ТЕОРІЯ МОВ ПРОГРАМУВАННЯ

Лекція 2

Введення до теорії мов програмування

Людство поки ще досить далеко від створення єдиного універсального мови програмування. Нові мови з'являються майже кожен день, до вже існуючих додаються нові можливості. Покращення в мовах програмування служать створенню більш надійних програм, скорочують час розробки і спрощують супровід. Покращення необхідні і для задоволення нових вимог, таким як розробка паралельних, розподілених або мобільних додатків.

Визначення мови програмування починається з опису його синтаксису. Що вважати за краще, `x: = 1` або `x = 1`? Чи потрібно ставити дужки після `if` чи ні? І взагалі, яку послідовність символів можна вважати програмою? Для відповіді на ці запитання є корисний інструмент: формальна граматика. Використовуючи граматику, можна точно описати синтаксис мови програмування, що, в свою чергу, допомагає створювати програми перевірки синтаксичної коректності інших програм.

Однак навіть суворе визначення синтаксично коректної програми не дозволяє передбачити, що саме трапиться після її запуску. Визначаючи мову програмування, необхідно також описати його семантику, тобто очікуване поведінка програми під час виконання. Дві мови можуть мати однаковий синтаксис, але різну семантику.

Наведемо приклад того, що неформально розуміють під семантикою. Обчислення значення функції зазвичай пояснюють наступним чином. «Для обчислення результату **V** вираження, записаного в формі **f e_1 ... e_n**, де символ **f** позначає функцію, яка визначається виразом **f x_1 ... x_n = e'**, необхідно: по-перше, обчислити значення **W_1, ..., W_n** аргументів **e_1, ..., e_n**, потім зв'язати ці значення зі змінними **x_1, ..., x_n**, і, нарешті, обчислити вираз **e'**. Отримане в результаті значення **V** і є результатом обчислення ».

Таке пояснення семантики, виражене природною мовою (українською, в даному випадку), звичайно, дасть нам уявлення про те, що відбудеться під час виконання програми, але наскільки точним це уявлення виявиться? Розглянемо, наприклад, програму:

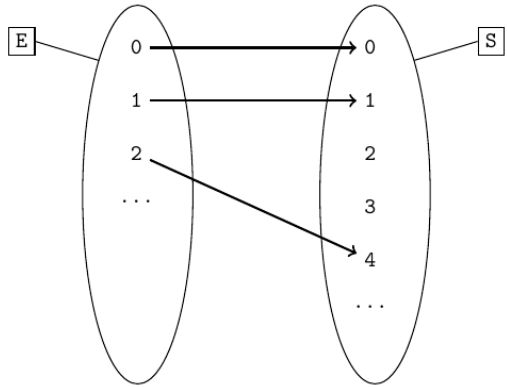
```
f x y = x
g z = (n = n + z; n)
n = 0; print (f (g 2) (g 7))
```

Залежно від способу інтерпретації даного вище пояснення можна встановити, що результатом програми буде або значення **2**, або значення **9**. Причина в тому, що пояснення природною мовою не уточнює, чи слід обчислювати **g 2** до або після **g 7**, тоді як у розглянутій програмі порядок обчислення важливий. У поясненні слід було б сказати: «аргументи $e_1, \ldots, e_n$ обчислюються, починаючи з e_1», або навпаки, «починаючи з e_n».

Два програміста, прочитавши неоднозначне пояснення, можуть зрозуміти його по-різному. Гірше того, розробники компіляторів мови можуть вибрати різні угоди. Тоді одна і та ж програма буде давати різні результати в залежності від використовуваного компілятора.

Добре відомо, що природні мови занадто неточні для опису синтаксису мови програмування, замість них слід використовувати формальні мови. У разі семантики ситуація аналогічна, для її опису також необхідний якийсь формальний мову.

Так що ж таке семантика програми? Візьмемо, наприклад, програму **p**, яка запитує ціле число, обчислює його квадрат і відображає отриманий результат. Щоб описати поведінку цієї програми, нам знадобиться ввести відношення **R** між вхідними значеннями і відповідними їм результатами.



