ТЕОРІЯ КОМПІЛЯТОРІВ

Лекція 9

ФОРМАЛІЗАЦІЯ СЕМАНТИКИ МОВ ПРОГРАМУВАННЯ

2020

Повний текст лекції буде розміщений на сайті baklaniv.at.ua

Опис мов програмування багато в чому спирається на теорію формальних мов. Ця теорія є фундаментом для організації синтаксичного аналізу і перекладу.

Існує два основних способи визначення мов:

- механізм породження або генератор;
- механізм розпізнавання або распознаватель.

Вони тісно пов'язані. Перший зазвичай використовується для опису мов, а другий для їх реалізації. Обидва способи дають можливість окреслити мови кінцевим чином, незважаючи на нескінченне число породжуваних ними ланцюжків.

Неформально мову **L** - це безліч ланцюжків кінцевої довжини в алфавіті **T**. Механізм породження дозволяє описати мови за допомогою системи правил, званої граматикою. Ланцюжки (пропозиції) мови будуються відповідно до цих правил. Гідність визначення мови за допомогою граматик в тому, що операції, вироблені в ході синтаксичного аналізу і перекладу, можна робити простіше, якщо скористатися структурою, що пропонується ланцюжкам за допомогою цих граматик.

Механізм розпізнавання використовує алгоритм, який для довільної вхідний ланцюжка зупиниться і відповість «так» після кінцевого числа кроків, якщо цей ланцюжок належить мові. Якщо ланцюжок не належить мові, алгоритм відповість **«ні»**. Розпізнавачі використовуються безпосередньо при побудові синтаксичних аналізаторів і є як би їх формальною моделлю. Розпізнавачі будуються на основі теорій кінцевих автоматів і автоматів з магазинної пам'яттю.

Формальні граматики

Граматикою називається четвірка

$$G = (N, T, P, S),$$

де **N -** кінцева множина нетермінальних символів (нетерміналів),

T - множина терміналів (що не перетинаються з N),

S - символ з **N**, який звуть початковим,

P - кінцева підмножина множини, називаного множиною правил: $(N \cup T)^{*} N (N \cup T)^{*} \times (N \cup T)^{*}$.

Множина правил **P** описує процес породження ланцюжків мови. Элемент $\mathbf{p_i} = (\alpha, \beta)$ множини **P** называється **правилом** (**продукцією**) і записується у вигляді $\alpha \Rightarrow \beta$. Маємо **α і β -** ланцюжки, що складаються з терміналів та нетерміналів. Цей запис може читатися одним з наступних способів:

– ланцюжок α породжує ланцюжок β;

– з ланцюжка α виводиться ланцюжок β.

Таким чином, правило **P** має дві частини: леву, визначаєму, та праву, що підставляється. Тобто правило $\mathbf{p_i}$ **-** це двійка ($\mathbf{p_{i1}}$, $\mathbf{p_{i2}}$), де $\mathbf{p}_{i1} = (N \cup T)^* N (N \cup T)^*$ **-** ланцюжок, що має хоча б один нетермінал, $\mathbf{p_{i2}} = (N \cup T)^*$ довільний, можливо порожній ланцюжок (**ε -** ланцюжок).

Якщо ланцюжок α має $\mathbf{p_{i1}}$, то, у відповідності до правила $\mathbf{p_i}$, можно утворити новий ланцюжок β, заміною одно входження $\mathbf{p_{i1}}$ на $\mathbf{p_{i2}}$. Говорять також, що ланцюжок β виведений з α в цій граматиці.

Для опису абстрактних мов у визначеннях та прикладах використовуються наступні позначення:

– термінали позначаються буквами a,b,c,d або цифрами 0,1, ...,9;

– нетермінали позначаються буквами A, B, C, D, S (причому нетермінал S - початковий символ граматики);

– букви U, V, ..., Z використовуються для позначення окремих терміналів або нетерміналів;

– через α, β, γ ... позначаються ланцюжки терміналів та нетерміналів;

– u, v, w, x, y, z - ланцюжки терміналів;

– для позначення порожнього ланцюжку використовується знак ε;

– знак «→» відокремлює ліву частину правила від правої й читається як «породжує» або «є за визначенням». Наприклад, A → cd, читається як «A породжує cd».

Ці позначення визначають деяку мову, призначену для опису правил побудови ланцюжків, а значить, для опису інших мов. Мова, призначена для опису іншої мови, називається **метамовою**.

