

Определение типа

Система типов — гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.

Б.Пирс

Определение типа

Система типов — гибко управляемый синтаксический *метод доказательства* отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.

Б.Пирс

Описание утверждения о типах — *логическая спецификация*.

Определение типа

Система типов — гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.

Б.Пирс

Описание утверждения о типах — *логическая спецификация*.
Записывается: $\Gamma \vdash M : \sigma$, где Γ — это перечисление $x_i : \tau_i$ — aka контекст.

Определение типа

Система типов — гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.

Б.Пирс

Описание утверждения о типах — *логическая спецификация*.
Записывается: $\Gamma \vdash M : \sigma$, где Γ — это перечисление $x_i : \tau_i$ — aka контекст.
Читается: «в контексте Γ терм M имеет тип σ».

Определение типа

Система типов — гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по разновидностям вычисляемых ими значений.

Б.Пирс

Описание утверждения о типах — *логическая спецификация*.

Записывается: $\Gamma \vdash M : \sigma$, где Γ — это перечисление $x_i : \tau_i$ — aka контекст.

Читается: «в контексте Γ терм M имеет тип σ».

Понимается: «если придать переменным x_i типы τ_i, тогда можно установить, что тип выражения M есть σ».

Системы типов (неформально)

1. Простые типы (сорта);
2. Функциональные типы данных;
3. Алгебраические типы данных (сигнатуры + операции);
4. Упорядоченные сорта (решётки) типов данных — ООП.

Таблица связывания

КЗ-свойства имён вынуждают использовать таблицы связывания (имён и функций) с двумя базовыми операциями:

- bind :: ([таблица], [имя], [тип]) $\rightarrow$ [таблица];
- lookup :: ([таблица], [имя]) $\rightarrow$ [тип].

Пример правил типизации

- Сорта (простые типы): Bool, Int.
- Операторы: $=$, $+$, условный, вызов функции.
- Синтаксис:

[Prog]	::=	[Fs]	[Fs]	::=	[F] | [Fs]
[F]	::=	[TypeId] ([TIds]) = [Exp]			
[Exps]	::=	[Exp] | [Exp], [Exps]			
[TypeId]	::=	(Bool | Int) **id**	[TIds]	::=	[TypeId], [TIds] | [TypeId]
[Exp]	::=	**num** | **id** | [Exp]+[Exp] | [Exp] = [Exp] | **id** ([Exps])			
		| if [Exp] then [Exp] else [Exp] | let **id** = [Exp] in [Exp]			

Пример правил типизации

[Prog] ::=	[Fs]		[Fs] ::=	[F] | [Fs]
[F] ::=	[TypeId] ([TIds]) = [Exp]			
[Exps] ::=	[Exp] | [Exp], [Exps]			
[TypeId] ::=	(Bool | Int) **id**		[TIds] ::=	[TypeId], [TIds] | [TypeId]
[Exp] ::=	**num** | **id** | [Exp]+[Exp] | [Exp] = [Exp] | **id** ([Exps])			
	| if [Exp] then [Exp] else [Exp] | let **id** = [Exp] in [Exp]			

Пример правил типизации

[Prog] ::=	[Fs]		[Fs] ::=	[F] | [Fs]
[F] ::=	[TypeId] ([TIds]) = [Exp]			
[Exps] ::=	[Exp] | [Exp], [Exps]			
[TypeId] ::=	(Bool | Int) **id**		[TIds] ::=	[TypeId], [TIds] | [TypeId]
[Exp] ::=	**num** | **id** | [Exp]+[Exp] | [Exp] = [Exp] | **id** ([Exps])			
	| if [Exp] then [Exp] else [Exp] | let **id** = [Exp] in [Exp]			

tchExp(Exp, vtable, ftable) = case Exp of

num	int
id	| t == undef = err; int | otherwise = t where t = lookup(vtable, **id**)
Exp_1+Exp_2	| $t_1 \neq$ int || $t_2 \neq$ int = err; int | otherwise = int where t_1=tchExp(Exp_1,vtable,ftable), t_2=tchExp(Exp_2,vtable,ftable)

Пример правил типизации

tchExp(Exp, vtable, ftable) = case Exp of

num	int
id	| t == undef = err; int | otherwise = t where t = lookup(vtable, **id**)
Exp$_1$+Exp$_2$	| $t_1 \neq$ int || $t_2 \neq$ int = err; int | otherwise = int where t_1=tchExp(Exp$_1$,vtable,ftable), t_2=tchExp(Exp$_2$,vtable,ftable)
Exp$_1$=Exp$_2$	| $t_1 == t_2$ = bool | otherwise = err; bool where t_1=tchExp(Exp$_1$,vtable,ftable), t_2=tchExp(Exp$_2$,vtable,ftable)