Тепер можна сказати, що семантикою цієї програми є відношення **R** між елементами безлічі вхідних значень **E** і елементами безлічі вихідних значень **S**, тобто деяку підмножину декартового добутку **E × S**.

Отже, семантика програми є бінарним відношенням. У свою чергу, семантика мови програмування це тернарного ставлення: «програма **p** з вхідним значенням **e** повертає вихідне значення **s**». Позначимо це відношення в такий спосіб: **p, e ‹ → s**. Програма **p** і вхід **e** доступні до початку виконання програми. Найчастіше ці два елементи об'єднуються в терм **p e**, і семантика мови приписує значення цього терму. В цьому випадку семантика мови виявляється бінарним відношенням **t ‹ → s**.

Тепер для вираження семантики мови програмування нам потрібно мову, що дозволяє описувати відносини. Якщо семантика програми є функціональним відношенням, тобто для кожного вхідного значення існує не більше одного вихідного, будемо говорити, що програма є *детермінованою*.

Прикладами недетермінірованних програм є комп'ютерні ігри, оскільки для того, щоб гра приносила задоволення, необхідний елемент випадковості. Мова називається детермінованим, якщо всі програми, які можна написати на цій мові, детерміновані, або, що те ж саме, якщо його семантика є функціональним відношенням. В цьому випадку семантику можна визначити не описом відносин, а за допомогою деякого мови опису функцій.

Терми і відношення

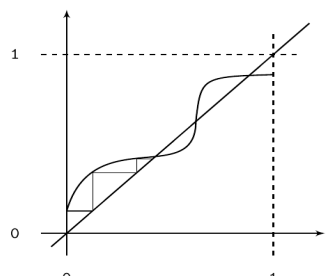
Оскільки семантика мови програмування являє собою відношення, ми почнемо з опису деяких засобів завдання множин і відносин.

Найпростішим засобом є явне визначення. Ми можемо, наприклад, явно визначити функцію, множити свій аргумент на 2: `x ↦ → 2 * x`, безліч парних чисел: `{n ∈ N | ∃p ∈ N n = 2 * p}`, або відношення подільності: `{(n, m) ∈ N2 | ∃p ∈ N n = p * m}`.

Однак подібних явних визначень недостатньо для опису всіх потрібних нам об'єктів. Другим засобом для завдання множин і відносин є індуктивне визначення. Воно ґрунтується на простому факті - *теоремі про нерухому точку*.

Теорема про нерухому точку

Теорема 1 (*перша теорема про нерухому точку*). Нехай **=<** - слабо повне відношення порядку на множині **E**, яке має найменший елемент **m**. Нехай функція **f** діє з **E** в **E**. Якщо **f** неперервна, то $\mathbf{p = lim_i\ (f^i\ m)}$ є найменшою нерухомою точкою **f**.



Друга теорема про нерухому точку стверджує існування нерухомої точки для зростаючих функцій, які можуть не бути безперервними, в разі, коли порядок задовольняє більш сильному вимогу.

Теорема 2 (*друга теорема про нерухому точку*). Нехай `=<` - сильно повне впорядкування на безлічі `E`. Функція `f` діє з `E` в `E`. Якщо `f` зростає, то `p = inf {c | f c =< c}` є найменшою нерухомою точкою `f`.

Індуктивні визначення

Теперь мы увидим, как теоремы о неподвижных точках могут быть использованы для определения множеств и отношений.

Пусть A — произвольное множество, функция f действует из An в A и E — подмножество A. Множество E замкнуто относительно действия функции f, если для любых a1 , . . . , an из E элемент f a1 . . . an также принадлежит E. Например, множество чётных чисел замкнуто относительно действия функции $n \mapsto\rightarrow n + 2$.

Пусть A — произвольное множество. Индуктивное определение подмножества E $\subseteq$ A это семейство частичных функций: f1 из An1 в A, f2 из An2 в A и т. д. Множество E определяется как наименьшее замкнутое относительно действия функций f1 , f2 , . . . подмножество A.

Например, подмножество N, которое содержит все чётные числа, определяется индуктивно числом 0 — то есть функцией из N 0 в N, которая возвращает значение 0 — и функцией из N в N: n $\mapsto\!\!\rightarrow$ n + 2. Подмножество {a, b, c}* , состоящее из слов вида an bcn , определяется индуктивно словом b и функцией m $\mapsto\!\!\rightarrow$ amc.