Приклад граматики G1:

$$G1 = (\{A, S\}, \{0, 1\}, P, S),$$

де P:

1. $S \rightarrow 0A1$;
2. $0A \rightarrow 00A1$;
3. $A \rightarrow \varepsilon$.

Виведений ланцюжок граматики **G**, що не має в складі нетерміналів, називається **термінальним ланцюжком**, який породжений граматикою **G**.

Мова **L**(**G**), породжувана граматикою **G**, **-** це множина термінальних ланцюжків, породжуваних граматикою **G**.

Введемо відношення $\Rightarrow_G$ безпосереднього виведення на множині $(N \cup T)^*$, яке записується наступним чином:

$$\varphi \Rightarrow_G \psi$$

Данний запис читається: ψ безпосередньо виведений з φ для граматики $G = (N, T, P, S)$ і означає: якщо $\alpha\beta\gamma$ – ланцюжок з множини $(N \cup T)^*$ та $\beta \to \delta$ – правило з P, то $\alpha\beta\gamma \Rightarrow_G \alpha\delta\gamma$.

Через $\Rightarrow_G^+$ позначимо транзитивне замикання (нетривіальне виведення за один або більше кроків). Тоді $\varphi \Rightarrow_G^+ \psi$ читається як: ψ виведена з φ нетривіальним чином.

Через $\Rightarrow_G^*$ позначимо рефлексивне і транзитивне замикання (виведення за ноль і більше кроків). Тоді $\varphi \Rightarrow_G^* \psi$ означає: ψ виведена з φ.

Нехай $\Rightarrow^K$ – k-ий ступінь відношення $\Rightarrow$. Тобто, якщо $\alpha \Rightarrow^K \beta$, то існує послідовність $\alpha_0\ \alpha_1\ \alpha_2\ \alpha_3\ \ldots\ \alpha_k$ з к+1 ланцюжків

$$\alpha = \alpha_0, \alpha_1, \ldots \alpha_{i-1} \Rightarrow \alpha_i,\ 1 \le i \le k \text{ та } \alpha_k = \beta$$

Приклад виведень для граматики G1:

$S \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 0011;$

$S \Rightarrow^{1} 0A1;\ S \Rightarrow^{2} 00A11;\ S \Rightarrow^{3} 0011;$

$S \Rightarrow^{+} 0A1;\ S \Rightarrow^{+} 00A11;\ S \Rightarrow^{+} 0011;$

$S \Rightarrow^{*} S;\ S \Rightarrow^{*} 0A1;\ S \Rightarrow^{*} 00A11;\ S \Rightarrow^{*} 0011,$

где $0011 \subset \mathbf{L(G1)}$.

Граматики з обмеженнями на правила

Незважаючи на велику різноманітність граматик, при побудові трансляторів знайшли широке застосування тільки ряд з них, що мають деякі обмеження. Це пов'язано з практичною доцільністю використання певних типів правил, так як складність їх побудови безпосередньо впливає на складність побудови трансляторів.

За виглядом правил виділяють декілька класів граматик. У відповідності до классифікації Хомського граматика **G** називається:

– **праволінійною**, якщо кожне правило з **P** має вигляд:

$$A \to xB \text{ або } A \to x,$$

де **A**, **B** - нетермінали, **x** - ланцюжок, що складається з терміналів;

– **контекстно-вільною** (**КВ**) або **безконтекстною**, якщо кожне правило з **P** має вигляд: $A \to \alpha$, де $A \in N$, а $\alpha \in (N \cup T)^*$, тобто є ланцюжком, що складається з множини терміналів і нетерміналів, можливо порожнім;

– **контекстно-залежной** (**КЗ**) або **невкороченою**, якщо кожне правило з **Р** має вигляд: $\alpha \rightarrow \beta$, де $|\alpha| \leq |\beta|$. Тобто, нові породжувані ланцюжки не можуть бути коротше, ніж вхідні, а, значить, і порожніми (інші обмеження відсутні);

– **граматикою вільного вигляду**, якщо в ній відсутні вище згадані обмеження.

Приклад праволінійної граматики:

$$G2 = (\{S,\}, \{0,1\}, P, S),$$

де P:

1. $S \rightarrow 0S$;
2. $S \rightarrow 1S$;
3. $S \rightarrow \varepsilon$,

визначає мову $\{0, 1\}^*$.