Правила типизации в форме вывода

$$\frac{}{\Gamma \vdash \textbf{num} : \mathrm{int}} \qquad \frac{\Gamma \vdash t_1 : \mathrm{int}, \Gamma \vdash t_2 : \mathrm{int}}{\Gamma \vdash t_1 + t_2 : \mathrm{int}}$$

$$\frac{}{\Gamma, \textbf{id} : \tau \vdash \textbf{id} : \tau} \qquad \frac{\Gamma \vdash t_1 : \sigma, \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 = t_2 : \mathrm{bool}}$$

$$\frac{\Gamma \vdash t_1 : \mathrm{bool}, \Gamma \vdash t_2 : \sigma, \Gamma \vdash t_3 : \sigma}{\Gamma \vdash \mathrm{if}\ t_1\ \mathrm{then}\ t_2\ \mathrm{else}\ t_3 : \sigma}$$

$$\frac{\Gamma, \textbf{f\_id} : (\tau_1, \ldots, \tau_n) \to \tau_0 \vdash t_i : \tau_i}{\Gamma, \textbf{f\_id} : (\tau_1, \ldots, \tau_n) \to \tau_0 \vdash \textbf{f\_id}(t_1, \ldots, t_n) : \tau_0}$$

$$\frac{\Gamma, \textbf{id} : \tau \vdash s : \sigma, \Gamma \vdash t : \tau}{\Gamma \vdash M, \mathrm{let}\ \textbf{id} = t\ \mathrm{in}\ s : \sigma}$$

Параметрический полиморфизм

Параметрический полиморфизм — возможность определять функции равномерно для возможных типов аргументов. Альтернатива — ad hoc полиморфизм — функции перегружены, но определяются отдельно для каждого набора типов.

Параметрический полиморфизм

Параметрический полиморфизм — возможность определять функции равномерно для возможных типов аргументов. Альтернатива — ad hoc полиморфизм — функции перегружены, но определяются отдельно для каждого набора типов.

- Полиморфные типы — переменные, пробегающие все возможные сорта (простые типы)...

Параметрический полиморфизм

Параметрический полиморфизм — возможность определять функции равномерно для возможных типов аргументов. Альтернатива — ad hoc полиморфизм — функции перегружены, но определяются отдельно для каждого набора типов.

- Полиморфные типы — переменные, пробегающие все возможные сорта (простые типы)...
- ...если разрешены функции высших порядков (HOF) — тогда все возможные функциональные типы.

Как проверять типы полиморфных функций, если их алгебра бесконечнозначна?

Неформально о λ-исчислении

- Формальная модель вычислений, позволяет компактно описывать семантику ЯП с HOF.
- Бестиповая версия — А. Черч, 1930-е (и много типизированных).
- Базисные операции — применение (функция $\rightarrow$ данные) и абстракция (данные $\rightarrow$ функция).

Неформально о λ-исчислении

Пусть F, X — термы. $F\ X$ — операция применения терма F (функции) к терму X (данным).

Пусть $M \equiv M[x]$ — терм, возможно содержащий x. Тогда абстракция $\lambda x.M$ обозначает анонимную (неименованную) функцию от x: $x \to M[x]$.

Неформально о λ-исчислении

Пусть F, X — термы. $F\ X$ — операция применения терма F (функции) к терму X (данным).

Scheme

```scheme
; Первый элемент пары — функция, применяемая
; ко второму элементу.
(Fun1 Fun2)
```

Пусть $M \equiv M[x]$ — терм, возможно содержащий x. Тогда абстракция $\lambda x.M$ обозначает анонимную (неименованную) функцию от x: $x \to M[x]$.

Scheme

```scheme
(lambda (x) M)
```

β-редукция и β-эквивалентность

Применение анонимной функции к аргументу: $(\lambda x.M[x])\ N \to_\beta M[x := N]$ назовём β-редукцией.

Если $T_1 \to_\beta^+ T_2$ или $T_2 \to_\beta^+ T_1$, скажем, что термы T_1 и T_2 β-эквивалентны.

Считаем, что все связанные λ-абстракциями переменные имеют разные имена (как в исходном терме, так и в результатном).

Полуформально о типизации HOF

1. Если $M[x]$ имеет тип σ в контексте $x : \tau$, тогда естественно, что $\lambda x.M$ имеет тип $\tau \to \sigma$;
2. Если $(M\ N)$ имеет тип σ, а N имеет тип τ, тогда естественно, что M имеет тип $\sigma \to \tau$.