Вообще, любая контекстно-свободная грамматика может быть определена как индуктивное множество. В логике множество теорем является подмножеством всех утверждений, которое индуктивно определено аксиомами и правилами вывода.

Функции $\mathtt{f}_1, \mathtt{f}_2, \ldots$ называются *правилами*. Вместо записи правила в виде $\mathtt{x}_1 \ldots \mathtt{x}_n \mapsto \mathtt{t}$ мы будем использовать следующую нотацию:

$$\frac{\mathtt{x}_1 \ldots \mathtt{x}_n}{\mathtt{t}}.$$

Пусть $\mathtt{P}$ обозначает множество чётных чисел. Иногда мы будем записывать правила так:

$$\frac{}{0 \in \mathtt{P}}, \qquad \frac{\mathtt{n} \in \mathtt{P}}{\mathtt{n}+2 \in \mathtt{P}}.$$

Иногда, чтобы определить язык индуктивно, мы будем использовать нотацию, позаимствованную из теории формальных языков, где, к примеру, множество слов вида $\mathtt{a}^n\mathtt{b}\mathtt{c}^n$ задаётся так:

$$\mathtt{m} = \mathtt{b} \mid \mathtt{a}\,\mathtt{m}\,\mathtt{c}$$

Чтобы продемонстрировать существование наименьшего множества $\mathtt{A}$, замкнутого относительно действия функций $\mathtt{f}_1, \mathtt{f}_2, \ldots$, определим функцию $\mathtt{F}$ из $\wp(\mathtt{A})$ в $\wp(\mathtt{A})$:

$$\mathtt{F}\,\mathtt{C} = \{\mathtt{x} \in \mathtt{A} \mid \exists \mathtt{i}\ \ \exists \mathtt{y}_1 \ldots \mathtt{y}_{\mathtt{n}_\mathtt{i}} \in \mathtt{C}\ \ \mathtt{x} = \mathtt{f}_\mathtt{i}\,\mathtt{y}_1 \ldots \mathtt{y}_{\mathtt{n}_\mathtt{i}}\}.$$

Подмножество $\mathtt{C} \subseteq \mathtt{A}$ замкнуто относительно действия функций $\mathtt{f}_1, \mathtt{f}_2, \ldots$, если и только если $\mathtt{F}\,\mathtt{C} \subseteq \mathtt{C}$.

Функция $\mathtt{F}$, очевидно, возрастает, то есть, если $\mathtt{C} \subseteq \mathtt{C}'$, то $\mathtt{F\,C} \subseteq \mathtt{F\,C}'$. Более того, она непрерывна: если $\mathtt{C_0} \subseteq \mathtt{C_1} \subseteq \mathtt{C_2} \subseteq \ldots$, то $\mathtt{F}(\bigcup_\mathtt{j} \mathtt{C_j}) = \bigcup_\mathtt{j} (\mathtt{F\,C_j})$. Действительно, если элемент $\mathtt{x} \in \mathtt{A}$ принадлежит $\mathtt{F}(\bigcup_\mathtt{j} \mathtt{C_j})$, то существует индекс $\mathtt{i}$ и элементы $\mathtt{y_1} \ldots \mathtt{y_{n_i}} \in \bigcup_\mathtt{j} \mathtt{C_j}$, такие что $\mathtt{x = f_i\ y_1} \ldots \mathtt{y_{n_i}}$. Каждый из этих элементов лежит в одном из $\mathtt{C_j}$. Так как последовательность возрастает, то все элементы принадлежат некоторому наибольшему множеству $\mathtt{C_k}$. Следовательно, $\mathtt{x} \in \mathtt{F\,C_k}$, а также $\mathtt{x} \in \bigcup_\mathtt{j} (\mathtt{F\,C_j})$. Наоборот, если $\mathtt{x} \in \bigcup_\mathtt{j} (\mathtt{F\,C_j})$, то он принадлежит некоторому $\mathtt{F\,C_k}$, и, значит, существует номер $\mathtt{i}$ и элементы $\mathtt{y_1} \ldots \mathtt{y_{n_i}} \in \mathtt{C_k}$, такие что $\mathtt{x = f_i\ y_1} \ldots \mathtt{y_{n_i}}$. Элементы $\mathtt{y_1} \ldots \mathtt{y_{n_i}}$ принадлежат $\bigcup_\mathtt{j} \mathtt{C_j}$, а значит, $\mathtt{x} \in \mathtt{F}(\bigcup_\mathtt{j} \mathtt{C_j})$.

