МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

МАТЕМАТИЧНА ЛОГІКА ТА ТЕОРІЯ АЛГОРИТМІВ

КОНСПЕКТ ЛЕКЦІЙ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 113 «Прикладна математика»,
освітньою програмою «Наука про дані та математичне моделювання»

Київ
КПІ ім. Ігоря Сікорського
2021

Рецензенти: *Боярінова Ю.Є.*, кандидат техн. наук., с.н.с., доцент кафедри системного програмування і спеціалізованих комп'ютерних систем ФПМ
Сулема Є.С., д-р техн. наук., доцент, завідувач кафедри програмного забезпечення комп'ютерних систем ФПМ

Відповідальний редактор *Тарасенко-Клятченко О.В.*, кандидат техн. наук, доц.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 2 від 09.12.2021 р.)
за поданням Вченої ради факультету прикладної математики
(протокол № 4 від 29.11.2021 р.)

Електронне мережне навчальне видання

Темнікова Олена Леонідівна

МАТЕМАТИЧНА ЛОГІКА ТА ТЕОРІЯ АЛГОРИТМІВ

Математична логіка та теорія алгоритмів: Конспект лекцій [Електронний ресурс]: навч. посіб. для студ. спеціальності 113 «Прикладна математика», освітньої програми «Наука про дані та математичне моделювання» / О.Л.Темнікова ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 3,60 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 177 с.

Навчальний посібник розроблено для ознайомлення студентів з лекційним курсом дисципліни «Математична логіка та теорія алгоритмів» та призначено для студентів, які навчаються за спеціальністю 113 «Прикладна математика», освітньою програмою «Наука про дані та математичне моделювання» факультету прикладної математики КПІ ім. Ігоря Сікорського. «Математична логіка та теорія алгоритмів: Конспект лекцій» містить викладання законів пропозиційної логіки, кванторизації та методів логіки 1-го порядку; засобів формалізації знань, основ логічного програмування; теорії багатозначних та модальних логік, поняття нечітких множин та лінгвістичних змінних. В курсі розглядаються примітивно-рекурсивні функції, різні алгоритмічні схеми та філософія проблем обчислюваності.

ЗМІСТ

ВСТУП

Навчальне видання призначене для студентів, які вивчають дисципліну «Математична логіка та теорія алгоритмів» з циклу загальної підготовки навчального плану спеціальності 113 «Прикладна математика» при навчанні за освітньою програмою «Наука про дані та математичне моделювання», що спрямована на розроблення і використання нових інформаційних технологій, моделювання складних систем, систем штучного інтелекту, і для вирішення інших задач професійної діяльності.

Лекційний курс з кредитного модулю «Математична логіка та теорія алгоритмів» розраховано на 36 академічних годин аудиторних занять, вивчається на факультеті прикладної математики у 3 семестрі.

Метою викладання дисципліни є оволодіння основними поняттями і методами, що необхідні для формування світогляду на математичну логіку як на фундаментальну науку, що призначена для формалізації знань, розробки і використання нових інформаційних технологій, моделювання складних систем, систем штучного інтелекту. Навчальне видання містить викладання законів пропозиційної логіки, кванторизації та методів логіки 1-го порядку; засобів формалізації знань, основ логічного програмування; теорії багатозначних та модальних логік, поняття нечітких множин та лінгвістичних змінних. В курсі розглядаються примітивно-рекурсивні функції, різні алгоритмічні схеми та філософія проблем обчислюваності. Викладання кредитного модулю надає змогу формування у студентів здатностей до абстрактного мислення, знання основних алгоритмічних схем теорії алгоритмів та основних понять теорії обчислюваня. Курс знайомить студентів з сучасними розділами некласичних логік та прикладами їх практичного застосування.

Після засвоєння кредитного модуля «Математична логіка та теорія алгоритмів» студенти мають продемонструвати такі результати навчання: формалізувати знання про прикладну проблемну область мовою логіки предикатів; доводити логічні наслідки в даній проблемній області за допомогою достовірного (дедуктивного) виводу; формалізувати знання за допомогою нетрадиційних логік та міркувань; застосовувати методи багатозначних логік і нечітких множин для опису погано структурованих проблемних областей.

Навчальне видання має стати в нагоді під час опрацювання матеріалів лекцій, підготовки до екзамену, до практичних занять.

1. ТЕОРІЯ ПРЕДИКАТІВ ПЕРШОГО ПОРЯДКУ

1.1. Методологічні принципи формальної логіки

Логіка - наука про мислення, про правильні міркування, про прийоми та методи пізнання за допомогою міркувань. Логіка, вивчаючи правильні міркування, оперує поняттями істинності і хибності. Правильне міркування або висловлювання покладається істинним, неправильне – хибним.

Методологічні основи формальної логіки полягають у виконанні декількох принципів

Принцип тотожності. Істинність фактів, що лежать в основі висловлювань і міркувань, встановлюється на підставі відомих законів, спостережень. .

Принцип несуперечності означає, що, стверджуючи що-небудь, не можна заперечувати те ж саме.

Принцип виключення третього. Не можна одночасно відкидати висловлювання і його заперечення. Будь-яке висловлювання може бути або істинним, або хибним, – третього не дано.

Принцип достатньої підстави. Будь-яке висловлювання має бути обгрунтовано, тобто істинність твердження не можна приймати на віру. Якщо твердження виводиться з будь-яких суджень, даних, фактів – *підстав,* то їх повинно бути досить для встановлення істинності твердження.

Можна ще вказати на одну дуже важливу властивість **- монотонностіь достовірних міркувань.** Якщо істинність деякого висловлювання *(A)* вже встановлена, то додавання нових фактів *(B)* не змінює істинності *A*.

Судження, або висловлювання – це думка, в якій стверджується наявність або відсутність якихось фактів або зв’язків між фактами. Висловлення виражаються розповідними реченнями.

Просте висловлювання – це просте розповідне речення, відносно якого можна однозначно сказати, істинне воно чи хибне.

Складні висловлювання складаються з простих за допомогою сполучників, яким відповідають логічні операції: унарна операція заперечення $\neg$ («ні»), бінарні операції кон’юнкції & («і»), диз’юнкція $\vee$ («або»), імплікація $\rightarrow$ («якщо…, то…»), еквівалентність $\equiv$ («тоді і тільки тоді»). Символи операцій називаються *пропозиціональними зв’язками.*

Приписування пропозиціональним літерам їх істинностних значень називається *інтерпретацією формули*. Множина всіх інтерпретацій формули утворює її таблицю істинності. Якщо виконати відображення $0 \Leftrightarrow F$, $1 \Leftrightarrow T$, то кожній пропозиціональній зв'язці буде відповідати булева операція, а кожній формулі алгебри висловлювань – булева формула, з чого випливає, що алгебра висловлювань є інтерпретацією булевої алгебри. В зв'язку з цим в ній

зберігаються всі аксіоми і теореми булевої алгебри, зокрема й зображуваність формул алгебри висловлювань у вигляді ДДНФ і ДКНФ.

Тотожно істинна формула називається *тавтологією*. Тотожно хибна формула називається *протиріччям*. Формула, яка приймає істинне значення хоча б на одній своїй інтерпретації, називається *здійсненною.* Формула, яка приймає на одних наборах істинні значення, а на якихось – хибні, називається *нейтральною.*

Проблема можливості розв'язання в алгебрі висловлювань полягає в тому, щоб відшукати ефективну процедуру (алгоритм), за допомогою якої для кожної формули логіки висловлювань можна встановити, чи є вона тавтологією, чи ні. Очевидно, що така процедура для формул логіки висловлювань існує: це побудова таблиць істинності.

Правила формального виведення:

$A \to B, A \models B$	***modus ponens***
$A \to B, B \to C \models A \to C$	***правило силогізму***
$A \to B \models \neg B \to \neg A$	***правило контрапозиції***
$A \to B, \neg B \models \neg A$	***modus tollens***
$A \to B, A \to \neg B \models \neg A$	***правило приведення до абсурду***

1.2. Поняття предиката

Існують такі логічні схеми міркувань, які не можуть бути обґрунтовані в логіці висловлювань. Розглянемо умовивід: «*Всі люди смертні* (*A*). *Сократ – людина* (*B*). *Отже, Сократ смертний* (*C*)». Очевидно, що *C* випливає з *A* і *B*, проте, логічне виведення *A, B* | = *C* неможливо довести в алгебрі висловлювань. Причина полягає у внутрішній структурі висловлювання.

Внутрішню структуру висловлювання можна розділити на суб'єкт і предикат, де суб'єкт є підмет - поняття про предмет думки, а предикат визначає властивість суб'єкта - поняття про особливість предмета думки, наявність якої в ньому стверджується або заперечується (рис. 1.1).

Наприклад, *Сократ* – це суб'єкт, який має властивість бути людиною. Це «чиста» властивість являє собою одномісний предикат, визначений на множині людей: « *x* є людина». Позначимо його *P()*, де *x* – змінна, що позначає так зване «вільне місце предиката». Підставляючи на місце змінної *x* об'єкти з області визначення предиката, отримуємо висловлювання. Таким чином, одномісний предикат, визначений на деякій множині об'єктів, задає властивість, яким ці об'єкти можуть володіти чи не володіти. При підстановці на вільне місце предиката будь-якого об'єкта з його області визначення предикат перетворюється в висловлювання, істинне або хибне. Таким чином, предикат розбиває цю множину на дві області: області істинності і хибності.

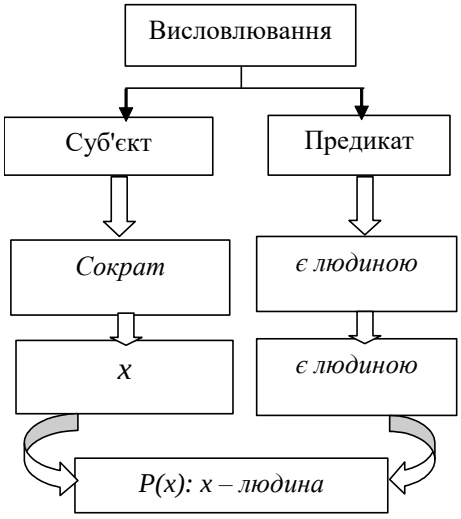


Рис. 1.1. Структура висловлювання

Визначення 1.1. *Одномісним предикатом* $P(x)$, визначеним на множині M, називається вираз, який після підстановки в нього замість x предмета з області визначення M перетворюється в висловлювання. Область визначення предиката називається *предметною областю*. Елементи з області визначення називаються *предметними постійними* (*предметами*). Змінна, від якої залежить предикат, називається *предметною змінною.*

Одномісні предикати традиційно служать для формалізації понять. *Поняття* уявляють собою одиницю мислення. Абстрактне мислення ґрунтується на поняттях, що відображають дійсність, тому абстрактне мислення називають понятійним. Поняття виникають як результат узагальнення множини предметів за системою ознак, загальної тільки для цих виділених предметів.

Ознака - це наявність або відсутність властивості у предмета, а також наявність або відсутність відносин між предметами. Поняття характеризується своїм змістом і обсягом.

Зміст поняття - це система ознак, на основі якої множина предметів узагальнюється в поняття. Наприклад, поняття «квадрат» характеризується ознаками: рівні боки, 4 прямих рівних кута і т.д.

Обсяг поняття - це множина предметів, таких, що узагальнюються і виділяються в поняття, тобто множина предметів, яка характеризується системою ознак, які складають це поняття. Наприклад, обсягом поняття «ліс» служить не множина дерев (це можна віднести до змісту поняття), а «Чорний ліс», «Овруцький ліс», «той ліс, який біля будинку».

Наприклад, поняття «*риба*» можна охарактеризувати як множину всіх живих істот (обсяг поняття), які мають ознаки: живуть у воді, плавають, мають зябра, плавники і хвіст (зміст поняття). Кожну з перерахованих властивостей

можна задати одномісним предикатом, визначеним на множині *всіх живих істот*: *V* (*x*) – *x живе в воді, P* (*x*) – *x плаває, G* (*x*) – *x має зябра, L* (*x*) – *x має плавники, R* (*x*) – *x має хвіст.* Таким чином, поняття риба може бути описано виразом: *V* (*x*) & *P* (*x*) & *G* (*x*) & *L* (*x*) & *R* (*x*). Область істинності цього виразу становить *обсяг* поняття - це всі існуючі риби.

Між обсягом і змістом поняття існує зворотна залежність: чим більший об'єм, тим менше зміст, тобто тим воно більш абстрактно, розпливчато. Наприклад, поняття «*мешканці водних глибин*» можна визначити як «*множина всіх істот, що живуть у воді*». Зміст цього поняття описується предикатом *V*(*x*) – *x живе в воді.* Додавши властивість *P*(*x*) – *x плаває,* ми збільшимо зміст поняття, але зменшимо об'єм: будуть виключені молюски, ракоподібні та інші мешканці водних глибин, що не плавають. Додавши нові властивості, ми ще більше зменшимо обсяг поняття.

Тотожні або рівнозначні поняття збігаються за обсягом. Перехресні поняття - поняття, обсяг кожного з яких має частину загальних елементів. Якщо обсяг одного поняття повністю входить в обсяг іншого, то між поняттями можна встановити відношення підпорядкованості.

Двомісний предикат задає відношення між двома об'єктами. Об'єкти можуть належати одній і тій же, або різним областям визначення. Наприклад, предикат *P*(*x, y*): *x* > *y*, де *x, y* ∈ ***R,*** задає відношення «більше» на множині дійсних чисел; підставив в нього значення, отримаємо висловлювання, наприклад: 5 > 2 = *T*, 6.8 > 10 = *F*. Якщо в предикат *P*(*x, y*): *x* > *y*, підставити значення *y*=0, отримаємо одномісний предикат: *x*>0, який задає *властивість* дійсних чисел бути (або не бути) більше нуля і визначає поняття «*додатні дійсні числа*». На місце змінної в предикат можна підставити функцію. Наприклад, якщо в предикат *P*(*x, y*) підставити на місце *x* функцію *f*(*u, v*)=*u*+*v*, отримаємо новий предикат: *R*(*f*(*u, v*), *y*): *u* + *v* > *y*, що визначає відношення між сумою двох чисел і третім числом.

Інший приклад двомісного предиката: *S*(*x, y*): «*x народився в y році*», де *x* ∈ { *люди* }, *y* ∈ ***N***. Предикат *S*(*x, y*) задає відношення на множині людей і множині натуральних чисел. При заміні *y* на об'єкт з області визначення, наприклад, *y* = 1891 отримаємо одномісний предикат *S*(*x,* 1891), що визначає властивість: «*людина x народилася в 1891 році*». При заміні обох змінних отримаємо висловлювання, наприклад, *«Булгаков народився в 1891 році*».

Таким чином, двомісний предикат задає деяке бінарне відношення на заданих множинах, причому при заміні однієї змінної місність предиката знижується (двомісний предикат стає одномісним), а при заміні обох змінних на предметні постоянні він перетворюється у висловлювання.

У загальному випадку *n*-місний предикат визначає *n*-місне відношення.

Визначення 1.2. *N-місним предикатом*, визначеним на множинах *M*1, *M*2,..., *Mn*, називається вираз, який перетворюється в висловлювання при

заміні кожної предметної змінної на елемент з її області визначення. Якщо всі предметні змінні визначені на одній і теж самій множині, то предикат називається *однорідним*.

Приклади**.** *R*(*x, y, z, t*)*:* «*x народився в році y в місті z, має освіту t*», *x* ∈ { *люди* }, *y* ∈ ***N,*** *z* ∈ { *міста* }, *t* ∈ {*початкова, середня, вища*}. *R*(*x, y, z, t*) – неоднорідний чотиримісний предикат. Однорідний предикат: *Q*(*x, y, z*)*:* «*паралелепіпед має висоту x, ширину y, довжину z*», де *x, y, z* ∈ ***N.***

1.3. Формули логіки предикатів

1.3.1. Операції над предикатами

Предикат можна розглядати як функцію, що визначена на деякій множині об'єктів, та може приймати два значення, *T* і *F*, тобто як булеву функцію. Тому над предикатами визначені всі булеві операції: ¬ (заперечення), & (кон'юнкція), ∨ (диз'юнкція), → (імплікация), ≡ (еквівалентність), а також дві нові операції – операції навішування кванторів: ∀ – загальності і ∃ – існування.