Полуформально о типизации HOF

1. Если $M[x]$ имеет тип σ в контексте $x : \tau$, тогда естественно, что $\lambda x.M$ имеет тип $\tau \to \sigma$;
2. Если $(M\ N)$ имеет тип σ, а N имеет тип τ, тогда естественно, что M имеет тип $\sigma \to \tau$.

Логическая спецификация

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma}$$

$$\frac{\Gamma \vdash M : \tau \to \sigma, \Gamma \vdash N : \tau}{\Gamma \vdash (M\ N) : \sigma}$$

Проблема типизации HOF

Рассмотрим терм $\lambda x.(x\ x)$. Какой у него тип?

Проблема типизации HOF

Рассмотрим терм $\lambda x.(x\ x)$. Какой у него тип?

- Пусть тип аргумента применения (то есть x) — это τ, а тип результата применения (то есть $(x\ x)$) — это σ. Тогда $\tau = \tau \to \sigma$.

Проблема типизации HOF

Рассмотрим терм $\lambda x.(x\ x)$. Какой у него тип?

- Пусть тип аргумента применения (то есть x) — это τ, а тип результата применения (то есть $(x\ x)$) — это σ. Тогда $\tau = \tau \to \sigma$.

Уравнение $\tau = \tau \to \sigma$ не имеет неподвижной точки, отличной от $\perp$ (унификация зацикливается).

Проблема типизации HOF

Рассмотрим терм $\lambda x.(x\ x)$. Какой у него тип?

- Пусть тип аргумента применения (то есть x) — это τ, а тип результата применения (то есть $(x\ x)$) — это σ. Тогда $\tau = \tau \to \sigma$.

Уравнение $\tau = \tau \to \sigma$ не имеет неподвижной точки, отличной от $\perp$ (унификация зацикливается).

Зацикливается не только унификация: см. $(\lambda x.(x\ x))\ (\lambda x.(x\ x))$. Однако в некоторых случаях всё успешно вычисляется: напр. $(\lambda x.(x\ x))\ (\lambda x.(\lambda y.(y\ x)))$.

Проблема типизации HOF

Рассмотрим терм $\lambda x.(x\ x)$. Какой у него тип?

- Пусть тип аргумента применения (то есть x) — это τ, а тип результата применения (то есть $(x\ x)$) — это σ. Тогда $\tau = \tau \to \sigma$.

Уравнение $\tau = \tau \to \sigma$ не имеет неподвижной точки, отличной от $\perp$ (унификация зацикливается).

Зацикливается не только унификация: см. $(\lambda x.(x\ x))\ (\lambda x.(x\ x))$. Однако в некоторых случаях всё успешно вычисляется: напр. $(\lambda x.(x\ x))\ (\lambda x.(\lambda y.(y\ x)))$.

$\lambda x.(x\ x)$ — частичная функция и не может быть конечным образом определена на всех полиморфных типах.

Нормальная форма HOF

Редекс — это подтерм вида $((\lambda x.M)\ N)$.
Нормальная форма λ-терма T — это λ-терм, β-эквивалентный исходному, не содержащий редексов.

Просто типизированное λ-исчисление

Ограничим множество λ-термов только такими, типы которых всегда выводимы по описанным выше правилам.

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma} \qquad \frac{\Gamma \vdash M : \tau \to \sigma, \Gamma \vdash N : \tau}{\Gamma \vdash (M\ N) : \sigma}$$

Просто типизированное λ-исчисление

Ограничим множество λ-термов только такими, типы которых всегда выводимы по описанным выше правилам.

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma} \qquad \frac{\Gamma \vdash M : \tau \to \sigma, \Gamma \vdash N : \tau}{\Gamma \vdash (M\ N) : \sigma}$$

...а теперь забудем про термы и посмотрим только на типы. Что получилось?

$$\frac{\Gamma, \tau \vdash \sigma}{\Gamma \vdash \tau \to \sigma}$$ (правило введения импликации)

$$\frac{\Gamma \vdash \tau \to \sigma, \Gamma \vdash \tau}{\Gamma \vdash \sigma}$$ (правило удаления импликации aka *modus ponens*)

Связь логики и HOF: соответствие Карри–Ховарда

- Существует взаимно-однозначное соответствие между типами замкнутых термов в просто типизированном λ-исчислении и тавтологиями в минимальной импликативной логике.
- (теорема о нормализации) Все термы просто типизированного λ-исчисления имеют нормальную форму.
- Доказательствам в минимальной логике соответствуют всюду определенные полиморфные функции высшего порядка.