Множество `E` определяется как наименьшая неподвижная точка функции `F`. Это наименьшее множество, удовлетворяющее свойству `F E = E`, и, в соответствии со второй теоремой о неподвижной точке, наименьшее множество, удовлетворяющее свойству $\texttt{FE} \subseteq \texttt{E}$. Таким образом, это наименьшее замкнутое относительно действия функций $\texttt{f}_1$, $\texttt{f}_2$, ... множество.

Множество чётных чисел не единственное подмножество $\mathbb{N}$, содержащее 0 и замкнутое относительно действия функции $\texttt{n} \mapsto \texttt{n+2}$, (само множество $\mathbb{N}$, к примеру, также удовлетворяет этим свойствам), но наименьшее из таковых. Оно может быть определено как пересечение всех таких множеств. Вторая теорема о неподвижной точке позволяет нам обобщить это наблюдение и определить `E` как пересечение всех замкнутых относительно действия функций $\texttt{f}_1$, $\texttt{f}_2$, ... множеств.

Первая теорема о неподвижной точке показывает, что элемент x принадлежит E в том и только том случае, когда существует некоторый номер k, такой что $x \in F^k\varnothing$. То есть, если существует функция f_i, такая что $x = f_i\, y_1 \dots y_{n_i}$, где $y_1, \dots, y_{n_i} \in F^{k-1}\varnothing$. С помощью итерации, то есть индукцией по k, мы можем показать, что элемент $x \in A$ тогда и только тогда лежит в E, когда существует дерево, узлы которого помечены элементами множества A, корень помечен x, и если узел помечен c, то его дети помечены $d_1, \dots, d_n$, такими что для некоторого правила f имеем: $c = f\, d_1 \dots d_n$. Такое дерево называется *выводом* x. Понятие вывода обобщает понятие доказательства в логике. Мы можем определить множество E как набор элементов $x \in A$, для которых существует вывод.

Мы будем использовать специальную нотацию для выводов. Во-первых, корень дерева будет расположен внизу, а листья наверху. Далее, мы будем писать черту над каждым узлом дерева, а над этой чертой — детей этого узла.

Число 8, к примеру, принадлежит множеству чётных чисел, как показывает следующий вывод:

$$\dfrac{\dfrac{\dfrac{\dfrac{0}{2}}{4}}{6}}{8}$$

Если обозначить через P множество чётных чисел, можно записать вывод иначе:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{0 \in \mathtt{P}}}{2 \in \mathtt{P}}}{4 \in \mathtt{P}}}{6 \in \mathtt{P}}}{8 \in \mathtt{P}}$$

Структурна індукція

Индуктивные определения подсказывают метод проведения доказательств. Если свойство наследуется, то есть, если каждый раз, когда оно выполнено для $y_1, \ldots, y_{n_i}$, оно также выполнено и для $f_i\ y_1 \ldots y_{n_i}$, то можно заключить, что оно присуще всем элементам E.

Обосновать это можно, используя вторую теорему о неподвижной точке и заметив, что подмножество $P \subseteq A$, содержащее все элементы, которые удовлетворяют данному свойству, замкнуто относительно действия функций f_i и, таким образом, включает E. Другой способ обоснования — использовать первую теорему о неподвижной точке и показать индукцией по k, что все элементы $F^k\varnothing$ удовлетворяют рассматриваемому свойству.