Якщо *P*(*x*) визначає деяку властивість на множині *M*, то формула ∀*x P*(*x*) позначає висловлювання: «для будь-якого предмета *x*∈*M* властивість *P*(*x*) виконується», або «всі *x* мають властивість *P*(*x*)». Значення формули | ∀ *xP* (*x*) | = *T* (істинно), якщо властивість *P* виконано для всіх об'єктів з *M,* і | ∀ *xP* (*x*) | = *F* (хибно), якщо існує хоча б один елемент *x=a, a*∈*M,* для якого властивість *P* не виконана, тобто |*P*(*a*)|=*F*. Наприклад: якщо *P*(*x*): *x смертний, x* ∈ { *люди* }, то ∀*x P*(*x*) - « *всі люди смертні* » (значення формули | ∀*x P*(*x*) | = *T*); якщо *P*(*x*): *x* > 0, *x* ∈ ***R,*** то ∀*x P*(*x*) - « *всі дійсні числа додатні*» (|∀*x P*(*x*)|=*F*).

Формула ∃*x P*(*x*) означає: «існує принаймні один предмет *x*, що володіє властивістю *P*(*x*)», або: «деякі *x* мають властивість *P*(*x*)». Значення формули | ∃*x P*(*x*) | = *T* (істинно), якщо існує хоча б один елемент *x=a, a* ∈ *M,* для якого властивість *P* виконується: | *P*(*a*) | = *T,* значення | ∃*x P*(*x*) | = *F* (хибно), якщо властивість *P* не виконана для всіх об'єктів з *M.* Наприклад: якщо *P*(*x*): *x* > 0, *x* ∈ ***R,*** то ∃*x P*(*x*) - цей вислів: «*деякі дійсні числа – додатні*», тоді | ∃*x P*(*x*) | = *T* ; якщо *P*(*x*)*: x смертний, x* ∈ { *люди* }, то ∃ *x* ¬ *P*(*x*) - *«існують безсмертні люди»* (хибне висловлювання).

Якщо *M* = {a_1, a_2,..., a_n} - кінцева область визначення предиката *P*(*x*), то формули з кванторами можуть бути виражені через кон'юнкцію і диз'юнкцію:

∀*xP*(*x*)=*P*(a_1)&*P*(a_2)&...&*P*(a_n), ∃*xP*(*x*)=*P*(a_1)∨*P*(a_2)∨...∨*P*(a_n).

Таким чином, квантор загальності є узагальненням кон'юнкції, а квантор існування – узагальненням диз'юнкції на нескінченній області визначення.

Квантори $\forall$ і $\exists$ пов'язані один з одним за принципом подвійності (по законам де Моргана):

$\neg \forall x\, P(x) \equiv \exists x \neg P(x)$, $\neg \exists x\, P(x) \equiv \forall x \neg P(x)$.

Наприклад, якщо $P(x)$: «*x смертний*», $x \in$ { *люди* }, то формула $\neg\forall x P(x)$ означає висловлювання: «*не всі люди смертні*», яке еквівалентно висловлюванню «*існують безсмертні люди*», тобто $\exists x \neg P(x)$, а формула $\neg\exists x P(x)$ – «*не існує смертних людей*» еквівалент висловлюванню «*всі люди – безсмертні*», тобто, $\forall x \neg P(x)$.

1.3.2. Визначення формули

Основними символами мови логіки предикатів є:

– пропозіціональние символи $\neg$ і $\rightarrow$,

– квантори загальності $\forall$ і існування $\exists$,

– допоміжні символи (і),

– предметні змінні $x_1, x_2, ..., x_n, ...$,

– предметні постійні $a_1, a_2, ..., a_n, ...$,

– функціональні символи $f_1^1, f_1^2, ..., f_k^j, ...$,

– предикатні символи $P_1^1, P_1^2, ..., P_k^j, ...$

Нижній індекс предикатного або функціонального символу – це номер, який служить для розрізнення однойменних символів з однаковим числом аргументів, верхній індекс вказує число аргументів.

Визначимо поняття *терма* і *формули.*

Визначення терма.

1. Кожна предметна змінна є *терм*.
2. Кожна предметна постійна є *терм.*
3. Функціональний символ $f(t_1, ..., t_n)$, де $t_1, ..., t_n$ – терми, є терм.
4. Інших термів немає.

Визначення формули.

1. $P_i^n(t_1, ..., t_n)$, де P_i^n – предикатний символ, $t_1, ..., t_n$ – терми, є *атомарна* (*елементарна) формула*.
2. Якщо A і B – формули і x – предметна змінна, то формулами є $(\neg A)$, $(A \rightarrow B)$, $(\forall x\, A)$, $(\exists x\, A)$.
3. Інших формул немає.

Вирази $A \,\&\, B$, $A \vee B$, $A \equiv B$ визначаються так само, як в обчисленні L.

Пріоритет операцій повинен бути таким: $\neg$, $\forall$ і $\exists$, $\&$, $\vee$, $\rightarrow$, $\equiv$.

Визначення 1.3. Формула, на яку поширюється дія квантора, називається *областю дії квантора*. Змінні, за якими навішується квантор і потрапляють в його область дії, називаються *пов'язаними змінними*. Змінні, що лежать поза областю дії квантора, називаються *вільними*.

Формула, що не містить вільних змінних, називається *замкненою*. Замкнені формули є висловлюваннями.

Область дії квантора обмежується дужками, якщо вона містить більше одного предиката.

Приклади.

1. На рис. 1.2 наведені приклади формул логіки предикатів і вказані вільні і зв'язані змінні.

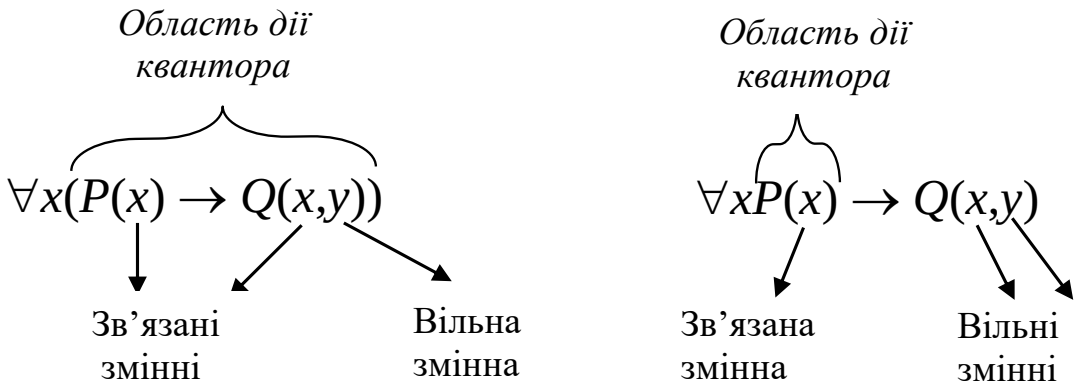


Рис. 1.2. Вільні і зв'язані змінні.

2. Нехай $Q(x, y)$: «*x народився в році y*», $x \in \{$ *люди* $\}$, $y \in \{$ *роки* $\}$, тоді формула $\forall x \exists y\, Q(x, y)$ позначає висловлювання: «*Кожна людина народилася в якомусь році*», а формула $\exists y \forall x\, Q(x, y)$ - вислів:« *Існує такий рік, в якому народилися всі люди*». З цього прикладу видно, що різнойменні квантори в загальному випадку не переставляються.

Визначення 1.4. *Окремим випадком формули* $A(x)$ називається формула, отримана підстановкою деякої формули замість всіх входжень деякої змінної в $A(x)$.

Формула $A(y)$, отримана з $A(x)$ заміною всіх входжень змінної x на y, також є окремим випадком формули $A(x)$.

Визначення 1.5. Кажуть, що *y вільно для x в формулі* $A(x)$, якщо при заміні x на y ні яке вільне входження x не стає зв'язаним в результаті заміни.

Приклад. $\exists x (x = 2y)$, $x, y \in R$. У цій формулі z вільно для y: $\exists x (x = 2z)$, але y не є вільним для x, так як в результаті заміни x на y отримаємо зовсім іншу формулу: $\exists y (y = 2y)$.

1.4. Інтерпретація формул логіки предикатів

Формули мають сенс тільки тоді, коли існує яка-небудь інтерпретація символів, що входять до неї.

Визначення 1.6. Під *інтерпретацією* будемо розуміти систему, що складається з непорожньої множини D, званою *областю інтерпретації*, а також відповідності, що ставить кожній предикатній букві P_i^n деяке відношення на області D, кожній предметній постійній a_i – деякий елемент з області D, кожній функціональній букві f_i^n – деяку m- місцеву операцію на області D (тобто функцію $D^n \to D$).

При заданій інтерпретації всі предметні змінні пробігають всі значення з області D, а логічні зв'язки мають звичайний логічний зміст.

Для заданої інтерпретації будь-яка замкнена формула є висловлюванням, яке істинно або хибно, а формула з вільними змінними виражає відношення на області D, яке може бути істинно (виконано) при одних значеннях змінних і хибно (не виконано) при інших.

Приклади.

1. У таблиці 1.1. наведені три інтерпретації однієї й тієї ж формули.

Таблиця 1.1.

Область інтерпретації ***D***	Інтерпретація	Висловлювання $\forall x\,(P(x) \to Q(x))$
Множина живих істот	$P(x)$: x - риба, $Q(x)$: x живе в воді.	Всі риби живуть у воді.
Множина живих істот	$P(x)$: x - людина, $Q(x)$: x смертний.	Всі люди смертні.
Множина цілих чисел	$P(x)$: x ділиться на 6, $Q(x)$: x ділиться на 3.	Всі числа, які діляться на 6, діляться на 3.

2. Формула $\exists x\,\exists y\,P(f(x, y), a)$ є висловлюванням.

Предикат $P(v, u)$ – двомісний, змінні x, y – зв'язані, a – предметна постійна. Істинність або хибність цього висловлювання можна дізнатися лише при заданій інтерпретації. Задамо наступну інтерпретацію: область інтерпретації D – множина дійсних чисел ***R,*** константа a=1, функціональний символ $f(x, y) = x^2 + y^2$, предикат $P(u, t)$: $u = t$. Тоді формула має вигляд: $\exists x\,\exists y\,(x^2 + y^2 = 1)$ і задає рівняння кола. Вона істинна, так як існують такі x та y, що $x^2 + y^2 = 1$.

3. Інтерпретація може бути частковою, наприклад, $f(x, y) = x^2 + y^2$, $t = r^2$. Тоді формула $\exists x\,\exists y\,(x^2 + y^2 = r^2)$ –одномісний предикат, який визначає властивість точок на дійсній площини належати (або не належати) деякій окружності з радіусом r.

Визначення 1.7. Інтерпретація називається *моделлю* для даної множини формул Г, якщо кожна формула з Г істинна в даній інтерпретації.

Визначення 1.8. Формула називається *здійсненною,* якщо існує хоча б одна інтерпретація, на якій формула істинна.

Визначення 1.9. Формула називається *логічно-загальнозначущої* (*ЛЗЗ)*, якщо вона істинна на будь-який інтерпретації для будь-яких значень змінних.

Так само, як тавтології, логічно-загальнозначущі формули позначаються: $| = A (x)$.

Визначення 1.10. Формула, яка хибна на будь-якій інтерпретації при будь-яких значеннях змінних, називається *протиріччям*.

Логічно-загальнозначущі формули є виділеними формулами алгебри предикатів.

Так як область визначення предиката може бути нескінченною, то, очевидно, що побудова таблиці істинності не може служити алгоритмом для визначення логічної-загальнозначущості формул. Однак існують інші способи, які в окремих випадках дозволяють визначити логічну-загальнозначущість, здійсненність або еквівалентність формул. Можна будувати таблиці істинності формул алгебри предикатів для часткових інтерпретацій на обмежених областях. Наприклад, візьмемо область інтерпретації, що складається з двох довільних елементів: $D = \{ a, b \}$. Побудуємо таблицю істинності формул: $E_1 = \exists x P (x)$ і $E_2 = \forall x P (x)$. Одномісний предикат на області визначення з двох елементів може приймати одне з чотирьох значень, які визначаються таблицями істинності (табл. 1.2).

Табліца.1.2

x	$P_1(.)$	$P_2(.)$	$P_3(.)$	$P_4(.)$
a	F	F	T	T
b	F	T	F	T

Формули E_1 і E_2 прийматимуть на цих інтерпретаціях наступні значення (табл. 1.3).

Табліца.1.3

$P(.)$	$\exists xP(x)$	$\forall xP(x)$
P_1	F	F
P_2	T	F
P_3	T	F
P_4	T	T

Побудуємо таблиці істинності на області інтерпретації з двох елементів $D = \{ a, b \}$ для наступних формул:

$E_1 = \forall yP(y) \to \exists xQ(x)$, $E_2 = \forall y(P(y) \to \exists xQ(x))$, $E_3 = \forall y \exists x(P(y) \to Q(x))$. Для цих формул існує 16 інтерпретацій, так як кожен з одномісних предикатів P і Q приймає по 4 значення відповідно до таблиці 1.2. Розглянемо обчислення значень формул на інтерпретацій P_2, Q_1.

$E_1 = \forall y P_2(y) \to \exists x Q_1(x) = F \to F = T;$

$$E_2 = \forall y(P_2(y) \to \exists x Q_1(x)) = \forall y\left(\frac{P_2(1) \to \exists x Q_1(x)}{P_2(2) \to \exists x Q_1(x)}\right) = \forall y\left(\frac{F \to F = T}{T \to F = F}\right) = F$$

$E_3 = \forall y\, \exists x(P_2(y) \to Q_1(x)) =$

$$= \forall y\left(\frac{\exists x(P_2(1) \to Q_1(x)}{\exists x(P_2(2) \to Q_1(x)}\right) = \forall y\left(\frac{\exists x\left(\frac{P_2(1) \to Q_1(1)}{P_2(1) \to Q_1(2)}\right)}{\exists x\left(\frac{P_2(2) \to Q_1(1)}{P_2(2) \to Q_1(2)}\right)}\right) = \forall y\left(\frac{\exists x\left(\frac{F \to F = T}{F \to F = T}\right) = T}{\exists x\left(\frac{T \to F = F}{T \to F = F}\right) = F}\right) = F$$

Істинності значення E_1, E_2, E_3 для восьми інтерпретацій наведено в табл. 1.4. З таблиці видно, що формула E_1 не є еквівалентною формулам E_2 і E_3, а формули E_2 і E_3, можливо, еквіваленти, - для остаточного рішення потрібно розглянути інтерпретації, що залишилися.

Таблиця. 1.4.

P (.)	Q (.)	E_1	E_2	E_3
P_1	Q_1	T	T	T
P_1	Q_2	T	T	T
P_1	Q_3	T	T	T
P_1	Q_4	T	T	T
P_2	Q_1	T	F	F
P_2	Q - $_2$	T	T	T
P_2	Q_3	T	T	T
P_2	Q_4	T	T	T

1.5. Логічно загальнозначущі формули логіки предикатів

1.5.1. Основні логічно-загальнозначущі формули алгебри предикаті

Основні логічно-загальнозначущі формули логіки предикатів наведені в таблиці 1. 5.

Таблиця 1.5.

