ТЕОРІЯ МОВ ПРОГРАМУВАННЯ

Лекція 15

Імплементація логічного програмування

Весна 2021

Імперативні мови програмування відображають архітектуру основного комп'ютера, що зберігається програмою фон Неймана: програми оновлюють розташування пам'яті під контролем інструкцій. Виконання (здебільшого) послідовне. Послідовне виконання регулюється лічильником програми. Імперативні програми є розпорядчими. Вони точно диктують, як слід обчислювати результат за допомогою послідовності операторів, яку повинен виконувати комп'ютер. Розгляньте цю програму, використовуючи мову Small, представлену у попередніх лекціях.

Що ми хочемо знати про програму на рис. 15.1? Ми стурбовані детальним описом того, що відбувається, коли комп'ютер працює? Ми хочемо знати, на що налаштовано ПК, коли програма закінчується? Нас цікавить, що знаходиться в пам'яті 13 після другої ітерації циклу? На ці запитання не потрібно відповідати. Вони не говорять нам нічого про те, що робить програма.

Натомість, якщо ми хочемо зрозуміти програму, ми хочемо мати можливість описати взаємозв'язок між входом і виходом. Результатом є залишок після ділення першого значення на друге. Якщо це те, що нас насправді турбує, то чому б не програмувати, описуючи відносини, а не прописуючи набір кроків.

У логічному програмуванні програміст описує логічну структуру проблеми, а не вказує, як комп’ютер повинен її вирішувати. Мови для логічного програмування мають назву:
Мови опису: Програми виражаються як відомі факти та логічні взаємозв'язки щодо проблеми. Програмісти стверджують існування бажаного результату, а логічний інтерпретатор потім використовує комп'ютер для пошуку бажаного результату, роблячи висновки, щоб довести його існування.
Непроцедурні мови: Програміст зазначає лише те, що має бути виконано, і залишає за інтерпретатором, щоб визначити, як це має бути здійснено.

Реляційні мови: бажані результати виражаються як відношення або предикати, а не як функції. Замість того, щоб визначити функцію для обчислення квадратного кореня, програміст визначає відношення, скажімо **`sqr t (x, y)`**, яке відповідає дійсності саме тоді, коли $\mathbf{y^2 = x}$.

```
1  let val m = ref 0
2      val n = ref 0
3  in
4    m:=Int.fromString(
5        input("Please enter an integer: "));
6    n:=Int.fromString(
7        input("lease enter another: "));
8    while !m >= !n do m:=!m-!n;
9    println(!m)
10 end
```

Рис.15.1 Невеликий прикладом

Хоча існує багато мов логічного програмування, специфічних для додатків, є одна мова, яка виділяється як мова логічного програмування загального призначення. Пролог - це мова, яка найчастіше асоціюється з логічним програмуванням. Модель обчислення для Prolog не базується на архітектурі фон Неймана. Він базується на логічному механізмі, який називається уніфікація. Уніфікація - це процес, коли змінні уніфікуються до термінів. У цьому тексті досліджено різноманітні мови - від мови збірки JCoCo, до Java та C ++, до Standard ML та тепер Prolog. Ці мови відображають континуум від мов припису до мов опису.

- Мова асамблеї - це дуже передбачувана мова, що означає, що ви повинні думати з точки зору конкретної машини і відповідно вирішувати проблеми. Програмісти повинні думати категоріями моделі комп'ютера, що зберігається у програмі фон Неймана.
- C ++ - мова високого рівня, а отже, дозволяє більш наочно думати про проблему. Однак основною обчислювальною моделлю все ще залишається машина фон Неймана.
- Стандартна ML - теж мова високого рівня, але дозволяє програмісту математично думати про проблему. Ця мова дещо відходить від традиційної моделі фон Неймана.

- Prolog продовжує описовий компонент мов і дозволяє програмістам писати програми на основі виключно опису взаємозв’язків.