Рефлексивно-транзитивне замикання відношення

Рефлексивно-транзитивное замыкание отношения является примером индуктивного определения. Если R — бинарное отношение на множестве A, мы можем индуктивно определить другое отношение R^*, называемое рефлексивно-транзитивным замыканием R:

$$\frac{}{x\ R^*\ y} \text{ если } x\ R\ y,$$

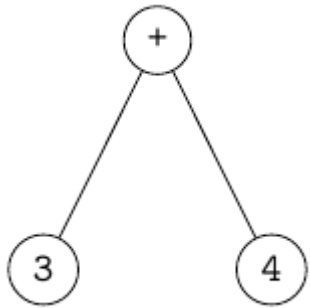
$$\frac{}{x\ R^*\ x},$$

$$\frac{x\ R^*\ y \qquad y\ R^*\ z}{x\ R^*\ z}.$$

Если рассматривать R как ориентированный граф, то R* это отношение, соединяющее две вершины тогда и только тогда, когда существует путь, ведущий из одной вершины в другую.

Мови без змінних

Теперь, после того как введены индуктивные определения, их можно использовать для определения понятия языка. Вводимое понятие языка не учитывает второстепенные синтаксические детали, например, неважно, какую из форм записи применять: `3 + 4`, `+(3, 4)` или `3 4 +`. Данный терм будет представлен в абстрактной форме деревом. Каждый узел в дереве будет помечен символом. Количество дочерних вершин узла зависит от пометки узла: например, два дочерних узла, если пометка `+`, и нуль, если пометка `3` или `4`.


+
3
4

Язык представляет собой набор символов, каждый их которых снабжён числовой характеристикой — арностью или, проще говоря, количеством аргументов символа. Символы без аргументов называются константами.

Множество термов языка есть множество деревьев, индуктивно определённых следующим правилом:

если **f** это символ с **n** аргументами, t_1 , . . . , t_n — термы, то **f(**t_1 **, . . . ,** t_n **)** — то есть дерево с корнем, помеченным **f**, и поддеревьями t_1 , . . . , t_n — является термом.

Змінні

Представим, что перед нами стоит задача спроектировать язык для определения функций. Одна из возможностей — использовать константы **`sin`**, **`cos`**, . . . и символ **`°`** с двумя аргументами. В таком языке возможно построить, к примеру, терм **`sin °(cos ° sin)`**.

Однако известно, что для определения функций проще использовать понятие, введённое Ф. Виетом (1540–1603), — понятие переменной. Функция, определённая выше, с помощью переменной может быть записана следующим образом: sin(cos(sin x)).

С 1930-х годов применяются и другие способы записи указанной выше функции: $x \mapsto \sin(\cos(\sin x))$ или $\lambda x.\ \sin(\cos(\sin x))$, где символы $\mapsto$ и λ используются для связывания переменной x. Явно указывая, какие переменные связаны, можно отличить аргументы функции от потенциальных параметров, а также зафиксировать порядок аргументов.

Символ $\mapsto$ был введён Н. Бурбаки около 1930, а символ λ — А. Чёрчем примерно в то же время. Нотация с λ является упрощённой версией предыдущей версии $\hat{x}\ \sin(\cos(\sin x))$, использовавшейся А. Н. Уайтхедом и Б. Расселом с 1900-х.

Определение f = x ↦⟶ sin(cos(sin x)) иногда записывается в форме f x = sin(cos(sin x)). Преимущество записи f = x ↦⟶ sin(cos(sin x)) в том, что здесь можно разделить две операции: конструирование функции x ↦⟶ sin(cos(sin x)) и само определение, которое даёт имя только что сконструированному объекту. В информатике часто бывает важным создавать объекты, не присваивая им имён.

В этом цикле лекций используется нотация, при которой рассматриваемая функция выглядит так: fun x → sin(cos(sin x)). Терм fun x → sin(cos(sin x)) задаёт функцию. Однако его подтерм sin x не задаёт ничего: это ни вещественное число, ни функция, так как содержит свободную переменную, значение которой неизвестно.

Для связывания переменных в термах необходимо расширить понятие терма, включив в него свободные переменные, которые будут связываться позднее. Это также потребует новых символов, подобных fun, которые выполняют связывание переменных в своих аргументах.

Примерами таких символов связывания могут служить { | }, ∂/∂, $\int$ d, $\sum$, $\prod$, $\forall$, $\exists$ и т. д. В этой книге используется несколько подобных символов: уже упомянутый выше fun, а также символы fix, let, fixfun и др.