$\forall\, xP\,(x) \to P\,(y)$	правило універсальної конкретизації;
$P\,(a) \to \exists\, x\,(P\,(x))$	правило екзистенціального узагальнення;
$\neg\, \forall\, xP\,(x) \equiv \exists\, x \neg\, P\,(x)$	правило де Моргана
$\neg\, \exists\, xP\,(x) \equiv \forall\, x \neg\, P\,(x)$	правило де Моргана
$\forall\, x\,(P\,(x)\ \&\ Q\,(x)) \equiv \forall\, xP\,(x)\ \&\ \forall\, xQ\,(x)$	закон пронесення $\forall$ через &

$\exists x (P(x) \vee Q(x)) \equiv \exists xP(x) \vee \exists xQ(x)$	закон пронесення $\exists$ через $\vee$
$\forall xP(x) \vee \forall xQ(x) \to \forall x (P(x) \vee Q(x))$	закон пронесення $\forall$ через $\vee$
$\exists x (P(x) \,\&\, Q(x)) \to \exists xP(x) \,\&\, \exists xQ(x)$	закон пронесення $\exists$ через $\&$
$(\forall x (P(x) \to Q(x)) \to (\forall xP(x) \to \forall xQ(x))$	закон пронесення $\forall$ через $\to$
$(\exists xP(x) \to \exists xQ(x)) \to \exists x (P(x) \to Q(x))$	закон пронесення $\exists$ через $\to$
$\forall x (P(x) \equiv Q(x)) \to (\forall xP(x) \equiv \forall xQ(x))$	закон пронесення $\forall$ через $\equiv$
$\forall x (P(x) \,\&\, B) \equiv \forall xP(x) \,\&\, B$	B не містить входжень x
$\forall x (P(x) \vee B) \equiv \forall xP(x) \vee B$	B не містить входжень x
$\exists x (P(x) \,\&\, B) \equiv \exists xP(x) \,\&\, B$	B не містить входжень x
$\exists x (P(x) \vee B) \equiv \exists xP(x) \vee B$	B не містить входжень x
$\forall x (P(x) \to B) \equiv (\exists xP(x) \to B)$	B не містить входжень x
$\exists x (P(x) \to B) \equiv (\forall xP(x) \to B)$	B не містить входжень x
$\forall x \forall yP(x, y) \equiv \forall y \forall xP(x, y)$	закон перестановки кванторів $\forall$
$\forall x \forall yP(x, y) \to \forall xP(x, x)$	
$\exists x \exists yP(x, y) \equiv \exists y \exists xP(x, y)$	закон перестановки кванторів $\exists$
$\exists xP(x, x) \to \exists x \exists yP(x, y)$	
$\exists y \forall xP(x, y) \to \forall x \exists yP(x, y)$	закон перестановки кванторів $\exists$ і $\forall$
$\forall xP(x) \to \exists xP(x)$	
$(\forall xP(x) \to \exists xQ(x)) \equiv \exists x (P(x) \to Q(x))$	

$(\exists\, xP\,(x) \to \forall\, xQ\,(x)) \to \forall\, x\,(P\,(x) \to Q\,(x))$	
$\forall\, xP\,(x) \equiv \forall\, yP\,(y)$	якщо y вільно для x в $P\,(x)$
$\exists\, xP\,(x) \equiv \exists\, yP\,(y)$	якщо y вільно для x в $P\,(x)$

Кожна логічно загальнозначуща формула виражає деяке істинне висловлювання щодо властивостей об'єктів. Наприклад, логічно-загальнозначуща формула $\exists\, x\,(P\,(x)\ \&\ Q\,(x)) \to \exists\, xP\,(x)\ \&\ \exists\, xQ\,(x)$ виражає той факт, що, якщо деякі об'єкти володіють відразу двома властивостями P і Q, то існують об'єкти, що володіють властивістю P, і об'єкти, що володіють властивістю Q. Так, якщо існують лікарі-шарлатани, то існують люди, які є лікарями, і існують шарлатани. Очевидно, що зворотна імплікація $\exists\, xP\,(x)\ \&\ \exists\, xQ\,(x) \to \exists\, x\,(P\,(x)\ \&\ Q\,(x))$ буде істинна далеко не завжди: з того, що існують лікарі і існують шарлатани, ще не випливає, що існують лікарі-шарлатани, - ці дві множини можуть не перетинатися.

Нижче наводяться змістовні інтерпретації деяких логічно-загальнозначущих формул.

$\forall\, xP\,(x) \to P\,(y)$	Якщо все люди смертні, то смертна будь-яка людина.
$P\,(a) \to \exists\, x\,(P\,(x))$	Якщо кішка a - сіра, то існують сірі кішки.
$\neg\, \forall\, xP\,(x) \equiv \exists\, x \neg\, P\,(x)$	Не всі кішки сірі ≡ Існують не сірі кішки.
$\neg\, \exists\, xP\,(x) \equiv \forall\, x \neg\, P\,(x)$	Не існує сірих кішок ≡ Всі кішки не сірі.
$\forall\, x\,(P\,(x)\ \&\ Q\,(x)) \equiv \forall\, xP\,(x)\ \&\ \forall\, xQ\,(x)$	Всі кішки з вусами і з хвостами ≡ Кожна кішка має вуса і кожна кішка має хвіст.
$\exists\, x\,(P\,(x) \vee Q\,(x)) \equiv \exists\, xP\,(x) \vee \exists\, xQ\,(x)$	Деякі кішки білі або чорні ≡ Існує хоча б одна біла кішка або існує хоча б одна чорна кішка.
$\forall\, x\,(P\,(x) \to Q\,(x)) \to (\forall\, xP\,(x) \to \forall\, xQ\,(x))$	Якщо все сторожові собаки злі, то якщо всі собаки - сторожові, то всі вони злі. Зворотне не завжди вірно.
$(\exists\, xP\,(x) \to \exists\, xQ\,(x)) \to \exists\, x\,(P\,(x) \to Q\,(x))$	Якщо з того, що існують собаки, слідує, що існують гавкаючі істоти, то існують такі собаки, які гавкають. Зворотне не завжди вірно.

1.5.2. Перевірка загальнозначущості формул алгебри предикатів

Перевірка логічної загальнозначущості формул може бути здійснена зведенням до протиріччя, тобто методом редукції. Допускаємо, що існує така інтерпретація формули *E,* на якій вона приймає хибне значення, тобто $| E^* | = F$, і пробуємо знайти таку інтерпретацію. Якщо в результаті отримуємо протиріччя, це означає, що таких інтерпретацій не існує, і, отже, формула логічно-загальнозначуща.

Приклад. Розглянемо формулу $\forall x\ (A\ (x) \vee B) \equiv \forall x\ A\ (x) \vee B$, де B не залежить від x. Допустимо, що існує така інтерпретація, на якій формула хибна: $| \forall\ x\ (A^*\ (x) \vee B^*) \rightarrow \forall\ x\ A^*\ (x) \vee B^* | = F$. Це можливо, якщо $|\forall x\ (A^*(x) \vee B^*)| = T$, а $| \forall\ x\ A^*\ (x) \vee B^* | = F$. З останньої рівності випливає, що $| B^* | = F$ і $|\forall x\ (A^*\ (x))| = F$. Якщо $| \forall\ x\ (A^*\ (x)) | = F$, то існує хоча б одне значення $x = a$, таке, що $| A^*\ (a) | = F$.

Формула $| \forall\ x\ (A^*(x) \vee B^{*)} | = T$. Але в області інтерпретації цієї формули існує значення $x = a$, для якого $| A^*\ (a) | = F$ і $| B^* | = F$. Можливо, що існує інше значення $x = b$, для якого $| A^*\ (b) | = F$.

Тоді $| \forall\ x\ (A^*\ (x) \vee B^{*)} | = \forall \begin{Bmatrix} A^*(a) \vee B^* = F \vee F = F \\ A^*(b) \vee B^* = T \vee F = T \end{Bmatrix} = F$, що суперечить

припущенню $| (A^*\ (x) \vee B^*) | = T$.

Перевіримо виконання загальнозначущості в іншу сторону. Припустимо, що $| \forall\ x\ A^*\ (x) \vee B^* \rightarrow \forall\ x\ (A^*\ (x) \vee B^*) | = F$. тоді $| \forall\ x\ (A^*\ (x) \vee B^*) | = F$, і $| \forall\ x\ A^*\ (x) \vee B^* | = T$. З $| \forall\ x\ (A^*\ (x) \vee B^*) | = F$ випливає, що існує таке $x = a$, що $| A^*\ (a) \vee B^* | = F$. Звідси випливає, що $| A^*\ (a) | = F$, $| B^* | = F$. Отже, в області визначення предиката $A\ (x)$ існує значення $x = a$, при якому предикат $| A^*\ (a) | = F$, значить, $| \forall\ x\ A^*\ (x) | = F$. Тоді формула $| \forall\ x\ A^*\ (x) \vee F | = F$, що суперечить нашому припущенню. Отже, формула $\forall x\ (A\ (x) \vee B) \equiv \forall\ x\ A\ (x) \vee B\ \epsilon$ логічно-загальнозначущою.

1.6. Логічне слідування в логіці предикатів

1.6.1. Визначення логічного слідування

Визначення 1.11. Кажуть, що формула B логічно випливає з формули A, якщо в будь-який інтерпретації, в якій A приймає дійсні значення, B також приймає істинні значення. Позначення: $A | = B$.

У загальному випадку формула B є логічним наслідком множини формул Г, якщо вона істинна на всіх тих інтерпретаціях, на яких виконані (істинні одночасно) все формули з Г.

Визначення 1.12. Кажуть, що формула A рівносильна, або логічно-еквівалентна, формулі B, якщо кожна з них логічно тягне іншу, тобто якщо $A \models B$ і $B \models A$. Позначення: $A \Leftrightarrow B$.

Твердження.

1. $A \models B$ тоді і тільки тоді, коли $\models A \to B$.
2. $A_1, A_2, \ldots, A_n \models B$, тоді і тільки тоді, коли $\models A_1 \& A_2 \& \ldots \& A_n \to B$.
3. $A \Leftrightarrow B$ тоді і тільки тоді, коли $\models A \equiv B$.
4. Якщо $A \models B$ і $|A| = T$, то $|B| = T$ в деякій інтерпретації.
5. Якщо $\Gamma \models B$ і $\forall i \, (|\Gamma_i| = T)$, то $|B| = T$.

1.6.2. Основні правила виведення логіки предикатів

Розглянемо деякі логічні слідування, які виконані в логіці предикатів. Кожне таке логічне проходження задає правило виведення в логіки предикатів; деякі з них будуть використані в формальної теорії предикатів.

1. Правило універсальної конкретизації (УК):

$\forall x A(x) \models A(y)$, якщо y вільно для x в $A(x)$.

Доведення. Потрібно довести, що якщо $|\forall x A^*(x)| = T$, то $|A^*(y)| = T$ в деякій інтерпретації D. Припустимо $|A^*(y)| = F$. Тоді існує $y = b$, $b \in D$, таке що $|A^*(b)| = F$. Але за умовою на області D формула $|\forall x A^*(x)| = T$, а так як $b \in D$, то $|\forall x A^*(x)| = F$ на D. Отримане протиріччя доводить теорему.

2. Правило екзистенціальної конкретизації (ЕК):

$\exists x A(x) \models A(b)$, де $b \in D$.

Доведення. Припустимо, $|\exists x A^*(x)| = T$ в деякій інтерпретації D. Тоді існує таке $x = b$, $b \in D$, що $|A^*(b)| = T$, і отже, $|\exists x A^*(x)| = T$.

3. Правило екзистенціального узагальнення (ЕУ):

$A(y) \models \exists x A(x)$, де x вільно для y в $A(y)$.

Доведення. Якщо $|A^*(y)| = T$ в деякій інтерпретації D, то існує $y = b$, $b \in D$, таке що $|A^*(b)| = T$. Отже, $|\exists x A^*(x)| = T$ в інтерпретації D.

Частний випадок, $A(b) \models \exists x A(x)$.

4. Правило загальності:

$C \to A(x) \models C \to \forall x (A(x))$.

Доведення. За умовою $|C \to A^*(x)| = T$ в інтерпретації D. Це можливо, якщо

a) $|C| = F$, тоді $|C \to A^*(x)| = T$ і $|C \to \forall x A^*(x)| = T$;

b) $|C| = T$, $|C \to A^*(x)| = T$, отже, $|A^*(x)| = T$ в інтерпретації D для будь-якого x, значить $|C \to \forall x A^*(x)| = T$.

5. Правило існування:

$A(x) \to C \models \exists x A(x) \to C$.

Доведення. $| A^*(x) \to C | = T$ в деякій інтерпретації *D.* Припустимо $| \exists x A^*(x) \to C | = F$ в інтерпретації *D.* Тоді $| C | = F$, (*C* не залежить від *x)* і $| \exists x A^*(x) | = T$, отже, існує $x = b$, таке що $| A^*(b) | = T$ і $| A^*(b) \to C | = F$, в той час, як за умовою $| A^*(x) \to C | = T$. Отримане протиріччя доводить теорему.

6. Правило узагальнення *Gen* (від англійського слова Generalization):

якщо $\Gamma | = A(x)$, то $\Gamma | = \forall x A(x)$, якщо *x* не входить вільно ні в одну з формул Γ.

Доведення. Припустимо, обрана область інтерпретації *D* і проведена заміна в *A* всіх вільних змінних на елементи з *D,* наприклад, $x = b \in D$. Тоді $| A^*(b) | = T$, так як $| \Gamma_i | = T$ для будь-якого *i.* Так як *x* не входить вільно ні в одну з формул Γ, то в множині Γ заміни *x* на *b* не було і, отже, для $x \in D$, такого що $| A^*(x) | = T$, $\Gamma | = A^*(x)$, отже, $\Gamma | = \forall x A^*(x)$.

1.7. Обчислення предикатів першого порядку – формальна теорія К

Оскільки побудова таблиць істинності для будь-якої формули не представляється можливою для перевірки загальнозначущості формул теорії предикатів, аксіоматичний метод необхідний для дослідження формул, що містять квантори. Розглянемо формальну теорію *К* – числення предикатів першого порядку.

Обчислення предикатів, яке не містить предметних констант, функціональних символів, предикатів і власних аксіом, називається чистим. Обчислення предикатів, що містить предметні константи та/або функціональні символи та/ або предикатні символи і власні аксіоми, що зв'язують їх, називаються прикладними. Обчислення предикатів, в якому квантори можуть пов'язувати тільки предметні змінні, але не можуть пов'язувати функціональні символи або предикати, називається обчисленням першого порядку. Обчислення, в яких квантори можуть пов'язувати не тільки предметні змінні, але і функціональні символи, предикати або інші множини об'єктів, називаються численнями вищих порядків.

Практика показує, що обчислення предикатів першого порядку виявляється достатнім для формалізації змістовних теорій у всіх розумних випадках.

Символами **теорії *К*** служать ті ж самі символи логіки предикатів: пропозиціональні зв'язки $\to$, $\neg$, $\forall$, $\exists$, допоміжні символи (,), множини предметних змінних: $x_1, x_2, \ldots$, предметних постійних: $a_1, a_2, \ldots$, функціональні символи: f_i^n, $i=1,\ldots,k$, $n=0,\ldots,m$, предикатні символи: P_i^n, $i=1,\ldots,k$, $n=0,\ldots,m$. Визначення терма, формули і пропозиціональних зв'язок &, $\vee$, $\equiv$ залишаються в силі для теорії першого порядку.

Аксіоми теорії K розбиваються на *логічні* аксіоми і *власні*.

Логічні аксіоми. Хоч би якими були формули A, B, C теорії K, такі формули є логічними аксіомами теорії K.

$A1$ $A \to (B \to A)$.

$A2$ $(A \to (B \to C)) \to ((A \to B) \to (A \to C)$.

$A3$ $(\neg B \to \neg A) \to ((\neg B \to A) \to B)$.

$A4$ $\forall x A(x) \to A(y)$, якщо y вільно для x у формулі $A(x)$.

$A5$ $\forall x (A \to B(x)) \to (A \to \forall x B(x))$, якщо A не містить вільних входжень x.

Власні аксіоми формулюються окремо для кожної конкретної змістовної предметної області.

Теорія першого порядку, що не містить власних аксіом, називається *обчисленням предикатів першого порядку.*

Правилами виведення у всякій теорії першого порядку є:

1) *modus ponens* (MP): з A і $A \to B$ слідує B,

2) правило *узагальнення Gen:* з $\Gamma \mid = A(x)$, слідує $\Gamma \mid = \forall x A(x)$, якщо x не входить вільно ні в одну з формул Γ.

Моделлю теорії першого порядку K називається всяка інтерпретація, в якій істинні всі аксіоми теорії K. Якщо правила виведення MP і *Gen* застосовуються до істинних в даній інтерпретації формул, то результатом є формули, також справжні в тій же інтерпретації. Отже, будь-яка теорія K істинна у всякій її моделі.

Множина логічних наслідків логічних аксіом теорії K збігається з множиною теорем теорії K. Для обчислення предикатів першого порядку множина його теорем збігається з множиною логічно-загальнозначущих формул. Аксіоми $A1$, $A2$, $A3$ теорії K і правило MP визначені в теорії L, отже, всі теореми теорії L включені в множину теорем теорії K.

Метатеореми про дедукції в теорії K може бути сформульовані в ослабленому вигляді.

Метатеореми про дедукції. Якщо існує висновок формули B з множини гіпотез Γ і формули A, і в цьому висновку ні при якому застосуванні правила *Gen* до формул, що залежать від A, не зв'язується квантором ніяка вільна змінна формули A, то $\Gamma \mid\!\!- A \to B$.