Пролог був розроблений у 1972 році. Ален Колмерауер, Філіп Руссель та Роберт Ковальскі були ключовими гравцями у розвитку мови Пролог. Це напрочуд маленька мова з великою силою. Інтерпретатор Prolog працює, здійснюючи глибокий перший пошук простору пошуку, одночасно об’єднуючи терміни, намагаючись дійти висновку щодо питання, яке програміст ставить перед інтерпретатором. Програміст описує факти та стосунки, а потім задає питання.

Ця проста модель програмування була використана в самих різних додатках, включаючи автоматичне написання рекламних оголошень про нерухомість, додаток, який пише юридичні документи різними мовами, інший, що аналізує соціальні мережі, та експертну систему управління звалищами. Це лише вибірка багатьох, багатьох додатків, написаних за допомогою цієї простої, але потужної моделі програмування.

Знову про Пролог

Якщо у вас ще немає перекладача Prolog, ви захочете завантажити його та встановити. Доступно багато версій Prolog. Деякі безкоштовні, а інші ні. Стандартне безкоштовне впровадження доступне за адресою http://www.swi-prolog.org.
Існують двійкові дистрибутиви, доступні для Microsoft Windows, Mac OS X та Linux, тому має бути щось, що відповідає вашим потребам.

На відміну від SML, немає можливості написати програму інтерактивно з Prolog. Натомість ви пишете текстовий файл, який іноді називають базою даних, що містить список фактів та предикатів. Потім ви запускаєте перекладач Prolog, переглядаєте файл і ставите запитання так чи ні, які перекладач Prolog намагається довести, що вони відповідають дійсності.
Щоб запустити інтерпретатор Prolog, ви вводите **`pl`** або **`swipl`** залежно від інсталяції SWI Prolog. Для виходу з інтерпретатора введіть **`ctl-d`**. Програма Prolog - це база даних фактів та предикатів, яка може бути використана для встановлення подальших взаємозв'язків між цими фактами. Предикат - це функція, яка повертає **`true`** або **`false`**. Програми пролог описують стосунки.

Простим прикладом є база даних фактів про декількох людей у розширеній родині та стосунки між ними, як показано на рис. 15.2.

Рис. 15.2
Родинне дерево

```
parent(fred, sophusw). parent(fred, lawrence).
parent(fred, kenny). parent(fred, esther).
parent(inger,sophusw). parent(johnhs, fred).
parent(mads,johnhs). parent(lars, johan).
parent(johan,sophus). parent(lars,mads).
parent(sophusw,gary). parent(sophusw,john).
parent(sophusw,bruce). parent(gary, kent).
parent(gary, stephen). parent(gary,anne).
parent(john,michael). parent(john,michelle).
parent(addie,gary). parent(gerry, kent).
male(gary). male(fred).
male(sophus). male(lawrence).
male(kenny). male(esther).
male(johnhs). male(mads).
male(lars). male(john).
male(bruce). male(johan).
male(sophusw). male(kent).
male(stephen). female(inger).
female(anne). female(michelle).
female(gerry). female(addie).
father(X,Y):-parent(X,Y),male(X).
mother(X,Y):-parent(X,Y), female(X).
```

Запитання, які ми можемо задати:
1. Софус батько Гері?
2. Хто батьки Кента?
3. Чи є Ларс батьком?
На всі ці питання може відповісти Пролог, враховуючи базу даних на рис. 15.2.

Камінці у фундаменті

Прологові програми (бази даних) складаються з фактів. Факти описують взаємозв'язок між термінами. Прості терміни включають числа та атоми. Атоми - це символи на зразок софуса, що представляють об'єкт у нашому всесвіті дискурсу. Атоми ПОВИННІ починатися з маленької літери. Числа починаються з цифри і включають як цілі, так і дійсні числа. Дійсні числа записуються в наукових позначеннях. Наприклад, 3.14159e0 або просто 3.14159, коли показник дорівнює нулю.

Згадуючи дисципліну “Дисретна математика”, предикат - це функція, яка повертає **`true`** або **`false`**. Предикати визначаються в Prolog, фіксуючи факт або факти про них. Наприклад, рис. 15.2 встановлює той факт, що Йохан був батьком Софуса. батько - присудок, що представляє справжній факт про стосунки Йохана та Софуса.