Приклад КВ-граматики:

$$G3 = (\{E, T, F\}, \{a, +, *, (,)\}, P, E), \quad (3.11)$$

де P:

1. $E \rightarrow T$;
2. $E \rightarrow E + T$;
3. $T \rightarrow F$;
4. $T \rightarrow T * F$;
5. $F \rightarrow (E)$;
6. $F \rightarrow a$.

Дана граматика породжує найпростіші арифметичні вирази.

Приклад КЗ-граматики:

$$G4 = (\{B, C, S\}, \{a, b, c\}, P, S),$$

де P:

1. $S \rightarrow aSBC$;
2. $S \rightarrow abc$;
3. $CB \rightarrow BC$;
4. $bB \rightarrow bb$;
5. $bC \rightarrow bc$;
6. $cC \rightarrow cc$,

породжує мову $\{ a^n b^n c^n \}$, $n \geq 1$.

Примітка 1. Згідно з визначенням кожна праволінейная граматика є контекстно-вільною.

Примітка 2. За визначенням КЗ-граматика не допускає правил: $A \rightarrow \varepsilon$, де ε - порожній ланцюжок. Тобто КС-граматика з порожніми ланцюжками в правій частині правил не є контекстно-залежною. Наявність порожніх ланцюжків веде до граматики без обмежень.

Якщо мова L породжена граматикою типу G, то L називається мовою типу G.

Приклад: L(G3) - КВ мова типу G3.

Найбільш широке застосування при розробці трансляторів знайшли КВ-граматики і породжувані ними КВ-мови. Надалі при вивченні КВ-мов будуть розглядатися тільки корисні для нас з практичної точки зору (теорія мов вельми обширна і для детального її вивчення необхідно багато часу). Для отримання поглиблених знань рекомендується звернутися до фундаментальної монографії Ахо і Ульмана.

Про синтаксис мови програмування відомо досить багато; менш вивчений питання коректного визначення семантики мови. Керівництво по використанню мови програмування має включати опис кожної конструкції мови як окремо, так і в сукупності з іншими конструкціями.

Завдання коректного визначення семантики схожа на завдання визначення синтаксису. У мові є безліч різних конструкцій, точне визначення яких необхідно як програмісту, котрі використовують мову, так і розробнику реалізації цієї мови. Програмісту ці відомості потрібні для того, щоб писати правильні програми і заздалегідь знати результат виконання будь-яких операторів програми. Розробнику коректні визначення конструкцій необхідні для створення правильної реалізації мови.

У більшості посібників визначення семантики дається у вигляді звичайного тексту. Як правило, спочатку за допомогою будь-якої формальної граматики (наприклад, НФБ-граматики) дається визначення синтаксису конструкції, а потім для пояснення семантики наводяться кілька прикладів і невеликий пояснювальний текст.

На жаль, зміст цього тексту часто неоднозначний, так що різні читачі можуть розуміти його по-різному. Програміст може отримати помилкове уявлення про те, що саме буде робити написана ним програма при виконанні, а розробник може реалізувати будь-яку мовну конструкцію інакше, ніж розробники інших реалізацій того ж мови. Як і в разі синтаксису, потрібен якийсь метод, що дозволяє дати удобочитаем, точне і лаконічне визначення семантики всього мови.

Завдання визначення семантики мови програмування розглядається теоретиками так само довго, як і завдання визначення синтаксису, але в даному випадку набагато важче знайти задовільне рішення. Було розроблено безліч різних методів формального визначення семантики. Нижче ми наводимо опису деяких з них.

Граматичні моделі. Деякі ранні спроби додати коректне визначення семантики до мови програмування робилися шляхом додавання розширень до НФБ-граматиці, що визначає цю мову. Додаткову інформацію про семантику можна було витягти з дерева синтаксичного розбору. Ми коротко обговоримо атрибутивні граматики як спосіб отримання цієї додаткової інформації.

Імперативні (операційні) моделі. Операційне визначення мови програмування дає опис того, як складені на даному мовою програми виконуються на віртуальному комп'ютері. Зазвичай віртуальний комп'ютер визначається як автомат, але набагато більш складний у порівнянні зі звичайними моделями автоматів, які використовуються при вивченні синтаксису і здійсненні синтаксичного розбору.

Внутрішні стану цього автомата відповідають станам програми при її виконанні; це означає, що в стан автомата входять значення всіх змінних, що виконується програма і різні допоміжні системні структури даних. Для визначення можливих змін внутрішнього стану автомата в результаті виконання одного оператора програми використовується набір формально певних операцій.