Арность символа f более не будет просто числом, вместо этого используется конечная последовательность чисел (k1 , . . . , kn), которая показывает, что f связывает k1 переменных своего первого аргумента, k2 переменных второго аргумента, . . . , kn переменных своего n-го аргумента.

Далее, после того как задан язык, то есть множество символов с предписанными арностями, и бесконечное множество переменных, можно индуктивно определить множество термов следующим образом:

— переменные являются термами;

—если

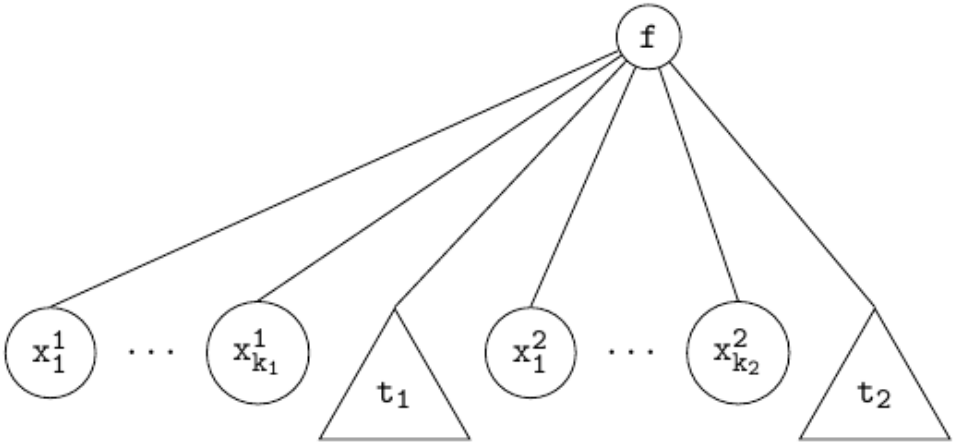
– f это символ арности (k1 , . . . , kn),

– t1 , . . . , tn — термы,

– x11 ,..., x1k1 , ... , xn1 , ..., xnkn — переменные,

то

f(x11 . . . x1k1 t1 , . . . , xn1 . . . xnkn tn) это терм.


f
x_1^1
$x_{k_1}^1$
t_1
x_1^2
$x_{k_2}^2$
t_2

Поясним данное определение на примере. Построим язык, в котором термы задают вещественные числа и функции на них, и который включает: две константы **`sin`** и **`cos`**, представляющие функции синуса и косинуса; символ α, называемый применением, такой что **`α(f, x)`** это объект, получаемый применением функции **`f`** к объекту **`x`**; символ **`fun`** для построения новых функций. Таким образом, этот язык включает четыре символа: константы **`sin`** и **`cos`**, символ **`α`** арности **`(0, 0)`** и **`fun`** арности **`(1)`**; множество термов индуктивно определено следующим образом:

— переменные являются термами,
— `sin` является термом,
— `cos` является термом,
— если `t` и `u` это термы, то `α(t, u)` является термом,
— если `t` это терм, а `x` — переменная, то `fun(x t)` является термом.

Введём упрощённую форму записи, в которой вместо терма `α(t, u)` используется `t u`, а вместо `fun(x t) — fun x → t`.

К примеру, `fun x → sin(cos(sin x))` представляет собой терм данного языка.

Багатосортні мови

В этой книге местами используются более общие, так называемые многосортные языки. Например, язык описания векторов с конечным набором константных символов, сложением и умножением на скаляр. В таком языке имеются два вида термов: термы, описывающие векторы, и термы, описывающие скаляры. В определении языка мы указываем, что символ + имеет два аргумента, оба являющихся векторами, а символ · имеет два аргумента, один из которых скаляр, а другой — вектор.

Для этого вводят двухэлементное множество сортов **{vect, scal}**, и символу · приписывают арность **(scal, vect, vect)**. Арность показывает, что в терме вида **λ · v** терм **λ** должен иметь сорт **scal**, а терм **v** — сорт vect, сам же терм λ · v имеет сорт vect.