Древесная форма естественного вывода

Правила вывода для $\Rightarrow$ в minLOG

Применению и абстракции соответствуют следующие правила вывода (modus ponens и правило дедукции):

$$( ) : \frac{A \qquad A \Rightarrow B}{B} \qquad\qquad \lambda. : \frac{\begin{array}{c} * A \\ \mid \; B \end{array}}{A \Rightarrow B}$$

Каждое применение правила вывода в доказательстве minLOG соответствует использованию в терме из $\lambda_{\rightarrow}$ конструкции с именем этого правила вывода.

В контексте соответствия Карри-Ховарда стрелочный тип далее обозначаем и как $\alpha \rightarrow \beta$ (в программировании), и как $\alpha \Rightarrow \beta$ (в логике).

Унификация

Алгоритм построения $mgu(E_1, \dots, E_k)$

$$\frac{t = x}{x = t} \qquad \frac{t = t}{} \qquad \frac{f\,t_1 \dots t_n = f\,s_1 \dots s_n}{t_1 = s_1 \dots t_n = s_n}$$

$$\frac{x = t \quad x = s}{x = t \quad t = s} \qquad \frac{x = t \quad r = s}{x = t \quad r[x := t] = s[x := t]}$$

Условия завершения унификации

$$\left.\begin{array}{l} \text{1. Существует уравнение} \\ \quad f\,t_1 \dots t_n = g\,s_1 \dots s_m\text{, где } f \neq g. \\ \text{2. Существует уравнение } x = f\,t_1 \dots t_n, \\ \quad \text{где } x \text{ входит в некоторое } t_i. \end{array}\right\} \text{неудача}$$

3. Все уравнения имеют вид $x_i = t_i$, причем x_i не имеет вхождений в t_i — успех.

Алгоритм Хиндли для $\lambda_{\to}$

Пусть дан терм T. В изначально пустом контексте Γ параллельно строятся приближение Φ типа терма T и система уравнений E на переменные типа Φ обратным применением следующих правил.

Правила вывода (Y — свежий тип)

$$\frac{\Gamma, x : X \vdash P : \Psi,\ E}{\Gamma \vdash \lambda x.P : Y, E \cup \{Y = X \to \Psi\}} \qquad \frac{\Gamma, x : X \vdash x : X,\ E}{\Gamma, x : X \vdash x : Y,\ E \cup \{X = Y\}}$$

$$\frac{\Gamma \vdash P : \Phi,\ E_1 \quad \Gamma \vdash Q : \Psi,\ E_2}{\Gamma \vdash (P\ Q) : Y, E_1 \cup E_2 \cup \{\Phi = \Psi \to Y\}}$$

Пусть построено $\vdash T : \Phi, E$. Из этого приближения строится $\vdash T : \Phi[mgu(E)]$ — окончательный тип терма T.

Пример на типизацию в $\lambda_{\rightarrow}$

Вывести тип терма $(\lambda k.k\ (\lambda x\, y.x))$

Соглашения об обозначениях:

- $\square$ — символ конца ветки подвывода.
- $\Phi = \Psi_1 \rightarrow \Psi_2$ — уравнение абстракции.
- $\Psi_1 = \Psi_2 \rightarrow \Phi$ — уравнение применения.
- $X_i = X_j$ — уравнение извлечения из контекста.
- Шаг извлечения из контекста в дереве вывода приближения типа не выписываем, только выписываем уравнение.

Пример вывода типа

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\square}{k : T_1, x : T_5, y : T_7 \vdash y : T_9} \qquad \cfrac{\square}{k : T_1, x : T_5, y : T_7 \vdash x : T_{10}}}{k : T_1, x : T_5, y : T_7 \vdash y\ x : T_8}}{k : T_1, x : T_5 \vdash \lambda y.(y\ x) : T_6}}{k : T_1 \vdash \lambda x\, y.(y\ x) : T_4} \qquad k : T_1 \vdash k : T_3\ \square}{k : T_1 \vdash k\ (\lambda x\, y.(y\ x)) : T_2}}{(\lambda k.k\ (\lambda x\, y.(y\ x))) : T_0}$$

$T_0 = T_1 \to T_2$

$T_3 = T_4 \to T_2$

$T_1 = T_3$

$T_4 = T_5 \to T_6$

$T_6 = T_7 \to T_8$

$T_9 = T_{10} \to T_8$

$T_9 = T_7\ T_5 = T_{10}$

Уравнения применения: $T_3 = T_4 \to T_2,\ T_9 = T_{10} \to T_8$.