Друга частина визначення задає спосіб трансляції тексту програми в початковий стан автомата. Починаючи з цього вихідного стану, автомат відповідно до визначальними його правилами послідовно переходить до наступних станів, поки не досягне кінцевого. Таке операційне визначення мови програмування може являти собою досить пряму абстракцію можливої фактичної реалізації мови.

З іншого боку, це визначення може представляти і більш абстрактну модель, яку можна використовувати як основу для програмно моделюється інтерпретатора мови, але не для фактичної реалізації.

У 70-і рр. була розроблена операційна модель мови під назвою Vienna Definition Language (VDL) - метамова, призначений для опису інших мов. У цій моделі дерево синтаксичного аналізу включає в себе також машинний інтерпретатор. Стан обчислення входить в дерево програми, а також в дерево, яке описує всі дані для конкретної машини. Черговий оператор переводить дерево в новий стан.

Аплікативні моделі. Аплікативне визначення мови намагається безпосередньо сконструювати визначення функції, яку обчислює кожна програма, написана на цій мові. Таке визначення мови будується як ієрархія визначень функцій, які обчислюються кожної окремої програмної конструкцією. За аналогією з аплікативного мовами, цей метод визначення мови є аплікативного підхід до моделювання семантики.

У програмі будь-яка елементарна операція або операція, певна програмістом, являє собою деяку математичну функцію. Структури управління послідовністю дій можуть бути використані для комбінації цих функцій в більші послідовності, представлені в тексті програми виразами і операторами. Лінійні послідовності операторів і умовне розгалуження легко можуть бути представлені функціями, складеними з функцій, які відповідають окремим компонентам цих конструкцій.

Цикл зазвичай представляється за допомогою рекурсивної функції, складеної з компонентів, що входять в тіло циклу. Зрештою утворюється функціональна модель всієї програми. Прикладами такого підходу до визначення семантики є метод денотаціонной семантики Скотта (Scott) і Стречі (Strachey) і метод функціональної семантики Міллза (Mills).

Аксіоматичні моделі. Даний метод поширює на програми область застосування обчислення предикатів. Семантику кожної синтаксичної конструкції мови можна визначити як певний набір аксіом або правил виведення, який можна використовувати для виведення результатів виконання цієї конструкції. Щоб зрозуміти сенс всієї програми (тобто розібратися, що і як вона робить), ці аксіоми і правила виводу слід використовувати так само, як при доказі звичайних математичних теорем.

У припущенні, що значення вхідних змінних задовольняють деяким обмеженням, аксіоми і правила виводу можуть бути використані для отримання (виведення) обмежень на значення інших змінних після виконання кожного оператора програми. Зрештою, коли програма виконана, ми отримуємо доказ того, що обчислені результати задовольняють необхідним обмеженням на їх значення щодо вхідних значень.

Тобто доведено, що вихідні дані представляють значення відповідної функції, обчисленої за значеннями вхідних даних. Прикладом описаного підходу є метод аксіоматичній семантики, розроблений Хоору (Hoare) .

Моделі специфікацій. У моделі специфікацій ми описуємо відношення між різними функціями, які реалізують програму. Поки нам вдається показати, що реалізація підпорядковується цього відношення між будь-якими двома функціями, ми можемо стверджувати, що вона коректна по відношенню до специфікації.

Алгебраїчний тип даних є одним з видів формальної специфікації. Наприклад, якщо ви пишете програму, що реалізовує стеки, то дії push (записати в стек) і pop (прочитати з стека) мають протилежну дію в тому сенсі, що якщо для деякого заданого стека S виконати дію push, а потім негайно - дія pop, то в підсумку вийде вихідний стек.

Це можна сформулювати у вигляді аксіоми:

pop(push(S,x)) = S

Будь-яка реалізація, яка зберігає цю властивість (а також деякі інші), є коректною реалізацією стека.

Формальне визначення семантики стає загальноприйнятою частиною визначення нової мови. Наприклад, стандартне опис мови PL / 1 включає в себе VDL-подібну нотацію, що описує семантику операторів PL / 1, а для мови Ada було розроблено визначення на основі денотаціонной семантики.

Проте вивчення формальних визначень семантики не зробило такого сильного впливу на практичне визначення мов, як вивчення формальних граматик - на визначення синтаксису. Жоден з методів визначення семантики не опинився корисним ні для користувача, ні для розробника мови.