Слідування 1. Якщо існує висновок $\Gamma, A \mid\!\!- B$, і в цьому висновку жодного разу не застосовувалося правило *Gen* до формул, що залежать від A, то $\Gamma \mid\!\!- A \to B$.

Слідування 2. Якщо існує висновок $\Gamma, A \mid\!\!- B$, де A – замкнена формула, то $\Gamma \mid\!\!- A \to B$.

1.8. Доказ логічних слідувань в логіці предикатів

1.8.1. Формалізація речень природної мови

Мова логіки предикатів традиційно служить для формалізації висловлювань природної мови. Вважається, що «мова предикатів» є вродженою і лежить в основі засвоєння рідної мови і форм пізнавальної активності.

Судження природної мови діляться на одиничні, загальні і приватні.

Одиничне судження висловлює приналежність (чи неналежність) предмета класу предметів, наприклад: «Україна - європейська країна»

Загальні судження висловлюють включення (або невключення) класу предметів в інший клас, наприклад: «Всі судді - юристи».

Приватні судження висловлюють часткову приналежність (чи неналежність) предметів одного класу до іншого класу, наприклад, «Деякі вчені є викладачами».

Приклад. Розглянемо область визначення $M = \{люди\}$ з заданими на ній предикатами: $J(x)$ – *x поет;* $L(x)$ – *x письменник;* $S(x)$ – *x композитор;* $A(x,y)$ – *x любить y.*

Поняття «*письменник»* можна визначити як множину всіх людей, які пишуть літературні твори. Поняття «*поет»* можна визначити як множину людей, які пишуть літературні твори, але неодмінно у віршованій формі. Таким чином, множина *поетів* є підмножиною множини *письменників,* тобто властивість *бути поетом* тягне властивість *бути письменником,* і область істинності предиката J включена в область істинності предиката L (див. рис.1.3[1]), тобто справедливий вислів: *кожен поет є письменником,* що можна виразити у вигляді формули: $\forall x\,(J(x) \rightarrow L(x))$.

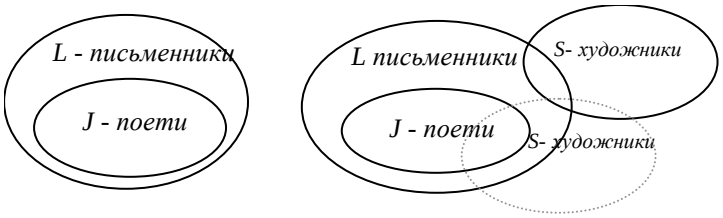


Рис. 1.3. Області визначення предикатів

Розглянемо висловлювання: «*Деякі письменники - художники».* Це висловлювання істинно, якщо існують такі об'єкти, які є одночасно

[1] При створенні загальної теорії силогістичної виведення Дж. Еріксон висунув теоретико-множинну модель логічного міркування, в якій робиться припущення, що посилки подумки репрезентуються у вигляді кіл Ейлера (інший підхід - аналіз діз'юнктівних порцій, третій - на основі раціонального мислення, тобто з використання математичної логіки).

і письменниками, і художниками: $\exists x\ (L\ (x)\ \&\ S\ (x))$, тобто області істинності предикатів $L(x)$ і $S(x)$ перетинаються: $L \cap S$. Чи випливає з цього, що *існують поети-художники*? Ні, області істинності предикатів $J\ (x)$ і $S\ (x)$ можуть перетинатися, а можуть і не перетинатися (див. рис.1.3). Ми могли б сказати: «*Можливо, існують поети-художники»,* - однак категорію *можливості* не можна виразити в класичній теорії предикатів 1-го порядку.

Формалізуємо деякі інші висловлювання:

$\exists x\ (S\ (x)\ \&\ \forall y\ (L\ (y) \rightarrow A\ (x, y)))$	*деякі художники люблять всіх письменників;*
$\exists x\ (S\ (x)\ \&\ \forall y\ (A\ (x, y) \rightarrow L\ (y)))$	*деякі художники люблять тільки письменників;*
$\exists x\ (S\ (x)\ \&\ \exists y\ (L\ (y)\ \&\ A\ (x, y)))$	*деякі художники люблять деяких письменників;*
$\forall x\ (S\ (x) \rightarrow \forall y\ (J\ (y) \rightarrow \neg A\ (x, y)))$	*художники не люблять поетів (або жоден з художників не любить поетів).*

Для останнього прикладу хотілось би продемонструвати формальну ідентичність обох речень, що випливає з правил формалізації та законів де Моргана.

1. «(всі) *Художники не люблять поетів»* – $\forall x\ (S\ (x) \rightarrow \forall y\ (J\ (y) \rightarrow \neg A\ (x, y)))$.

2. «*Жоден з художників не любить поетів*», тобто, «Не існує такого художника, який би любив поетів», тобто, «Не існує такого художника, щоб знайшовся такий поет, якого він любить». Формалізуємо останній варіант речення: $\neg\exists x\ (S\ (x)\ \&\ \exists y\ (J\ (y)\ \&\ A\ (x, y)))$ = далі - за законами де Моргана =

$= \forall x \neg\ (S\ (x)\ \&\ \exists y\ (J\ (y)\ \&\ A\ (x, y))) = \forall x\ (\neg S\ (x) \vee \neg\exists y\ (J\ (y)\ \&\ A\ (x, y))) =$

$= \forall x\ (\neg S\ (x) \vee \forall y \neg\ (J\ (y)\ \&\ A\ (x, y))) = \forall x\ (\neg S\ (x) \vee \forall y\ (\neg J\ (y) \vee \neg A\ (x, y))) =$

= за законами еквівалентних перетворювань =

$= \forall x\ (S\ (x) \rightarrow \forall y\ (J\ (y) \rightarrow \neg A\ (x, y)))$.

1.8.2. Основні схеми суджень

Суб'єкт і предикат називаються термінами суджень. Зв'язка в судженні відображає зв'язок, який існує між предметом думки (суб'єктом) і відповідним властивістю. Вона вказує на наявність або відсутність у предмета судження властивості, ознаки, про яку йде мова в предикаті.

Слова, які виражають зв'язку, часто опускаються. Наприклад, *флегматик - /це є/ один з видів темпераментів; три - просте число.* Слід чітко розмежовувати поняття предмета і суб'єкта суджень. Предмет судження - це реальний предмет, про який йде мова в судженні, а суб'єкт - поняття про реальний предмет, який виступає предметом судження.

Залежно від обсягу суб'єкта судження діляться на загальні (всі) і часткові (деякі), в тому числі і поодинокі. Загальне судження - судження, в якому по кожному мислимому суб'єкту елемента множини стверджується або заперечується певна ознака. Часткове судження - судження, в якому міститься знання про наявність або відсутність ознаки у частини предметів мислимих в суб'єкті, а про наявність або відсутність цієї ознаки для інших предметів, ми можемо нічого не знати (одиничне судження - окремий випадок часткового, судження про суб'єкт, у якого є єдине поняття, наприклад, *Київ - столиця України,* але в логіці, поодинокі судження особливим чином не виділяються).

За якістю, тобто за характером зв'язки, судження можна розділити на позитивні і негативні. У стверджувальних - обсяг суб'єкта включається до обсягу предиката. Тобто, у судженнях констатується наявність ознаки у певного предмета або множини. А в негативних - обсяг виключається, у судженнях констатується відсутність ознаки у предметів, мислимих в суб'єкті судження. В цілому, негативні судження несуть менше інформації в порівнянні зі стверджувальними. Тому в логіці частіше зустрічаються судження, що відповідають правилу визначення понять, згідно яким визначення повинні бути стверджувальними.

Якщо за основу поділу суджень на види брати і кількісну і якісну характеристику, то можна виділити чотири основні схеми.

Загально стреджувальні - наявність ознаки для кожного предмета.

Всі *S* є *P*. Позначається[2] буквою **A.**

Загально негативне - констатується відсутність ознаки в кожному предметі, що мислиться в суб'єкті судження. Всі *S* не є *P*. Позначається буквою **E.**

Частково стреджувальне - наявність ознаки для частини предметів. Деякі *S* є *P*. Позначається буквою **I.**

Частково негативне - відсутність ознаки для частини предметів. Деякі *S* не є *P*. Позначається буквою **O.**

Наведемо ще кілька прикладів формалізації речень природної мови.

1) Загально стверджувальне судження :

$$A\text{: Всі } S \text{ суть } P\text{: } \forall x\,(S\,(x) \rightarrow P\,(x)).$$

Приклад. Нехай $y \in \{люди\}$, $x \in \{твори\}$. На цих областях задані предикати: $P\,(x)$: *y - письменник,* $V\,(x)$: *y - поет,* $W\,(y, x)$: *y пише x,* $N\,(x)$ *: x - роман,* $K\,(x)$ *: x - конспект,* $C\,(x)$ *: x - вірші,* $U\,(x)$ *: x - підручник.* Розглянемо два поняття: «*підручники*» і «*конспекти*». Поняття «*підручники*» володіє тією властивістю, що це книги, за якими вчаться. Предикат $U\,(x)$ серед всіх книг виділяє ті, які є підручниками. За конспектами також вчаться, однак,

[2] Позначення видів суджень йде від латинських «affirmo» (стверджую) - голосними A і I позначаються два види стверджувальних суджень, і голосними E і O позначаються два види негативних суджень від лат. «nero» (заперечую).

конспекти написані від руки. Тому конспекти є підмножиною підручників (див. рис. 1.4). Звідси випливає, що «кожен конспект є підручником», або «всі конспекти – підручники», що виражається формулою: $\forall x\,(K\,(x) \rightarrow U\,(x))$.

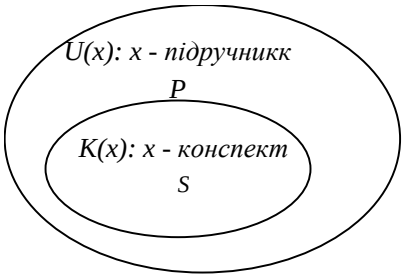


Рис. 1.4. $\forall x\,(K\,(x) \rightarrow U\,(x))$ – *Всі конспекти - підручники.*

2) Загально негативне судження:

E: Жодне S не є P : $\forall x\,(S\,(x) \rightarrow \neg P\,(x))$.

Приклад. Розглянемо два поняття: *«конспекти» і «романи».* Очевидно, що області істинності цих предикатів не перетинаються (див. рис. 1.5), тобто «Жоден конспект не є романом», що виражається формулою: $\forall x\,(K\,(x) \rightarrow \neg N\,(x))$.

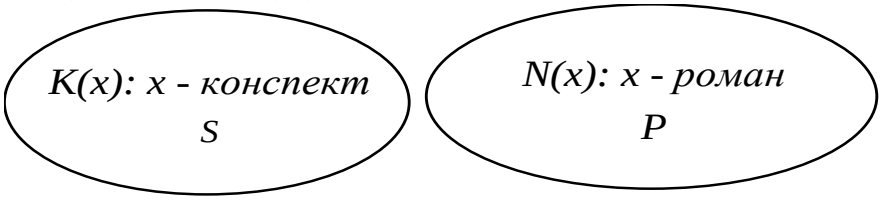


Рис. 1.5. $\forall x\,(K\,(x) \rightarrow \neg N\,(x))$ - *жоден конспект не є романом.*

3) Частково стверджувальне судження:

I: Деякі S суть P - $\exists x\,(S\,(x)\ \&\ P\,(x))$.

Приклад. Поняття «*романи*» і «*вірші*» мають пересічні обсяги (рic.1.6), – як відомо, існують романи у віршах. Твердження «деякі романи написані в віршах» виражається формулою: $\exists x\,(N\,(x)\ \&\ C\,(x))$.

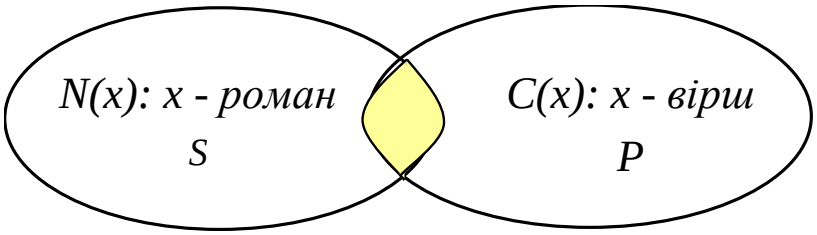


Рис. 1.6. $\exists x\,(N\,(x)\ \&\ C\,(x))$ - *деякі романи - вірші.*

4) Частково негативне судження:

O: Деякі S не є P - $\exists x\,(S\,(x)\ \&\ \neg P\,(x))$.

Приклад. Розглянемо твердження: «деякі романи - не вірші»: $\exists x\ (N\ (x)\ \&\ \neg C\ (x))$, «деякі конспекти - не романи»: $\exists x\ (K\ (x)\ \&\ \neg N\ (x))$. Області істинності відповідних предикатів можуть перетинатися, а можуть і не перетинатися (див. ис. 1.7).

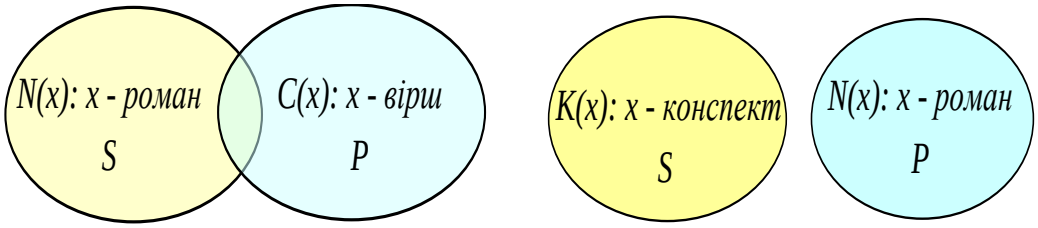


Рис. 1.7. $\exists x\ (N\ (x)\ \&\ \neg C\ (x))$ - деякі романи - не вірші;
$\exists x\ (K\ (x)\ \&\ \neg N\ (x))$ - деякі конспекти - не роман.

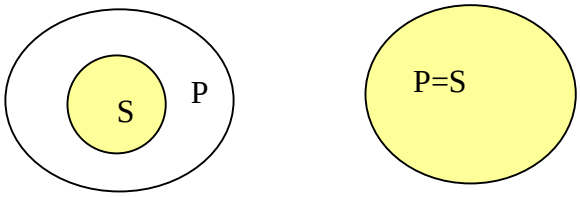
Інші приклади формалізації висловлювань наведені в таблиці 1.5.

Таблиця 1.5.