Уравнения абстракции: $T_0 = T_1 \to T_2,\ T_4 = T_5 \to T_6,\ T_6 = T_7 \to T_8$.

Подставляем уравнения извлечения из контекста в систему и строим mgu системы $\{T_1 = T_4 \to T_2,\ T_7 = T_5 \to T_8,\ T_0 = T_1 \to T_2,\ T_4 = T_5 \to T_6,\ T_6 = T_7 \to T_8\}$.

Ответ: искомый тип терма есть $T_0 = ((T_5 \to ((T_5 \to T_8) \to T_8)) \to T_2) \to T_2$.

Пример обратного соответствия

Покажем, что тип $((A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C) \Rightarrow C$ корректен. Поскольку это — функциональный тип, то внешним конструктором его терма будет абстракция (соответствует правилу дедукции).

$*(A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C$ (тип терма x)

(Тут нужно придумать, как построить терм типа C, имея только x)

C

$((A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C) \Rightarrow C$ (тип терма $\lambda x.\ldots$)

Поскольку x — это функциональный терм, принимающий аргументом функцию типа $\tau = A \Rightarrow ((A \Rightarrow B) \Rightarrow B)$, нужно попробовать построить терм, имеющий тип τ. Для этого опять понадобится абстракция (даже две).

Пример обратного соответствия

$*(A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C$ **(тип терма x)**

$*A$ **(тип терма y)**

Тут опять не хватает шагов: нужно построить терм типа $(A \Rightarrow B) \Rightarrow B$, *имея* y.

Так как это функция, добавляем ещё абстракцию.

$(A \Rightarrow B) \Rightarrow B$

$A \Rightarrow ((A \Rightarrow B) \Rightarrow B)$ **(тип терма $\lambda y. \ldots$)**

C **(тип терма $x\ (\lambda y. \ldots)$)**

$((A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C) \Rightarrow C$ **(тип терма $\lambda x.x\ (\lambda y. \ldots)$)**

Пример обратного соответствия

$*(A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C$ (тип терма x)

> $*A$ (тип терма y)
>
> > $*A \Rightarrow B$ (тип терма z)
> >
> > > *Как получить терм типа* B *из* y *и* z*?*
> > >
> > > B
> >
> > $(A \Rightarrow B) \Rightarrow B$ (тип терма $\lambda z.\ldots$)
>
> $A \Rightarrow ((A \Rightarrow B) \Rightarrow B)$ (тип терма $\lambda y.(\lambda z.\ldots)$)
>
> C (тип терма $x\ (\lambda y.(\lambda z.\ldots))$)

$((A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C) \Rightarrow C$ (тип терма $\lambda x.x\ (\lambda y.\lambda z.\ldots)$)

Пример обратного соответствия

$*(A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C$ (тип терма x)

$*A$ (тип терма y)

$*A \Rightarrow B$ (тип терма z)

B (тип терма $z\,y$)

$(A \Rightarrow B) \Rightarrow B$ (тип терма $\lambda z.(z\,y)$)

$A \Rightarrow ((A \Rightarrow B) \Rightarrow B)$ (тип терма $\lambda y.(\lambda z.(z\,y))$)

C (тип терма $x\ (\lambda y.(\lambda z.(z\,y)))$)

$((A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C) \Rightarrow C$ (тип терма $\lambda x.x\ (\lambda y.\lambda z.(z\,y))$)

Мы не только построили доказательство тавтологии $((A \Rightarrow ((A \Rightarrow B) \Rightarrow B)) \Rightarrow C) \Rightarrow C$ в minLOG, но ещё и предъявили всюду определённую функцию высшего порядка, которая реализует это доказательство.

Теория полиморфных типов

- Добавление других конструкций (пары, объединения по ключу) — расширение логики типов дополнительными операциями ($\&$, $\vee$).
- Добавление контейнерных типов (списки, обработка исключений) — расширение логики типов модальностями.
- Результат алгоритмизации проверки типов — *theorems for free*: семантические свойства полиморфных функций выполняются вследствие корректности их типизации.

Теория полиморфных типов

- Добавление других конструкций (пары, объединения по ключу) — расширение логики типов дополнительными операциями ($\&$, $\vee$).
- Добавление контейнерных типов (списки, обработка исключений) — расширение логики типов модальностями.
- Результат алгоритмизации проверки типов — *theorems for free*: семантические свойства полиморфных функций выполняются вследствие корректности их типизации.

Логические системы, описывающие полиморфные типы, отличаются от классической! У их алгебр нет конечного носителя. Некоторые классические тавтологии (например, $A \vee (A \Rightarrow B)$) не являются конструктивными.