Операційні моделі досить зручні для створення формальної моделі реалізації і можуть бути корисні розробнику, але для користувача ці моделі не мають великого значення, так як в них занадто багато непотрібних йому подробиць. Розробник навряд чи зможе керуватися функціональними і денотаціоннимі моделями, а для користувача вони, як правило, виявляються занадто складними, щоб їх можна було використовувати безпосередньо.

Користувачеві легше зрозуміти аксіоматичні моделі, але при спробі скласти повне визначення мови зазвичай вони стають надзвичайно складними, а для розробника ці моделі і зовсім непридатні.

Далі розглянемо короткий опис **атрибутивної граматики** як однієї з форм семантичної моделі мови програмування.

Атрибутивні граматики

Однією з перших спроб розробки семантичної моделі мови програмування була концепція атрибутивної граматики, запропонована *Дональдом Кнутом*. Ідея полягала в тому, щоб зіставити кожному вузлу дерева синтаксичного розбору даної програми деякуфункцію, задану семантичний зміст даного вузла. Атрибутивні граматики створювалися шляхом додавання функцій (атрибутів) до кожного правила граматики.

Успадкований атрибут - це функція, яка зіставляє нетермінальні значення в даному вузлі дерева з нетермінальнимі значеннями, розташованими на більш високому рівні дерева. Іншими словами, функціональне значення для нетермінальних символів, розташованих в правій частині будь-якого правила, є функцією нетермінальних символів, розташованих в його лівій частині.

Синтезований атрибут - це функція, яка співвідносить нетермінальні символи, розташовані в лівій частині правила, зі значеннями нетермінальних символів, розташованими в правій частині правила. Такі атрибути передають інформацію вгору по дереву (тобто вони синтезовані на основі інформації, взятої з нижніх рівнів дерева).

Розглянемо просту граматику для арифметичних виразів:

E -> T | E + T

T -> P | T x P

P -> I | (E)

Семантику цієї мови можна визначити за допомогою набору відносин між нетермінальними символами граматики. Наприклад, значення будь-якого виразу, що визначається цією граматикою, виводиться за допомогою наступних функцій.

Правило	Атрибут
E -> E + T	значення (E1) = значення (E2) + значення (T)
E -> T	значення (E) = значення (T)
T -> T x P	значення (T1) = значення (T2) x значення (P)
T -> P	значення (T) = значення (P)
P -> I	значення (P) = значення числа I
P -> (E)	значення (P) = значення (E)

Тут нижні індекси 1 і 2 позначають відповідно посилання на перший і другий однакові нетермінальні символи в даному правилі.

Атрибутивні граматики можна використовувати для передачі семантичної інформації по синтаксичному дереву. Наприклад, можна за допомогою правил оголошень зібрати всю інформацію про оголошеннях і інформацію з отриманої таблиці символів передавати вниз по дереву для використання при генерації коду для виражень.

Наприклад, для створення безлічі імен, оголошених в програмі, можна додати такі атрибути до нетермінальним символам <decl> и <declaration>.

Правило	Атрибут
<declaration> ::= <decl><declaration>	decl_set(declaration1) = decl_name(decl)Иdecl_set(declaration2)
<declaration> ::= <decl>	decl_set(declaration) = decl_name(decl)
<decl> ::= declare x	decl_name(decl) = x
<decl> ::= declare y	decl_name(decl) = y
<decl> ::= declare z	decl_name(decl) = z

Синтезований атрибут decl_set, пов'язаний з нетермінальним символом

Якщо в граматиці є тільки синтезовані атрибути (як у наведеному вище прикладі), то компілятор може обчислити їх в момент генерації дерева синтаксичного розбору на відповідному етапі трансляції.

Так працюють системи, подібні YACC.

YACC - це програма-генератор парсерів розроблена Стівеном С. Джонсоном в AT&T для операційної системи UNIX. Ім'я є акронімом до "Yet Another Compiler Compiler." Він генерує синтаксичний аналізатор (частина компілятора яка намагається побудувати синтаксичне дерево коду програми за формальною граматикою, записаною в нотації, подібній до БНФ.

Кожен раз, коли YACC визначає, яке застосувати правило (продукцію) НФБ-граматики, виконується підпрограма (наприклад, атрибутивна функція), яка доповнює дерево синтаксичного розбору семантичної інформацією.