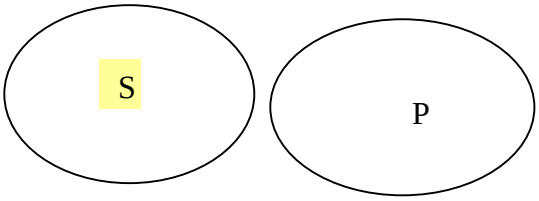
Всі конспекти - підручники.	$\forall x\ (K\ (x) \rightarrow U\ (x))$
Конспект з математики (М) – підручник.	K (М) $\&$ U (М)
Жоден підручник не написаний у віршах.	$\forall x\ (U\ (x) \rightarrow \neg C\ (x))$
Деякі романи написані у віршах.	$\exists x\ (N\ (x)\ \&\ C\ (x))$
«Енеїда» - це роман у віршах.	$N\ (E)\ \&\ C\ (E)$
Поети пишуть вірші.	$\forall y\ (V\ (y) \rightarrow \forall x\ (C\ (x) \rightarrow W\ (y, x))) =$ $= \forall y\ \forall x\ \ (V\ (y) \rightarrow\ \ (C\ (x) \rightarrow W\ (y, x)))$
Деякі письменники пишуть тільки романи.	$\exists y\ (P\ (y)\ \&\ \forall x\ (W\ (y, x) \rightarrow N\ (x))) =$ $= \exists y \forall x\ \ (P\ (y)\ \&\ (W\ (y, x) \rightarrow N\ (x)))$
Письменник Булгаков писав романи.	$P\ (Б)\ \&\ \forall x\ (N\ (x) \rightarrow W\ (Б, x))$
Кожен щось пише.	$\forall y\ \exists x\ W\ (y, x)$
Кожен, хто пише будь що, пише привітання з Новим роком (Р).	$\forall y\ (\exists x W\ (y, x) \rightarrow W\ (y, P))$
Деякі люди нічого не пишуть.	$\exists y\ \forall x\ \neg W\ (y, x)$

Співвідношення обсягів термінів судження (суб'єкта і предиката) для різних схем можна проілюструвати наступним чином:

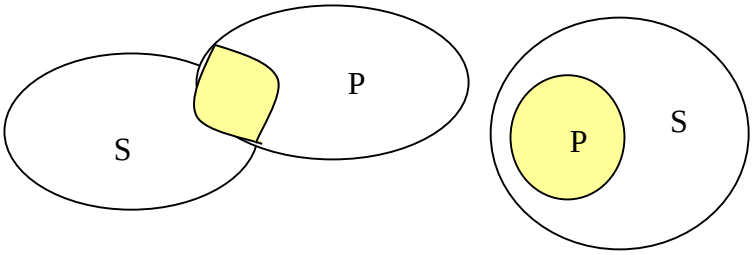
Загально стреджувальне А



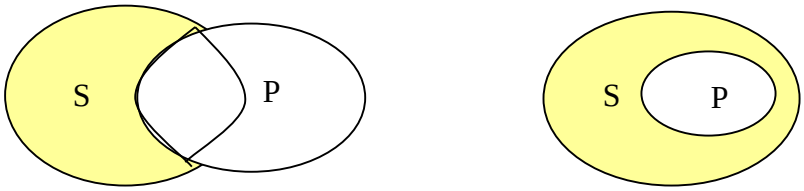
Загально негативне Е



Частково стреджувальне І



Частково негативне О



Ці чотири типи суджень утворюють так званий *логічний квадрат,* який показує зв'язок між схемами суджень (рис. 1.8).

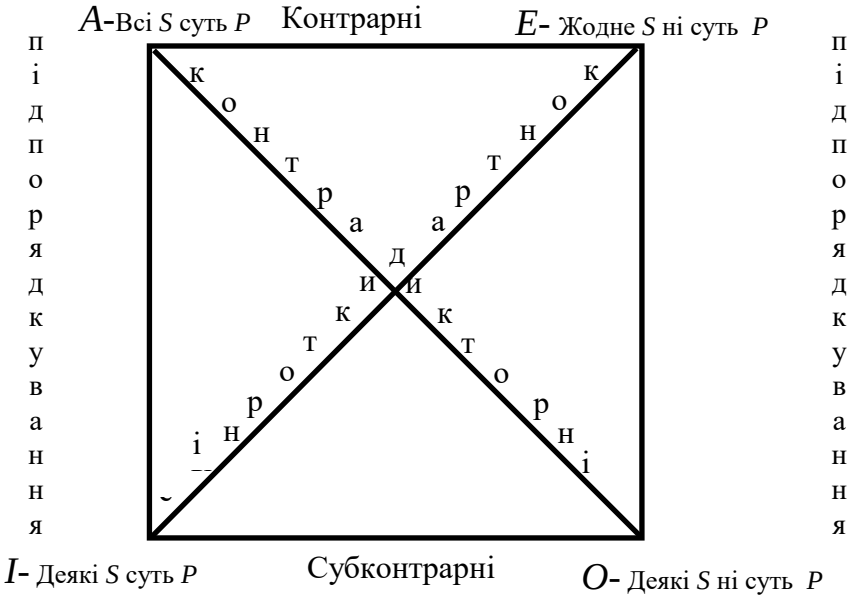


Рис. 1.8. Логічний квадрат

Логічний квадрат[3] - штучна графічна наочна схема для зображення зв'язків і властивостей суджень ***A, E, I, O,*** яка полегшує запам'ятовування характеру відносин між видами суджень. Якщо названі кути квадрата позначити буквами суджень, в яких мова йде про одне й те ж, то лінії, що з'єднують ці точки, позначать певне відношення між відповідними судженнями. Знати відносини між судженнями - значить бути здатним визначити логічне значення (*T* або *F)* одного значення щодо інших.

На малюнку: контрарні - протилежні, контрадикторні - суперечливі, субконтрарність або підконтрарні - нагадують протилежні, але істотно від них відрізняють, підпорядкування - перебувають у відношенні підпорядкування.

Горизонтальні боки квадрата показують відношення контрарности і субконтрарності.

Розуміння контрарних суджень призводить до того, що ***A*** і ***E*** не можуть бути одночасно істинними, але бувають одночасно хибними. З цього, що якщо одне з них істинно, то інше обов'язково хибне. Але якщо одне з них хибне, то зробити висновок про значення іншого ми не можемо, так як воно може бути як *T,* так і *F*.

Твердження *A*: Всі *S* суть *P* : $\forall x\ (S\ (x) \rightarrow P\ (x))$ і *E*: Жодне *S* ні є *P* : $\forall x\ (S\ (x) \rightarrow \neg P\ (x))$. Вони сумісні з хибності, але несумісні по істинності, тобто можуть бути одночасно хибними, але не можуть бути одночасно істинними. Наприклад, «всі романи написані у віршах»: $\forall x\ (N\ (x) \rightarrow C\ (x))$ і «жоден роман не написаний у віршах»: $\forall x\ (N\ (x) \rightarrow \neg C\ (x))$, – контрарні твердження; обидви вони хибні. Твердження: «всі люди смертні» і «всі люди безсмертні», – також контрарні, перше – істинно, друге – хибне.

Субконтрарні твердження сумісні по істинності, але несумісні по хибності, тобто можуть бути одночасно істинними, але не можуть бути одночасно хибними.

Твердження *I*: Деякі *S* суть *P:* $\exists x\ (S\ (x)\ \&\ P\ (x))$ і *O*: Деякі *S* не є *P*: $\exists x\ (S\ (x)\ \&\ \neg P\ (x))$.

Наприклад, «деякі романи – вірші»: $\exists x\ (N\ (x)\ \&\ C\ (x))$ і «деякі романи не вірші»: $\exists x\ (N\ (x)\ \&\ \neg C\ (x))$, – субконтрарність; обидва вони істинні.

Судження, з'єднані діагоналями, називаються *контрадикторними.*

Контрадикторні твердження несумісні по істинності і несумісні з хибності, тобто не можуть бути одночасно істинними, і не можуть бути одночасно хибними. Одне є запереченням іншого:

1) $\neg A = O$, тобто «Не всі *S* суть *P*» ≡ «деякі *S* ні є *P*». Дійсно: $\neg\forall x\ (S\ (x) \rightarrow P\ (x)) \equiv \exists x\ \neg(\neg S\ (x) \vee P\ (x)) \equiv \exists x\ (S\ (x)\ \&\ \neg P\ (x))$. Наприклад, $\forall x\ (N\ (x) \rightarrow C\ (x))$ («всі романи написані у віршах»

[3] Логічний квадрат, на думку К.Прантля, був запропонований в XI столітті візантійським філософом М. Пселлом.

і $\exists x\,(N\,(x)\,\&\,\neg C\,(x))$ («деякі романи – не вірші») – контрадикторні твердження, одне є запереченням іншого.

2) $\neg\, E = I$, тобто «Неправильно, що жодне S не є P » ≡ «деякі S суть P».

$\neg\forall x\,(S\,(x) \to \neg P\,(x)) \equiv \exists x\,\neg(\neg S\,(x) \vee \neg P\,(x)) \equiv \exists x\,(S\,(x)\,\&\,P\,(x))$.

Вертикальні боки квадрату показують відношення логічного слідування (в логічному квадраті – відношення підпорядкування): твердження, що знаходяться знизу, логічно випливають з тих, що знаходяться зверху. Дійсно, якщо «все S суть P», то і «деякі S суть P», тобто виконано логічне слідування: $\forall x\,(S\,(x) \to P\,(x)) \models \exists x\,(S\,(x)\,\&\,P\,(x))$, звідки випливає, що $|\ A \to I\ | \equiv T$. Наприклад, якщо «всі конспекти – підручники», то і «деякі конспекти – підручники». Інше логічне слідування також очевидно: якщо «жодне S не є P», то і «деякі S не є P»: $\forall x\,(S\,(x) \to \neg P\,(x)) \models \exists x\,(S\,(x)\,\&\,\neg P\,(x))$, звідки випливає, що $|\ E \to O\ | \equiv T$. Наприклад, якщо «ні один (жоден) підручник не написаний у віршах», то і «деякі підручники не написані в віршах». Але з хибності ***A*** (або ***E)*** не слідує однозначна відповідь щодо ***I*** (або ***O)***.

Залежність істиннісних значень суджень типу ***A, E, I, O,*** в яких мова йде про одне й те ж саме, можна передати за допомогою таблиці.

A	***E***	***I***	***O***
T	*F*	*T*	*F*
F	*F*	*T*	*T*
F	*T*	*F*	*T*

1.8.3. Доказ логічних слідувань

В даному розділі розглянемо два способи доказів логічних слідувань: неформальний спосіб, заснований на доказі від супротивного (метод редукції), та формальний метод в численні предикатів (формальна теорія ***K***). Надалі буде розглянуто більш ефективний метод, що дозволяє автоматизувати процес доказів логічного слідування (логічної загальнозначущості) – метод резолюції та його реалізація в логічному програмуванні.

Розглянемо декілька схем формалізації задач та перевірки логічного слідування.

Приклад 1.

Γ 1: $\forall$x $(D\,(\text{x}) \to L\,(x))$,

Γ 2: $\forall$x $(Q\,(\text{x}) \to \neg L\,(x))$.

Висновок G : $\forall$x $(D\,(\text{x}) \to \neg Q\,(\text{x}))$.

Це може відповідати такій умові на природній мові: *всі птахи літають, жодна риба не літає, отже, жодна птиця не є рибою.*

На області визначення «*живі істоти*» визначено $D\,(x)$: *x – птах,* $Q\,(x)$: *x – риба,* $L\,(x)$: *x літає.*

За визначенням, | $\Gamma 1$ | = *T*, | $\Gamma 2$ | = *T*. Припустимо, що | *G* | = *F*.

З припущення | $\forall x\ (D(x) \to \neg Q(x))$ | = ***F*** випливає, що ***існує принаймні одне значення x = a таке,*** що | $D(a) \to \neg Q(a)$ | = ***F***, звідки отримуємо, що $|D(a)| = T$, $|\neg Q(a)| = F$, тобто | $Q(a)$ | = *T*.

З посилки $\Gamma 1$: | $\forall x\ (D(x) \to L(x))$ | = *T* випливає, що ***раз посилка істина для всякого x, то в тому числі, для x = a,*** одержуємо | $(D(a) \to L(a))$ | = *T*, тобто $|(T \to L(a))| = T$, звідки | $L(a)$ | = *T*.

Оскільки посилка $\Gamma 2$: | $\forall x\ (Q(x) \to \neg L(x))$ | = *T* для всіх значень *x*, то вона істинна і для *x* = *a* : | $(Q(a) \to \neg L(a))$ | = *T*. Так як | $Q(a)$ | = *T*, то $|\neg L(a)| = T$. Отримуємо, що істинні обидва твердження: | $L(a)$ | = *T* і $|\neg L(a)| = T$. Отримане протиріччя доводить логічне слідування (припущення про хибність висновку було неправильним).

У формальному виведенні застосовуються правила: універсальної конкретизації (УК), правило відділення – modus ponens (МР) тощо.

Побудуємо **формальний висновок.**

1. $\forall x\ (D(x) \to L(x))$ Г 1
2. $\forall x\ (Q(x) \to \neg L(x))$ Г 2
3. $Q(z) \to \neg L(z))$ УК 2 – *заміна на іншу змінну, вільну від x*
4. $D(z) \to L(z)$ УК 1 – *заміна на змінну*
5. $L(z)) \to \neg Q(z)$ правило контрапозиции (ПК, контр.) 3
6. $D(z) \to \neg Q(z)$ правило силогізму (ПС, сил.) 4,5
7. $\forall x\ (D(x) \to \neg Q(x))$ Gen *(узагальнення, Generalization*) 6 – *навішуємо на змінну квантор загальності*

<u>Приклад 2.</u>

$\Gamma 1$: $\forall x\ (D(x) \to L(x))$,

$\Gamma 2$: $D(a)$

Висновок $G : L(a)$.

Це може відповідати умові: *рідкісний птах долетить до середини Дніпра. Пінгвін – птах рідкісний, отже, пінгвін долетить до середини Дніпра.*

На області визначення «*живі істоти*» ми визначимо $D(x)$: *x – птах,* $L(x)$: *x долетить до середини Дніпра, і a= «пінгвін», константа з області визначення.*

За визначенням, | $\Gamma 1$ | = *T*, | $\Gamma 2$ | = *T*. Припустимо, що | *G* | = *F*.

Тоді за припущенням | $L(a)$ | = *F*.

Оскільки посилка $\Gamma 2$: $|D(a)|=T$, то і з посилки $\Gamma 1$: $|\forall x(D(x) \to L(x))| = T$ випливає, що раз посилка істина для всякого x, то в тому числі, для x=*a*, то одержуємо | $(D(a) \to L(a))$ | = *T*, тобто $|(T \to L(a))| = T$, звідки | $L(a)$ | = *T*.

Отримуємо, що істинні обидва твердження: $|L(a)|=T$ і $|\neg L(a)| = T$. Отримане протиріччя доводить логічне слідування.

У формальному виведенні застосовуються вже знайомі логічні правила, а також модифіковане правило МР - ***узагальнене правило відділення*** (узагальнене МР – ***GMP***):

Формальний висновок.

1. $\forall x\,(D(x) \to L(x))$ Г 1
2. D (a) Г 2
3. $D(z) \to L(z)$ УК (1)
4. L (a) GMP (2,3)

*[**GMP**: підставляємо в п.3. замість змінної z константу a з області визначення*

$D(a) \to L(a)$, через те, що п.3. виконується для будь-якого значення. $D(a) \to L(a)$, $D(a)$ (п.2.) |= $L(a)$ за правилом МР]

Можна ввести за аналогією ще одне правило на базі ***GMP – узагальнене правило modus tollens – GMT.***

<u>Приклад 3.</u>

Вивчення правил виведення з точки зору психології привели до відкриття феномену, що назвали «ефектом атмосфери» (середина 30-х років минулого століття, Р.Вудвортс), згідно з яким «атмосфера», створювана посилками, налаштовує випробуваного на прийняття одних висновків і відкиданні інших. Цей феномен можна звести до двох принципів. По-перше, якщо, принаймні одна посилка негативна, то і висновок буде сформульований, скоріше, в негативній формі, хоча схильність до позитивних відповідей - характерна риса пізнавальних процесів у людини. По-друге, якщо, принаймні одна посилка є частковою (тобто містить квантор «деякі»), то і висновок буде швидше, частковим. В іншому випадку він буде сформульований в універсальній формі, для якої характерно використання кванторів «все» або «ні один».

Одним з яскравих прикладів є наступний силогізм:

1. *«Деякі бджолярі - художники»*
2. *«Жоден хімік не є бджолярем».*

Як показали експерименти з 20 випробовуваних, 12 заявили, що з цих посилок не можна зробити однозначного висновку. Лише двоє змогли дати правильну відповідь: *<u>«Деякі художники не є хіміками»</u>*.

Введемо наступні предикати на області визначення *x={люди}: H (x): x – бджоляр, A* (*x*): *x – художник, C* (*x*): *x – хімік.*

Формалізуємо речення:

1. $\exists x\,(H(x) \,\&\, A(x))$
2. $\forall x\,(C(x) \to \neg H(x))$ // формалізований запис посилки отримано за рахунок перетворювань: $\neg\exists x\,(C(x) \,\&\, H(x)) = \forall x \neg\,(C(x) \,\&\, H(x)) = \forall x\,(\neg C(x) \vee \neg H(x)) = \forall x\,(C(x) \to \neg H(x))$

Побудуємо **формальний висновок:**

1. $\exists x\,(H(x) \,\&\, A(x))$ Г1
2. $\forall x\,(C(x) \rightarrow \neg H(x))$ Г2
3. $H(a) \,\&\, A(a)$ екзистенціальна конкретизація (ЕК) 1
4. $H(a)$ видалення & 3
5. $A(a)$ видалення & 3
6. $C(y) \rightarrow \neg H(y)$ УК 2
7. $H(y) \rightarrow \neg C(y)$ ПК 6
8. $\neg C(a)$ GMP 4,7
9. $A(a) \,\&\, \neg C(a)$ введення & 5,8

Можемо отримати висновок: «деякі художники не є хіміками».