Часто терміни включають змінні в предикативні визначення для встановлення зв'язків між групами об'єктів. Змінна починається з великої літери. Змінні використовуються для встановлення взаємозв’язків між класами об’єктів. Наприклад, бути батьком означає, що ви повинні бути батьком когось і бути чоловіком. На рис. 15.2 присудок батька визначається письмом

```
father(X,Y):-parent(X,Y), male(X).
```

що означає, що X є батьком Y, якщо X є батьком Y, а X - чоловіком. Символ `:-` читається так, ніби і кома у визначенні предиката читається як і. Отже, X є батьком Y, якщо X є батьком Y, а X — чоловіком.

Для програмування в Prolog програміст спочатку пише базу даних, подібну до тієї, що зображена на рис. 15.2. Потім програміст звертається до бази даних, щоб інтерпретатор Prolog міг внутрішньо реєструвати факти, які там записані. Після консультації з базою даних можна задати питання щодо бази даних. Запитання, що задаються Prolog, обмежуються питаннями так чи ні, які задаються з точки зору предикатів у базі даних. Питання, поставлене Прологу, іноді називають запитом. Щоб дізнатись, чи Йохан є батьком Софуса, ви запускаєте Пролог за допомогою **`pl`** або **`swipl`**, потім звертаєтесь до бази даних і задаєте запит.

```
% swipl
?- consult('family.prolog').
?- father(johan,sophus).
Yes
?-
```

Запити можуть також містити змінні. Якщо ми хочемо з'ясувати, хто є батьком софуса, ми можемо запитати у Пролога, замінивши батьківську позицію в предикаті змінною. При використанні змінної у запиті Пролог відповість так чи ні. Якщо відповідь так, Prolog повідомить нам, яке значення було змінної, коли відповідь була **так**. Якщо є більше ніж один спосіб відповіді "**так**", тоді введення крапки з комою скаже Prolog шукати інші значення, де запит відповідає дійсності.

```
? - f a t h e r ( X , s o p h u s ).
X = johan
Yes
? - p a r e n t ( X , k e n t ).
X = gary ;
X = gerry ;
No
?-
```

Остаточний **No** is Prolog каже нам, що інших істинних батьківських стосунків **(X, kent)** не може бути.

Програма

Prolog виконує уніфікацію для пошуку рішення. Уніфікація - це просто перелік підстановок термінів на змінні. Запит бази даних узгоджується з її визначенням предикатів у базі даних. Терміни в запиті відповідають, коли серед параметрів предиката в базі даних знайдено відповідний шаблон. Якщо відповідний предикат залежить від істинності інших предикатів, тоді ці запити ставляться до інтерпретатора Prolog. Цей процес триває до тих пір, поки Пролог не виявить, що жодна заміна не задовольнить запит, або знайде відповідну заміну.

Пролог використовує глибинний перший пошук із зворотним відстеженням для пошуку правильної заміни. У пошуках істини він уніфікує змінні терміни. Як тільки буде знайдено дійсну заміну, він повідомить про заміну та почекає введення. Вище ми бачили, як інтерпретатор повідомляє, що **`X = gar y`** - це змінна, яка робить батьківський **`(X, kent)`** істинним. Пролог чекає, доки не буде натиснуто повернення або не буде введено крапку з комою. Коли вводиться крапка з комою, Пролог скасовує останню успішну заміну, яку він зробив, і продовжує пошук іншої заміни, яка задовольнить запит. Пролог повідомляє, що **`X = gerry`** також задовольнить запит.

Повторне натискання крапки з комою скасовує заміну **`X = gerry`**, Пролог продовжує свій глибинний перший пошук, шукаючи іншу заміну, не знаходить жодної та повідомляє **`No`**, вказуючи, що пошук вичерпав усі можливі заміни.
Об’єднання знаходить заміну термінів змінними або змінних термінами. Об’єднання - це симетрична операція. Це не працює лише в одному напрямку. Це означає (серед іншого), що предикати Прологу можуть працювати вперед і назад. Наприклад, якщо ви хочете знати, хто такий батько Кента, ви можете запитати це так само легко, як і хто є батьком Гері. У наступному прикладі ми дізнаємось, що Гері - батько Кента. Також ми з’ясовуємо, хто є батьком Гарі.

```
? - f a t h e r ( X , k e n t ).
X = gary ;
No
? - f a t h e r ( gary , X ).
X = kent ;
X = stephen ;
X = anne ;
No
```