Когда, кроме того, в языке присутствуют связанные переменные, арность символа **f** задаётся конечной последовательностью **((s$^1_1$,...,s$^1_{k1}$, s'1),..., (s$_{n1}$, . . . , snkn , s'$_n$), s'')**, указывающей, что символ имеет **n** аргументов, первый из которых принадлежит сорт **s '1** и в котором связываются переменные сортов **s$^1_1$, . . . , s$^1_{k1}$** , и т. д., результирующий терм принадлежит сорту **s'' .**

Вільні та зв’язані змінні

Множество переменных терма определяется структурной индукцией:

— **Var(x) = {x}**,

— **Var(f(x11 . . . x1k1 t1 , . . . , xn1 . . . xnkn tn)) = Var(t1) U {x11 , . . . , x1k1 } U . . . U Var(tn) U {x11 , . . . , xnkn }**.

Можно также определить множество свободных переменных терма:

— **FV(x) = {x}**,

— **FV(f(x11 . . . x1k1 t1 , . . . , xn1 . . . xnkn tn)) =**
(FV(t1) \ {x11 , . . . , x1k1 }) U . . . U
(FV(tn) \ {xn1 , . . . , xnkn }).

К примеру,
Var(fun x → sin(cos(sin x))) = {x},
FV(fun x → sin(cos(sin x))) = ∅.

Терм без свободных переменных называется замкнутым.
Высоту терма также можно определить посредством структурной индукции:

— **Height(x) = 0**,
— **Height(f(x11 . . . x1k1 t1 , . . . , xn1 . . . xnkn tn)) =**
1 + max(Height(t1), . . . , Height(tn)).

Підстановка

Первая операция, которую необходимо определить, это подстановка: действительно, назначение переменных не только быть связанными, но и использоваться для подстановки некоторых значений. Например, применение функции `fun x → sin(cos(sin x))` к терму `2*π` подразумевает, что терм `2*π` рано или поздно будет подставлен вместо переменной `x` в терме `sin(cos(sin x))`.

Подстановка это отображение из переменных в термы с конечной областью определения. Другими словами, подстановка это конечное множество пар, в каждой из которых первый элемент это переменная, а второй — терм, причём такое множество, что каждая переменная входит в качестве первого элемента пар не более одного раза. Можно также определить подстановку как ассоциативный массив $\theta = t_1 / x_1 \ . \ . \ . \ t_n / x_n$.

Когда подстановка применяется к терму, каждое вхождение переменной $x_1 \ , \ . \ . \ . \ , \ x_n$ в терм заменяется на $t_1 \ , \ . \ . \ . \ , \ t_n$ соответственно.

Разумеется, замена осуществляется только на свободных переменных.

К примеру, если выполняется подстановка терма `2` вместо переменной `x` в терме `x + 3`, результатом будет `2 + 3`. Однако если `2` подставляется вместо `x` в терме `fun x → x`, который представляет тождественную функцию, результатом должно быть `fun x → x`, а не `fun x → 2`.

Первая попытка строго определить применение подстановки к терму выглядит следующим образом:

— $\langle\theta\rangle x_i = t_i$,

— $\langle\theta\rangle x = x$, если x не принадлежит области определения θ,

— $\langle\theta\rangle f(y^1_1 \ldots y^1_{k1}\ u_1, \ldots, y^n_1 \ldots y^n_{kn}\ u_n) = f(y^1_1 \ldots y^1_{k1}\ \langle\theta|_{V\setminus\{y1\ 1\ ,\ldots,y1\ k1\}}\rangle u_1\ ,\ \ldots\ ,\ y_{n1} \ldots y_{nkn}\ \langle\theta|_{V\setminus\{yn1\ ,\ldots,ynkn\}}\rangle u_n)$,

где запись $\theta|_{V\setminus\{y1,\ldots,yk\}}$ используется для ограничения подстановки θ на множество $V \setminus \{y_1, \ldots, y_k\}$, то есть для исключения из подстановки всех пар, первый элемент которых принадлежит множеству $\{y_1, \ldots, y_k\}$.