10. $\exists x(A(x) \,\&\, \neg C(x))$ екзистенціальне узагальнення (ЕУ) 9

Тепер перевіримо логічне слідування неформальним методом – від супротивного.

Припустимо, що формула, яка відповідає умові, не є ЛЗЗ:

$|(\forall x\,(C\,(x) \rightarrow \neg H\,(x))) \,\&\, (\exists x\,(H\,(x) \,\&A\,(x))) \rightarrow \exists x\,(A\,(x) \,\&\, \neg C\,(x))\,|= F.$

Тоді

Г1: $|\,\forall x\,(C\,(x) \rightarrow \neg H\,(x))\,| = T,$

Г2: $|\,\exists x\,(H\,(x) \,\&\, A\,(x))\,| = T,$

G: $|\,\exists x\,(A\,(x) \,\&\, \neg C\,(x))\,| = F.$

Тоді для Г2 існує x = a такий, що

$|H\,(a) \,\&A\,(a)\,|= T; \Rightarrow |H\,(a)\,|= T$ і $|A\,(a)\,|= T.$

Далі з Г1, раз для всіх x виконується, то і для x = a повинно виконуватися $|C\,(a) \rightarrow \neg H\,(a)\,|= T$. Через те, що вище означено, що $|H\,(a)\,|= T$ слідує, що $|\,C\,(a)\,| = F$. Тоді для x = a буде виконуватися $|A\,(a) \,\&\neg C\,(a)\,|= T$, тобто висновок $|\exists x\,(A\,(x) \,\&\neg C\,(x))\,|= T$, а не F, як було припущено. Значить, логічне слідування виконується.

Вочевидь, що при проведенні формального висновку до п.9. застосувати правило Gen не можна, тобто висновок $\forall x\,(A\,(x) \rightarrow \neg C\,(x))$ логічно не слідує з гіпотез умови.

Покажемо, що метод редукції дає такий же негативний результат щодо висновку «для всіх» при наявності в умові «частковості».

Припустимо, що

Г1: $|\,\forall x\,(C\,(x) \rightarrow \neg H\,(x))\,| = T,$

Г2: $|\,\exists x\,(H\,(x) \,\&\, A\,(x))\,| = T,$

G: $/\,\forall x\,(A\,(x) \rightarrow \neg C\,(x))\,/= F.$

З висновку знайдеться хоча б одне таке x=a, що $/(A\,(a) \rightarrow \neg C\,(a))\,/= F$. Звідси слідує, що $/(A\,(a)\,/= T$ і $/C\,(a)\,/=T$. З Г1 слідує, що раз для всіх x виконується, то і для x=a теж, і тоді $/\,(C\,(a) \rightarrow \neg H\,(a)\,/= T$ і з вище означеного $/H\,(a)\,/= F$. Але це ще не приводить к протиріччю в Г2. Так, для

x=a |H (a) & A (a) |= F. Але може існувати такий x=b, що буде виконуватися |H (b) & A (b) |=T, і |∃x (H (x) & A (x)) |= T.

Приклад 4.

1. Всі леви – тварини дикі.
2. Я знаю, що деякі з них не п'ють кави.

Висновок: а) Жоден з диких левів не п'є кави.

б) Будь-який лев – дикий і не п'є кави.

Область визначення – живі істоти. Предикати L(x) – x – лев, W(x) – x – дика тварина, C(x) – x не п'є кави.

1. ∀x (L (x) → W (x))
2. ∃x (L (x) & C (x))

Висновок:

а) ∀x (L (x) & W (x) → C (x))

б) ∀x (L (x) → W (x) & C (x))

Якщо побудувати **формальний висновок:**

1. ∀x (L(x) → W(x))	Г1
2. ∃x (L(x) & C(x))	Г2
3. L(a) & C(a)	ЕК 2
4. L(a)	видалення & 3
5. C(a)	видалення & 3
6. L(y) → W(y)	УК 1
7. W (a)	GMP 4,6

Ми можемо побачити, що правило Gen не може бути застосовано до жодного з отриманих пунктів. Тому ні висновок а), ні б) – логічно не правомірний. Логічно виконується з множини посилок наступне:

8. W(a) & C(a)	введення & 5,7
9. ∃x(W(x) & C(x))	ЕУ 9 – існують дикі тварини, що не п'ють кави

Приклад 5.

Будь-які фрукти – корисні. Деякі фрукти – стиглі. Висновок ?: стиглі фрукти – корисні.

Область визначення – фрукти/ягоди – рослинна їжа. Предикати: F(x) – x - фрукт, U(x) – x – корисний, R(x) – x – стиглий.

1. ∀x (F (x) → U (x)),
2. ∃x (F (x) & R (x)),

G: ∀x (F (x) & R (x) → U (x)).

Якщо припустити, що висновок – хибний, тобто знайдеться такий x=a, що | F (a) & R (a) → U (a) | = F, то | F (a) | = T, | R (a) | = T та | U (a) | = F. Тоді, з Г1, раз виконується для всіх (|Г1|=T, |Г2|=T за умовою), то і для x=a теж, | F (a) → U (a) | повинно дорівнюватися T, але виходить F. Отримали протиріччя, висновок – правильний, Г2 – зайва; приклад демонструє існування

загального висновку при частковій посилці, але вона не приймає участь в логічному слідуванні.

Формальний висновок:

1. $\forall$x (F (x) $\rightarrow$ U (x)) Г1
2. $\exists$x (F (x) & R (x)) Г2
3. F (y) $\rightarrow$ U (y) УК 1
4. *F (a) & R (a)* *ЕК 2*
5. F (y) & R (y) $\rightarrow$ F (y) A1

[A1: A→(B →A) = A&B →A]

6. F (y) & R (y) $\rightarrow$ U (y) сил. 3,5
7. $\forall$x (F (x) & R (x) $\rightarrow$ U (x)) Gen 6

Приклад 6.

Деякі режисери знімають серіали. Ніхто з них не знімає мультфільми, отже, жоден серіал не є мультфільмом.

Предикати задані на областях X = {люди} і Y = {фільми}. Нехай P (x): x - режисер, D (y): y - серіал, Q (y): y -мультфільми, L (x, y) - x знімає y. Формалізуємо посилки:

$\Gamma 1$: $\exists x(P(x)\&\forall y(D(y) \rightarrow L(x,y)))$

$\Gamma 2$: $\forall x(P(x) \rightarrow \forall y(Q(y) \rightarrow \neg L(x,y)))$

G: $\forall y(D(y) \rightarrow \neg Q(y))$

Припустимо, що при істинності посилок | Г1 | = T, | Г2 | = T висновок приймає хибне значення: | G | = F.

Із $|\forall y(D(y) \rightarrow \neg Q(y))| = F$ слідує, що існує таке y = a, що $|D(a) \rightarrow \neg Q(a)| = F$. Тоді, $|D(a)| = T$, $|\neg Q(a)| = F$, *тобто* $Q(a)| = T$.

З $|\exists x(P(x)\&(D(a) \rightarrow L(x,a)))| = T$ слідує, що існує таке x = b, що $|P(b) \& (D(a) \rightarrow L(b,a))| = T$. Тоді $|P(b)| = T$ і $|D(a) \rightarrow L(b,a)| = T$, а через те, що $|D(a)| = T$, то $|L(b,a)| = T$.

Посилка Г2: для всіх x $|\forall x(P(x) \rightarrow (Q(a) \rightarrow \neg L(x,a)))| = T$, в тому числі для x = b, отже, $|P(b) \rightarrow (Q(a) \rightarrow \neg L(b,a))| = T$, а через те, що $|P(b)| = T$, $|Q(b)| = T$, то з $|Q(a) \rightarrow \neg L(b,a)| = T$ следует $|\neg L(b,a)| = T$.

Таким чином, отримуємо, що істинні обидва твердження: $|T \rightarrow L(b,a)| = T$ і $|T \rightarrow \neg L(b,a)| = T$, тобто $|L(b,a)| = T$ і $|\neg L(b,a)| = T$. Отримане протиріччя доводить логічне слідування.

Формальний висновок:

1. $\exists x(P(x)\&\forall y(D(y) \rightarrow L(x,y)))$ Г1
2. $\forall x(P(x) \rightarrow \forall y(Q(y) \rightarrow \neg L(x,y)))$ Г2

3. $P(b)\&\forall y(D(y) \to L(b,y))$	ЕК(1)
4. $P(b)$	видалення &(3)
5. $\forall y(D(y) \to L(b,y)))$	видалення &(3)
6. $P(r) \to \forall y(Q(y) \to \neg L(r,y))$	УК(2)
7. $\forall y(Q(y) \to \neg L(b,y))$	GMP(4, 6)
8. $Q(z) \to \neg L(b,z)$	УК(7)
9. $D(z) \to L(b,z)$	УК(5)
10. $L(b,z) \to \neg Q(z)$	контрапозиція(8)
11. $D(z) \to \neg Q(z)$	силогізм (9,10)
12. $\forall y(D(y) \to \neg Q(y))$	*Gen* (11)

Питання до розділу 1

1. Висловлювання. Поняття одномісного предиката. Приклади. Визначення предметної області, терма, формули.
2. Визначення багатомісного предиката. Приклади. Визначення предметної області, терма, формули.
3. Операції над предикатами. Навішування кванторів. Область дії квантора. Вільні і зв'язані змінні. У заданій формулі визначити, які змінні є зв'язаними, а які – вільними.
4. Інтерпретація і здійсненність формул алгебри предикатів. Таблиці істинності формул логіки предикатів. Здійснимі формули, протиріччя, логічно-загальнозначущі формули. Часткова інтерпретація формул на множині D = {1,2}.
5. Визначення логічно-загальнозначущих формул логіки предикатів. Записати правила Де Моргана для формул логіки предикатів. Записати ЛОЗ формули для пронесення зв'язок &, $\vee$. Рівносильність формул логіки предикатів.
6. Логічне слідування в логіці предикатів. Правила універсальної і екзистенціальної конкретизації. Доведення. Правило узагальнення Gen. Доведення.
7. Узагальнене правило відділення (GMP). Навести приклади.
8. Визначення формальної теорії K. Метатеореми про дедукції в зчисленні предикатів.
9. Формалізація речень природної мови. Навести приклади. Основні схеми суджень. Логічний квадрат. Сумісність по істинності і хибності схем суджень, що складають логічний квадрат.

2. АВТОМАТИЧНЕ ДОВЕДЕННЯ ТЕОРЕМ

2.1. Основа методу резолюції

Пошук загальної процедури для перевірки загальнозаначущості формул розпочато давно. Першим намагався знайти таку процедуру Лейбніц (1646-1716), в подальшому над цим працювала школа Гільберта. Це тривало до тих пір, поки Черч і Тьюринг незалежно один від одного не довели, що не існує ніякої загальної розв'язної процедури, ніякого алгоритму, що перевіряє загальнозначущість формул в логіці предикатів першого порядку. Проте, існують алгоритми *пошуку* доказів, які можуть підтвердити, що формула загальнозначуща. Для незагальнозначущих формул ці алгоритми, взагалі кажучи, не закінчують свою роботу. Беручи до уваги результат Черча і Тьюринга, це найкраще, що можна очікувати від алгоритму пошуку доказів.

У 1930 році важливий підхід до автоматичного доведення теорем був дан Ербраном. За визначенням загальнозначуща формула є формула, яка істинна при всіх інтерпретаціях. Ербран розробив алгоритм знаходження інтерпретації, яка спростовує дану формулу. Однак, якщо дана формула насправді загальнозначуща, то такій інтерпретації не існує, і алгоритм закінчує роботу за кінцеве число кроків. Метод Ербрана служить основою для більшості сучасних автоматичних алгоритмів пошуку доказів. Головний результат був отриманий Робінсоном, який ввів так званий *метод резолюцій*.

В основі методу резолюцій лежить процедура пошуку спростування, тобто замість доказу загальнозначущості формули доводиться, що заперечення формули суперечливо. Метод спростування для доказу логічного слідування полягає в наступному.

Нехай виконується логічне слідування:

***F*1, *F*2 |= *G*. Тогда |= *F*1&*F*2 → *G* логічно загальнозначуща, і, отже,**

|¬(*F*1&*F*2 → *G*)| = |*F*1&*F*2&¬*G*| ≡ *F*.

Оскільки за визначенням посилки *F* 1, *F* 2 правдиві, формула *F* 1 & *F* 2 & ¬*G* може звернутися в хибну тільки, якщо | ¬*G* | = *F*, тобто якщо | *G* | = *T*. Тоді логічний висновок виконано.

В принципі процедура спростування формалізує метод редукції.

Проблема пошуку автоматичного доведення при використанні процедури спростування значно полегшується завдяки використанню так званих «стандартних» форм формул. Будь-яку формулу логіки предикатів можна привести до еквівалентної їй ***попередженої нормальній формі, коли всі квантори винесені вперед, а в області дії кванторів формула знаходиться в кон'юнктивній нормальній формі.***

Якщо така формула має тільки квантори загальності, то вона буде хибною, якщо знайдеться хоча б одна інтерпретація, яка надає їй хибне

значення. Тоді еквівалентна їй вихідна формула також буде хибною, а її заперечення, відповідно, істинно.

Якщо ж серед кванторів є квантори існування, то проблема ускладнюється. Однак, квантори існування можна зняти (на підставі правила екзистенціальної конкретизації), в результаті буде отримана так звана «скулемовская стандартна форма». Тоді пошук спростовуючої інтерпретації застосовується до цієї форми.

2.2. Попереджені нормальні форми

Визначення 2.1. Формула A знаходиться в *попередженій нормальній формі* (ПНФ), якщо вона має вигляд:

$(Q_1 x_1)\ldots (Q_n x_n)\ M$, де кожне $Q_i x_i$ є $\exists x_i$ чи $\forall x_i$,

а M – формула в кон'юнктивній нормальній формі, яка не містить кванторів.

$(Q_1 x_1)\ldots (Q_n x_n)$ називається *префіксом,* а *M – матрицею* формули A.

Теорема 2.1. Існує ефективна процедура приведення будь-якої формули логіки предикатів до еквівалентної їй попередженої нормальної форми.

Доведення теореми конструктивне, тобто дає алгоритм перетворення будь-якої формули до попередженої нормальної форми. Теорема доводиться індукцією по числу зв'язок m.

1. Нехай $m = 0$. Тоді формула A не містить зв'язок і знаходиться в ПНФ.

2. Припустимо, що існує ПНФ для формули B з числом зв'язок n. Доведемо, що існує ПНФ для формули A з числом зв'язок $m = n + 1$.

1 *випадок*. Нехай існує ПНФ для $B = (Q_1 x_1)\ldots (Q_n x_n)\ M$. Формула A утворена з B за допомогою операції заперечення:

$A = \neg(Q_1 x_1)\ldots (Q_n x_n)\ M$.

За законами де Моргана зв'язка $\neg$ проноситься через квантори: $\neg\forall x M \equiv \exists x \neg M$, $\neg\exists x M \equiv \forall x \neg M$. Отримана формула буде перебувати в ПНФ.

2 *випадок*. Формула A утворена з двох формул $B1$ і $B2$ з числом зв'язок $n < m$, за допомогою зв'язок кон'юнкції & або диз'юнкції $\vee$:

$(Q_1x_1)\ldots(Q_nx_n)M1\&(Q_1y_1)\ldots(Q_ny_n)M2$ або $(Q_1x_1)\ldots (Q_nx_n)\ M1 \vee (Q_1y_1)\ldots (Q_ny_n)M2$.

Тоді, якщо формули $B1$ і $B2$ мають квантори по одній і тій же змінної, використовуємо закони заміни пов'язаних змінних:

$$\forall x P(x) \equiv \forall y P(y), \qquad \exists x P(x) \equiv \exists y P(y),$$

так, щоб жодна вільна змінна не стала пов'язаною в результаті цієї заміни. Після цього скористаємося законами коммутативности для & і $\vee$ і законами пронесення кванторів:

$$\forall x(P(x)\&B) \equiv \forall x P(x)\&B, \quad \forall x(P(x) \vee B) \equiv \forall x P(x) \vee B,$$
$$\exists x(P(x)\&B) \equiv \exists x P(x)\&B, \quad \exists x(P(x) \vee B) \equiv \exists x P(x) \vee B,$$

(B не містить входжень x).