Списки

Prolog підтримує списки як структуру даних. Список будується так само, як і в ML. Список може бути порожнім, що в Prolog записується як `[]`. Непорожній список будується з елемента та списку. Побудова списку з головою, `H` та хвостом, `T`, записується як `[H | T]`. Отже, `[1,2,3]` також можна записати як `[1 | [2 | [3 | []]]]`. Список `[a | []]` еквівалентно написанню `[a]`. На відміну від ML, списки в Prolog не повинні бути однорідними.

Отже, **[1, привіт, 4.3]** є дійсним списком Пролога. В силу того, що алгоритм Prolog - це пошук по глибині в поєднанні з уніфікацією, Prolog, природно, виконує відповідність шаблону. Не тільки **[H | T]** працює над побудовою списку, вона також працює для відповідності списку зі змінною. Додаток можна записати як взаємозв'язок між трьома списками. Результатом додавання перших двох списків є третій аргумент до предиката додавання. Перший факт нижче говорить про те, що додавання порожнього списку до передньої частини **Y** - це просто **Y**. Другий факт говорить про те, що додавання списку, першим елементом якого є **H**, до передньої частини **L2** призводить до **[H | T3]** при додаванні **T1** і **L2 T3**.

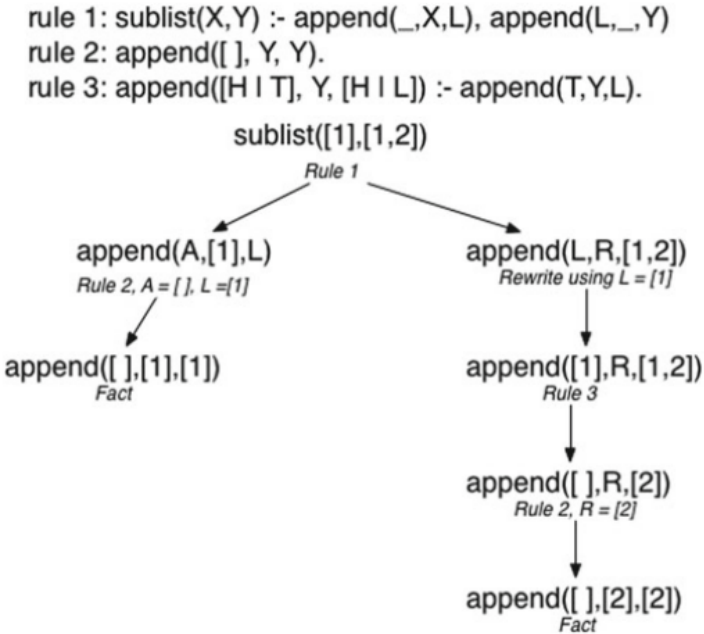
```
append([], Y, Y).
append([H|T1], L2, [H|T3]) :-
append(T1, L2, T3).
```

Спробуйте додати як назад, так і вперед! Визначення **append** може бути використано для визначення предиката, що називається підсписком, наступним чином:

```
sublist(X, Y):- append(_, X, L),
append(L, _, Y).
```

Викладено англійською мовою, це говорить про те, що **`X`** - це підсписок **`Y`**, якщо ви можете додати щось на передній частині **`X`**, щоб отримати **`L`**, а щось інше на кінці **`L`**, щоб отримати **`Y`**.
Підкреслення використовується в предикатних визначеннях для значень, які ми не піклуватися про.
Щоб довести, що підсписок **`([1], [1,2])`** відповідає дійсності, ми можемо використати визначення підсписку та додати, щоб знайти підстановку, для якої діє предикат. Малюнок 15.3 надає доказ того, що **`[1]`** є підсписком **`[1,2]`**.

Рис.15.3
Уніфіковане
дерево



На наступній лекції буде розглянуте продовження теми імплементації логічної мови програмування.