Это определение имеет один недостаток: подстановки могут захватывать переменные. Например, терм `fun x → (x + y)` представляет функцию, которая прибавляет y к своему аргументу. Если заменить `y` на `4`, получится терм, представляющий функцию, которая добавляет `4` к своему аргументу. Если заменить `y` на `z`, получится терм `fun x → (x + z)`, представляющий функцию, которая добавляет `z` к своему аргументу. Однако если заменить `y` на `x`, получится функция `fun x → (x + x)`, которая удваивает свой аргумент, вместо функции, которая добавляет `x` к своему аргументу, как следовало бы ожидать.

Можно избежать этой проблемы, если поменять имя связанной переменной: связанные переменные это просто заглушки, их имена не играют существенной роли. Другими словами, в терме f**un x → (x + y)** можно заменить связанную переменную **x** на любую другую, за исключением, конечно, **y**. Аналогично, подставляя в терм **u** термы $t_1, \ldots, t_n$ вместо переменных $x_1, \ldots, x_n$, можно поменять имена связанных переменных в **u**, чтобы быть уверенным в том, что их имён нет среди $x_1, \ldots, x_n$, или среди свободных переменных $t_1, \ldots, t_n$, или среди свободных переменных **u**, во избежание захвата.

Начнём с определения отношения эквивалентности на термах индукцией по высоте терма. Оно называется алфавитной эквивалентностью, или **α**-эквивалентностью, и соответствует переименованию переменных.

—$x \sim x$,

—$f(y^1_1 \ldots y^1_{k1}\ t_1\ , \ldots ,\ y^n_1 \ldots y^n_{kn}\ t_n) \sim f(y'^1_1 \ldots y'^1_{k1}\ t'_1\ , \ldots ,\ y'^n_1 \ldots y'^n_{kn}\ t'_n)$,

если для всех i и для каждой последовательности свежих переменных $z_1, \ldots, z_{ki}$ (то есть для переменных, не входящих свободно в t_i, t'_i) имеет место $\langle z_1/y^i_1, \ldots, z_{ki}/y^i_{ki} \rangle t_i \sim \langle z_1/y'^i_1, \ldots, z_{ki}/y'^i_{ki} \rangle t'_i$.

Например, термы `fun x → x + z` и `fun y → y + z` α-эквивалентны.

И дальше мы работаем с термами по модулю α-эквивалентности, то есть подразумевая вместо термов классы α-эквивалентности термов.

Теперь можно определить операцию подстановки индукцией по высоте терма:

—$\theta x_i = t_i$,

—$\theta x = x$, если x не принадлежит области определения θ,

—$\theta f(y^1_1 \ldots y^1_{k1}\ u_1\ ,\ \ldots\ ,\ y^n_1 \ldots y^n_{kn}\ u_n) = f(z^1_1 \ldots z^1_{k1}\ \theta\langle z^1_1 / y^1_1\ ,\ \ldots\ ,\ z^1_{k1} / y^1_{k1}\rangle u_1, \ldots, z^n_1 \ldots z^n_{kn}\ \theta\langle z^n_1 / y^n_1\ ,\ \ldots\ ,\ z^n_{kn} / y^n_{kn}\rangle u_n)$, где $z^1_1, \ldots, z^1_{k1}, \ldots, z^n_1 \ldots z^n_{kn}$ это не входящие свободно в $f(y^1_1 \ldots y^1_{k1}\ u_1\ ,\ \ldots\ , y^n_1 \ldots y^n_{kn}\ u_n)$ и θ переменные.

К примеру, подстановка **2*x** вместо **y** в терме **fun x → x + y** даёт **fun z → z+(2*x)**. Выбор переменной **z** произволен, можно было бы взять **v** или **w**, и в результате получился бы тот же самый терм по модулю **α**-эквивалентности.

Композицией подстановок $\theta = t_1 / x_1 \ . \ . \ . \ t_n / x_n$ и $\sigma = u_1 / y_1 \ . \ . \ . \ u_p / y_p$ является подстановка

$\theta \circ \sigma = \{\theta(\sigma z)/z \mid z \in \{x_1, \ldots, x_n, y_1, \ldots, y_p\}\}$.

Можно доказать индукцией по высоте **t**, что для любого терма **t** имеет место

$(\theta \circ \sigma)t = \theta(\sigma t)$.

На наступній лекції ми продовжимо розглядати основи теорії мов програмування.