3 *випадок.* Формула A утворена з B навішуванням квантора $\forall$ або $\exists$. Тоді, оскільки B знаходиться в ПНФ, знову отримана формула буде в ПНФ.

Приклади.

1. $\exists x P(x) \to \forall x(\exists y D(y) \& L(x, y)) = \neg\exists x P(x) \vee \forall x(\exists y(D(y) \& L(x, y))) =$
$= \forall x \neg P(x) \vee \forall x(\exists y(D(y) \& L(x, y))) = \forall x \neg P(x) \vee \forall z(\exists y(D(y) \& L(z, y))) =$
$= \forall x(\neg P(x) \vee \forall z(\exists y(D(y) \& L(z, y)))) = \forall x(\forall z(\exists y(D(y) \& L(z, y))) \vee \neg P(x)) =$
$= \forall x \forall z(\exists y(D(y) \& L(z, y))) \vee \neg P(x))) = \forall x \forall z \exists y((D(y) \& L(z, y)) \vee \neg P(x))) =$
$= \forall x \forall z \exists y((D(y) \vee \neg P(x)) \& (L(z, y)) \vee \neg P(x))).$

2. $\exists x(P(x) \& \forall y(D(y) \to L(x, y))) = \exists x(P(x) \& \forall y(\neg D(y) \vee L(x, y))) =$
$= \exists x(\forall y(\neg D(y) \vee L(x, y)) \& P(x)) = \exists x \forall y((\neg D(y) \vee L(x, y)) \& P(x)).$

3. $\forall x(P(x) \to \forall y(Q(y) \to \neg L(x, y))) = \forall x(\neg P(x) \vee \forall y(\neg Q(y) \vee \neg L(x, y))) =$
$= \forall x(\forall y(\neg Q(y) \vee \neg L(x, y)) \vee \neg P(x)) = \forall x \forall y(\neg Q(y) \vee \neg L(x, y) \vee \neg P(x)).$

2.3. Скулемівські стандартні форми

Визначення 2.2. Попереджена нормальна форма, яка містить тільки квантори загальності, називається *скулемівською стандартною формою* (*ССФ)*.

Процедура приведення ПНФ до скулемівської форми полягає в елімінації (видаленні) кванторів існування.

Нехай формула A знаходиться в попередженій нормальній формі $(Q_1 x_1) \ldots (Q_n x_n) M$, де M є *кон'юнктивна нормальна форма*. Якщо квантор існування - перший зліва квантор в префіксі: $(\exists x_1)(Q_2 x_2) \ldots (\ldots (Q_n x_n) M$, то його можна елімінувати на підставі правила екзистенціальної конкретизації. Виберемо константу c, відмінну від інших констант, що входять в M, замінимо всі входження x_1, що зустрічаються в M, на c, і викреслимо квантор $\exists x_1$ з префікса.

Якщо ж перед квантором існування знаходиться квантор загальності, наприклад, $\forall x \exists y M$, то змінна y перебуває в радіусі дії квантора загальності, і вираз $\forall x \exists y$ (для кожного x існує y) означає наявність деякої функціональної залежності $y=f(x)$. Якщо квантору існування передує кілька кванторів загальності, то функція залежить від всіх змінних, за якими навішені ці квантори. У загальному випадку, якщо $Qs_1, \ldots, Qs_m$ – список всіх кванторів загальності, що зустрічаються лівіше $\exists x_r$, $1 \le s_1 < s_2 < \ldots < s_m < r$, ми виберемо m місний функціональний символ f, відмінний від інших функціональних символів, замінимо всі x_r в M на $f(x_{s1}, \ldots, x_{sm})$ і викреслимо $\exists x_r$ з префікса. Потім весь цей процес можна використовувати для всіх кванторів існування в префіксі; остання з отриманих формул є *скулемівська стандартна форма* – для стислості, *стандартна форма* (ССФ) формули A. Функції, які використовуються для заміни змінних квантора

існування, називаються *скулемівськими* функціями (константи є нульмісними функціями).

Приклад. Отримаємо стандартну форму формули:

$A = \exists x \forall y \forall z \exists u \forall v \exists w P(x, y, z, u, v, w)$. У цій формулі лівіше $\exists x$ немає ніяких кванторів загальності, лівіше $\exists u$ стоять $\forall y$ і $\forall z$, а лівіше $\exists w$ стоять $\forall y$, $\forall z$ і $\forall v$. Отже, ми замінимо змінну x на константу a, змінну u - на двомісну функцію $f(y, z)$, змінну w - на тримісну функцію $g(y, z, v)$.

Таким чином, ми отримуємо наступну стандартну форму формули A:

$S = \forall y \forall z \forall v P(a, y, z, f(y, z), v, g(y, z, v))$.

Для розглянутих вище посилок в прикладі на ПНФ ССФ має вигляд:

1. $\forall x \forall z \exists \mathbf{y}((D(\mathbf{y}) \vee \neg P(x)) \& (L(z, \mathbf{y})) \vee \neg P(x))) \Rightarrow$
   $\forall x \forall z((D(\boldsymbol{f(x,z)}) \vee \neg P(x)) \& (L(z, \boldsymbol{f(x,z)})) \vee \neg P(x)))$
2. $\exists \mathbf{x} \forall y((\neg D(y) \vee L(\mathbf{x}, y)) \& P(\mathbf{x})) \Rightarrow \forall y((\neg D(y) \vee L(\boldsymbol{a}, y)) \& P(\boldsymbol{a}))$,
3. $\forall x \forall y(\neg Q(y) \vee \neg L(x, y) \vee \neg P(x))$

Якщо попереджена нормальна форма еквівалентна вихідній формулі, то скулемовская стандартна форма формули A, взагалі кажучи, не є еквівалентною їй.

Наприклад, нехай $A = \exists x P(x)$ і $S = P(a)$ є стандартна форма формули A. Нехай I є наступна інтерпретація: область $D = \{a, b\}$, $P(a) = F$, $P(b) = T$. Тоді A істинна в I, але S хибна в I. Таким чином, A не є еквівалентною S. Однак, якщо $P(a) = F$, $P(b) = F$, то $| A | = F$, і $S = P(a)$ також приймає значення F для будь-якого a. Таким чином, $A \equiv S$ в тому і тільки тому випадку, якщо A суперечлива.

Теорема 2.2. Нехай S – стандартна форма формули A. Тоді A суперечлива в тому і тільки тому випадку, коли S суперечлива.

Доведення. Нехай формула A знаходиться в ПНФ, тобто $A = (Q_1 x_1) \ldots\ldots (Q_n x_n) M[x_1, \ldots, x_n]$. (Запис $M[x_1, \ldots, x_n]$ означає, що матриця M містить змінні $x_1, \ldots, x_n$). Нехай Q_r – перший зліва квантор існування. Нехай $A_1 = (\forall x_1) \ldots (\forall x_{r-1}) (Q_{r+1} x_{r+1}) \ldots (Q_n x_n) M[x_1, \ldots, x_{r-1}, f(x_1, \ldots, x_{r-1}), x_{r+1}, \ldots, x_n]$, де f – скулемівська функція, відповідна x_r, $1 \le r \le n$. Ми хочемо показати, що A суперечлива тоді і тільки тоді, коли A_1 суперечлива.

Припустимо, що A суперечлива. Якщо A_1 несуперечлива, то існує така інтерпретація I, що A_1 істинна в I, тобто для всіх $x_1, \ldots, x_{r-1}$ існує принаймні один елемент $f(x_1, \ldots, x_{r-1})$, для якого $(Q_{r+1} x_{r+1}) \ldots (Q_n x_n) M[x_1, \ldots, x_{r-1}, f(x_1, \ldots, x_{r-1}), x_{r+1}, \ldots, x_n]$ істинна в I. Таким чином, A істинна в I, що суперечить припущенню, що A суперечлива. Отже, A_1 повинна бути суперечлива.

З іншого боку, припустимо, що A_1 суперечлива. Якщо A несуперечлива, то існує така інтерпретація I на області D, що A істинна в I, тобто для

всіх $x_1,..., x_{r-1}$ існує такий елемент x_r, що $(Q_{r+1} x_{r+1})... (Q_n x_n)$ $M[x_1,..., x_{r-1}, f(x_1,..., x_{r-1}), x_{r+1},..., x_n]$ істинна в I. Розширимо інтерпретацію I, включивши в неї функцію $f(x_1,..., x_{r-1}) = x_r$, яка відображає $(x_1,..., x_{r-1})$ на x_r для всіх $x_1,..., x_{r-1}$ в D, Позначимо це розширення I'. Тоді для всіх $x_1,..., x_{r-1}$ $(Q_{r+1} x_{r+1})... (Q_n x_n)$ $M[x_1,..., x_{r-1}, f(x_1,..., x_{r-1}), x_{r+1},..., x_n]$ істинна в I', тобто A_1 істинна в I', що суперечить припущенню, що A_1 суперечлива. Отже, A повинна бути суперечливою.

Припустимо тепер, що в A є m кванторів існування. Нехай $A_0 = A$. Нехай A_k виходить з A_{k-1} заміною першого квантора існування в A_{k-1} скулемівською функцією, $k = 1,..., m$. Тоді стандартна форма $S = A_m$. Використовуючи ті ж міркування, що були дані вище, ми можемо показати, що A_{k-1} суперечлива тоді і тільки тоді, коли A_k суперечлива при $k = 1,..., m$. Отже, A суперечлива тоді і тільки тоді, коли S суперечлива, що й треба було довести. ◊

Визначення 2.3. Диз'юнкція літер називається *диз'юнкт,* або *клоуз* (*clause)*. Однолітерний диз'юнкт називається *одиничним диз'юнктом*. **Коли диз'юнкт не містить ніяких літер, його називають *порожнім (пустим)* диз'юнктом. Так як порожній диз'юнкт не містить літер, які могли б бути істинними за будь-яких інтерпретаціях, то порожній диз'юнкт завжди хибний.** Порожній диз'юнкт позначається символом □.

Нехай S - стандартна форма формули A. Матриця формули, представленої в ССФ, знаходиться в кон'юнктивній нормальній формі, тобто у вигляді кон'юнкції диз'юнктів. Будемо представляти ССФ формули A множиною диз'юнктів, де кожна змінна вважається керованою квантором загальності. Множина диз'юнктів - це просто інша форма представлення стандартної форми формули A, тому в подальшому будемо позначати його так само, як і ССФ - символом S. Вважаємо, що множина диз'юнктів S є кон'юнкція всіх диз'юнктів з S.

Наприклад, ССФ прикладу 2: $\forall y ((\neg D(y) \vee L(a, y)) \& P(a))$ може бути представлена множиною диз'юнктів: $S = \{\neg D(y) \vee L(a, y), P(a)\}$.

Далі, якщо ми маємо $A = A_1 \&... \& A_n$, ми можемо окремо отримати множину диз'юнктів S_i, $i = 1,..., n$, де кожне S_i представляє стандартну форму A_i. Тоді A суперечлива тоді і тільки тоді, коли S суперечлива. Кажуть, що множина диз'юнктів *неможлива,* якщо відповідна стандартна форма суперечлива, і *здійсненна* в іншому випадку.

2.4. Ербранівскій універсум множини диз'юнктів

За визначенням, множина диз'юнктів нездійсненна тоді і тільки тоді, коли вона хибна при всіх інтерпретаціях на всіх областях. Так як неможливо

розглядати всі інтерпретації на всіх областях, було б зручно, якби ми могли фіксувати одну таку спеціальну область H, що S була би нездійсненна тоді і тільки тоді, коли S хибна при всіх інтерпретаціях на цій області. Ербран показав, що така область існує. Її називають *ербранівським універсумом* множини S і визначають наступним чином.

Визначення 2.4. Нехай H_0 - множина констант, що зустрічаються в S. Якщо ніяка константа не зустрічається в S, то H_0 складається з однієї довільної константи, наприклад, $H_0 = \{a\}$. Для $i = 1, 2,...$ нехай H_{i+1} є об'єднання H_i і множини всіх термів виду $f^n(t_1,..., t_n)$ (при всіх n) для всіх функцій f^n, що зустрічаються в S, де t_j ($j = 1,..., n$) належить H_j. Тоді кожне H_i називається *множиною констант* i-го рівня для S, і H_∞ називається *ербранівським універсумом S*.

Приклади.

1. Нехай $S_1 = \{P(a), \neg P(x) \vee P(f(x))\}$, тоді

$$H_0 = \{a\};$$
$$H_1 = \{a, f(a)\};$$
$$H_2 = \{a, f(a), f(f(a))\};$$
$$.....$$
$$H_\infty = \{a, f(a), f(f(a)), f(f(f(a))),...\}.$$

2. Нехай $S_2 = \{P(x) \vee Q(x), \neg R(z), R(y) \vee \neg Q(y)\}$. Так як не існує ніяких констант в S, візьмемо $H_0=\{a\}$. Оскільки не існує ніяких функціональних символів в S, то $H = H_0 = H_1 =... = \{a\}$.

Визначення 2.5. Нехай S є множина диз'юнктів. Тоді множина основних атомів виду $P^n(t_1,..., t_n)$ для всіх n -місних предикатів P^n, що зустрічаються в S, де $t_1,..., t_n$ – елементи ербранівского універсуму S, називається *множиною атомів множини S*, або *ербранівським базисом S*.

Приклад. Ербранівський базис множини диз'юнктів $S_1 = \{P(a), \neg P(x) \vee P(f(x))\}$:

$A = \{P(a), P(f(a)), P(f(f(a)), P(f(f(f(a))),...\}$.

Ербранівскій базис множини диз'юнктів $S_2 = \{P(x) \vee Q(x), R(z) \vee \neg Q(x)\}$:

$A = \{P(a), Q(a), R(a)\}$.

Визначення 2.6. *Основний приклад* диз'юнктів C множини диз'юнктів S є диз'юнкт, отриманий заміною змінних в C на члени ербранівського універсуму S.

Приклад. Нехай $S=\{P(x), Q(f(y))\vee R(y)\}$, $C=P(x)$ – диз'юнкт в S і $H = \{a, f(a), f(f(a)),...\}$ –ербранівський універсум S. Тоді $P(a), P(f(a)), P(f(f(a)))$ є основні приклади C.

Визначення 2.7. Нехай S – множина диз'юнктів, H – ербранівській універсум S і I – інтерпретація S над H. Кажуть,

що I є H - *інтерпретація множини* S, якщо вона задовільнюється таким умовам:

1. I відображає всі константи з S в самих себе;

2. нехай f є n -місний функціональний символ і $h_1,..., h_n$ – елементи H.

В I через f позначається функція, яка відображає $(h_1,..., h_n)$ (елемент з H^n) в $f(h_1,..., h_n)$ (елемент з H).

При цьому не виникає ніяких обмежень при наданні значення будь-якого n -місного предикатного символу в S. Нехай $A = \{A_1, A_2,..., A_n,...\}$ – ербранівській базис множини S.

H-інтерпретацію I зручно представляти у вигляді: $I = \{m_1, m_2,..., m_n,...\}$, де m_j є A_j або $\neg A_j$ для $j = 1, 2,...$ Сенс цієї множини в тому, що якщо m_j є A_j, то атому A_j присвоєно значення «істинно», в іншому випадку – значення «хибно».

Приклад. Розглянемо множину $S = \{P(x) \vee Q(x), R(f(y))\}$.

Ербранівській універсум H для S є $H = \{a, f(a), f(f(a)),...\}$.

В S входять три предикатних символи: P, Q і R. Отже, ербранівській базис S є

$A = \{P(a), Q(a), R(a), P(f(a)), Q(f(a)), R(f(a)),...\}$.

Деякі H-інтерпретації множини S:

$I_1 = \{P(a), Q(a), R(a), P(f(a)), Q(f(a)), R(f(a)),...\}$,

$I_2 = \{\neg P(a), \neg Q(a), \neg R(a), \neg P(f(a)), \neg Q(f(a)), \neg R(f(a)),...\}$,

$I_3 = \{P(a), Q(a), \neg R(a), P(f(a)), Q(f(a)), \neg R(f(a)),...\}$, і т. д.

Для будь-якої інтерпретації знайдеться відповідна їй H -інтерпретація.

Теорема 2.3. Множина диз'юнктів S нездійсненна тоді і тільки тоді, коли S хибна при всіх H-інтерпретаціях в S.

Доведення. Перша половина теореми очевидна, так як за визначенням S нездійсненна тоді і тільки тоді, коли S хибна при всіх інтерпретаціях на цій області. Щоб довести другу половину теореми, припустимо, що S хибна при всіх H -інтерпретаціях в S. Покладемо, що S здійснимо. Тоді існує така інтерпретація I на деякій області D, що S істинно при I. Нехай I^* є H-інтерпретація, відповідна I. Тоді S істинно при I^*. Це суперечить припущенням, що S хибна при всіх H-інтерпретаціях в S. Отже, S має бути нездійсненною, що й треба було довести. ◊

Таким чином, ми досягли мети, встановленої на початку цього параграфа, тобто нам необхідно розглядати тільки інтерпретації над ербранівським універсумом, точніше, H-інтерпретації, для перевірки того, здійснимі множини диз'юнктів чи ні. Тому надалі, згадуючи інтерпретацію, ми будемо мати на увазі H-інтерпретацію.

Визначення 2.8. Якщо A – атом, то кажуть, що дві літери A і $\neg A$ *контрарні* один одному, і множину $\{A, \neg A\}$ називають *контрарною парою.*

Відзначимо, що диз'юнкт є тавтологія, якщо він містить контрарну пару, так як $A \vee \neg A \equiv T$, і множина диз'юнктів нездійсненна, якщо вона містить два одиничних контрарних диз'юнкти, так як $A \,\&\, \neg A \equiv F$.

Приклади.

1. Нехай $S = \{P, Q \vee R, \neg P \vee \neg Q, \neg P \vee \neg R\}$.

Ербранівський базис множини S є $A = \{P, Q, R\}$, множина спростовуючих прикладів: $\{\neg P, \neg Q \vee \neg R, P \vee Q, P \vee R\}$.

2. Нехай $S=\{P(x), \neg P(x) \vee Q(f(x)), \neg Q(f(a))\}$. Ербранівський базис множини S є $A=\{P(a), Q(a), P(f(a)), Q(f(a)),...\}$. Спростовучими прикладам и будуть: $\{\neg P(a), P(a) \vee \neg Q(f(a)), Q(f(a))\}$.

Процедура Ербрана грунтується на теоремі 2.3, тому для перевірки нездійсненності множини диз'юнктів S розглядаються тільки інтерпретації над ербранівським універсумом. Якщо S хибна при всіх інтерпретаціях над ербранівським універсумом S, то можна зробити висновок, що S нездійсненна.

Теорема 2.4 (теорема Ербрана). Множина диз'юнктів S нездійсненна тоді і тільки тоді, коли існує кінцева нездійсненна множина S' основних прикладів диз'юнктів S.

Приклад. Нехай $S = \{\neg P(x) \vee Q(f(x), x), P(g(b)), \neg Q(y, z)\}$. Це множина S нездійсненна. Одне з нездійсненних множин основних прикладів диз'юнктів множини S є $S' = \{\neg P(g(b)) \vee Q(f(g(b)), g(b)), P(g(b)), \neg Q(f(g(b)), g(b))\}$.

2.5. Метод резолюцій

Далеко не завжди можна легко знайти нездійсненну множину основних прикладів диз'юнктів. Основні труднощі полягають в породженні основних прикладів диз'юнктів і пошуку спростовуючих прикладів. Оскільки множина диз'юнктів є кон'юнктівною нормальною формою, то задача перевірки нездійсненності цієї множини еквівалентна задачі перевірки хибності кон'юнктивної нормальної форми, а це завдання експоненційної складності. Після численних пошуків більш ефективних процедур, Дж. Робінсоном був запропонований метод, названий *методом резолюцій.*

Технічно головна ідея методу резолюцій полягає в тому, щоб перевірити, чи містить множина S порожній диз'юнкт □. Якщо S містить □, то множина S нездійсненна, якщо немає, то треба перевірити, чи може він бути отриманий з даної множини диз'юнктів. Іншими словами, необхідно знайти множину основних прикладів, які спростовують вихідну множину диз'юнктів. Ця процедура заснована на правилі резолюцій.

Правило резолюцій Робінсона. Якщо для будь-яких двох диз'юнктів C_1 і C_2 існує літера $L_1 \in C_1$ і контрарна їй літера $L_2 \in C_2$ ($L_2 = \neg L_1$), то викресливши L_1 з C_1 і L_2 з C_2 і побудувавши диз'юнкт з решти літер, отримаємо *резольвенту* C_1 і C_2 : $C'_1 \vee C'_2$, де $C'_1 = C_1 \setminus L_1$, $C'_2 = C_2 \setminus L_2$.

Теорема 2.5. *Резольвента* C є логічне слідування C_1 і C_2, що містять контрарні літери L і $\neg L$: $L \vee C'_1, \neg L \vee C'_2 \models C'_1 \vee C'_2$.

Доведення. Припустимо, що $| L \vee C'_1 | = T$, $| \neg L \vee C'_2 | = T$, $| C'_1 \vee C'_2 | = F$. Тоді $|C'_1| = F$, $|C'_2| = F$. Якщо $| L \vee C'_1 | = T$, то $| L | = T$, але $| \neg L \vee C'_2 | = T$, отже, $| L | = F$. Отримане протиріччя доводить теорему. ◊

Правило резолюцій є узагальненням багатьох відомих нам правил виведення. Наприклад, правило силогізму: $A \to B, B \to C \models A \to C$ може бути переписано у вигляді: $\neg A \vee B, \neg B \vee C \models \neg A \vee C$, що відповідає правилу резолюцій.

Правило MP: $A, A \to B \models B$ може бути переписано у вигляді: $A \neg A \vee B \models B$, що також відповідає правилу резолюцій.

Нарешті, закон суперечності $A \& \neg A \equiv F$ рівнозначний правилу: $A, \neg A \models \square$, згідно з яким резольвента двох контрарних однолітерних диз'юнктів є порожній диз'юнкт.

Визначення 2.13. *Резолютивним висновком* з множини диз'юнктів S є послідовність $C_1, C_2, \ldots, C_k$, така, що кожне C_i або належить S, або є резольвентою передуючих C_i. Якщо останній диз'юнкт $C_k = \square$, то множина диз'юнктів S є нездійсненною, а весь висновок називається спростуванням S. Якщо C_k не є порожнім диз'юнктом і подальше застосування правила резолюцій неможливо, то множина S є здійсненною.

Приклад. Необхідно перевірити логічне слідування в логіці висловлювань: $P \to S, S \to R, P \models R$. Складемо множину диз'юнктів S, для чого кожну формулу приведемо до КНФ, а від висновку R візьмемо заперечення. Отримаємо :

1. $\neg P \vee S$
2. $\neg S \vee R$
3. P
4. $\neg R$
5. $\neg S$ резольвента 4, 2
6. $\neg P$ резольвента 5, 1
7. $\square$ резольвента 3, 6

Правило резолюцій – дуже потужний засіб логічного доказу. Можна показати [Чень, Лі, 1983] повноту методу резолюцій, тобто довести, що множина диз'юнктів S нездійсненна тоді і тоді, коли існує резолютивний висновок порожнього диз'юнкту з S.

2.6. Підстановка та уніфікація

Застосування методу резолюцій в логіці предикатів ускладнюється тим, що диз'юнкти містять змінні, які можуть не збігатися в двох однакових

літерах. Наприклад, $C_1 = P(y) \vee Q(y)$, $C_2 = \neg P(f(x)) \vee V(x)$. У цих диз'юнктів немає контрарних літер, проте, якщо ми підставимо $f(x)$ замість y в C_1, то отримаємо $C_1' = P(f(x)) \vee Q(f(x))$; тепер літери $P(f(x))$ і $\neg P(f(x))$ будуть вже контрарні. Отримаємо резольвенту $Q(f(x)) \vee V(x)$.

Така процедура підстановки одних термів замість інших з метою отримання контрарних літер називається *уніфікацією.*

Визначення 2.14. Підстановка - це кінцева множина виду $\{t_1/v_1, \ldots, t_n/v_n\}$, де v_i - змінна, t_i - терм, відмінний від v_i, і все v_i різні.

Визначення 2.15. Нехай $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$ і E - вираз.

Тоді θE - вираз, отриманий з E заміною всіх входжень змінних v_i $(1 \le i \le n)$ на терм t_i.

Визначення 2.16. Нехай $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$, $\lambda = \{u_1/y_1, \ldots, u_k/y_k\}$ - підстановки. Композиція підстановок $\theta \circ \lambda$ виходить з множини $\{t_1\lambda/v_1, \ldots, t_n\lambda/v_n, u_1/y_1, \ldots, u_k/y_k\}$ викреслюванням усіх елементів $t_j \lambda / v_j$, для яких $t_j \lambda = v_j$ (тотожна підстановка), і всіх елементів u_i / y_i, таких, що $y_i \in \{v_1, \ldots, v_n\}$.

Приклад. $\theta = \{t_1/v_1, t_2/v_2\} = \{f(y)/x, z/y\}$, $\lambda = \{u_1/y_1, u_2/y_2, u_3/y_3\} = \{a/x, b/y, y/z\}$.

Тоді $\theta \circ \lambda = \{t_1\lambda/v_1, t_2\lambda/v_2, u_1/y_1, u_2/y_2, u_3/y_3\} = \{f(b)/x, y/y, a/x, b/y, y/z\}$.

Так як $t_2 \lambda / x_2 = y/y$, то y/y має бути викреслено з множини $\theta \circ \lambda$. Елементи a/x, b/y також повинні бути викреслені, так як підстановки для x і y вже визначені.

В результаті отримуємо: $\theta \circ \lambda = \{f(b)/x, y/z\}$.

У процедурі доведення методом резолюцій необхідно знаходити такі підстановки, які дозволяють зробити два і більше вираження тотожними.

Визначення 2.17. Підстановка θ називається *уніфікатором* множини $\{E_1, \ldots, E_m\}$ тоді і тільки тоді, коли $\theta E_1 = \ldots = \theta E_m$. Множина $\{E_1, \ldots, E_m\}$ називається *уніфікуючим,* якщо для нього існує уніфікатор. Уніфікатор σ для множини виразів $\{E_1, \ldots, E_m\}$ називається *найбільш загальним уніфікатором,* якщо для кожного уніфікатора цієї множини існує така підстановка λ, що $\theta = \sigma \circ \lambda$.

Найбільш загальний уніфікатор є найпростішим уніфікатором для множини диз'юнктів. Так, наприклад, для двох диз'юнктів $C_1 = P(y) \vee Q(y)$, $C_2 = \neg P(f(x)) \vee V(x)$ можна використовувати уніфікатор $\{f(a)/y\}$ для C_1 і $\{a/x\}$ для C_2, в результаті чого будуть отримані диз'юнкт $C_1' = P(f(a)) \vee Q(f(a))$, $C_2' = \neg P(f(a)) \vee V(a)$ і резольвента $Q(f(a)) \vee V(a)$. Однак константа a не входить в жоден з цих виразів, тому досить зробити підстановку $\{f(x)/y\}$ в C_1, в результаті чого буде отримана резольвента $Q(f(x)) \vee V(x)$. Ця підстановка і буде найбільш загальним уніфікатором.

Ввівши поняття уніфікації, ми зможемо зараз розглянути метод резолюцій для логіки предикатів першого порядку.

Визначення 2.18. Якщо дві або більше літер (з однаковим знаком) диз'юнкта C мають найбільш загальний уніфікатор σ, то $C\sigma$ називається *склеюванням* C. Якщо $C\sigma$ – одиничний диз'юнкт, то склейка називається *одиничним склеюванням.*

Приклад. Нехай $C = P(x) \vee P(f(y)) \vee \neg Q(x)$. Тоді перша і друга літери мають найбільш загальний уніфікатор $\sigma = \{f(y)/x\}$. Отже, $C\sigma = P(f(y)) \vee \neg Q(f(y))$ є склейка C.

Визначення 2.19. Нехай C_1 і C_2 – два диз'юнкти (*диз'юнкти-посилки)*, які не мають жодних спільних змінних. Нехай L_1 і L_2 – дві літери в C_1 і C_2 відповідно. Якщо L_1 і L_2 мають найбільш загальний уніфікатор σ, то диз'юнкт $(C_1\sigma \setminus L_1\sigma) \vee (C_2\sigma \setminus L_2\sigma)$ називають *(бінарною) резольвентою* C_1 і C_2. Літери L_1 і L_2 називаються *відрізаними літерами.*

Визначення 2.20. *Резольвенти* диз'юнктів-посилок C_1 і C_2 є однією з наступних резольвент:

1) бінарна резольвента C_1 і C_2;
2) бінарна резольвента C_1 і склейка C_2;
3) бінарна резольвента C_2 і склейка C_1;
4) бінарна резольвента склейки C_1 і склейка C_2.

Приклад. Нехай $C_1 = P(x) \vee P(f(y)) \vee R(g(y))$ і $C_2 = \neg P(f(a)) \vee Q(b)$. Склеювання C_1 є $C_1' = P(f(y)) \vee R(g(y))$. Бінарна резольвента C_1' і C_2 є $R(g(g(a))) \vee Q(b)$. Отже, $R(g(g(a))) \vee Q(b)$ є резольвента C_1 і C_2.

2.7. Практичне застосування методу резолюцій

Розглянемо приклад:

***F*1:** $\exists x(P(x) \& \forall y(D(y) \to L(x, y)))$

***F*2:** $\forall x(P(x) \to \forall y(Q(y) \to \neg L(x, y)))$

Висновок *G*: $\forall y(D(y) \to \neg Q(y))$

Приведмо посилки в ПНФ, а потім в ССФ:

1) $\forall y(\neg D(y) \vee L(a, y)) \& P(a)$,

2) $\forall x \forall y(\neg Q(y) \vee \neg L(x, y) \vee \neg P(x))$.

Знайдемо заперечення від висновку G і наведемо його до ПНФ; отримаємо:

$\neg \forall y(D(y) \to \neg Q(y)) = \exists y \neg(\neg D(y) \vee \neg Q(y)) = \exists y(D(y) \& Q(y))$.

Елімінуючи квантор ∃ і отримаємо ССФ: $D(b) \,\&\, Q(b)$.

Побудуємо резолютивний висновок:

1. $\neg D(y) \vee L(a, y)$
2. $P(a)$
3. $\neg Q(y) \vee \neg L(x, y) \vee \neg P(x)$
4. $D(b)$

5. $Q(b)$

6. $\neg L(x, b) \vee \neg P(x)$ $\{b/y\}$, резольвента 5,3

7. $\neg L(a, b)$ $\{a/x\}$, резольвента 2,6

8. $\neg D(b)$ $\{b/y\}$, резольвента 1,7

9. □ резольвента 4,8

Може виникнути враження, що метод резолюцій є алгоритмом доведення загальнозначущості формул, тобто розв'язною процедурою логіки предикатів. Однак це не так. Як приклад розглянемо задачу про цирульника: *Цирульник голить тих і тільки тих жителів села, які голяться самі. Чи повинен цирульник голити самого себе?*

Візьмемо предикати: *P (x) - x цирульник, Q (x, y) - x голить y.* Формалізуємо посилки і перетворимо їх до ССФ:

$\forall y\ (P(y) \rightarrow \forall x\ (Q(x, x) \rightarrow \neg Q(x, y))) => \forall x \forall y(\neg P(y) \vee \neg Q(x, x) \vee \neg Q(x, y))$;

$\forall y\ (P(y) \rightarrow \forall x(\neg Q(x, x) \rightarrow Q(x, y))) => \forall x \forall y(\neg P(y) \vee \neg Q(x, x) \vee Q(x, y))$.

Висновок: $\forall y\ (P(y) \rightarrow Q(y, y))$. *Беремо заперечення від висновку:*

$\neg \forall y\ (P(y) \rightarrow Q(y, y)) = \exists y \neg\ (\neg P(y) \vee Q(y, y)) = \exists y\ (P(y)\ \&\ \neg Q(y, y))$. *Це ПНФ. Елемінуємо квантор існування для отримання ССФ та множини диз'юнктів:*

$P(c)\ \&\ Q(c, c) =>\ P(c)$ та $Q(c, c)$.

Перевіримо, чи повинен цирульник голити самого себе: $Q\ (c, c)$ *(якщо в нас цірюльник є с)*. Побудуємо висновок:

1. P(с)

2. $\neg P(y) \vee \neg Q(x, x) \vee \neg Q(x, y)$

3. $\neg P(y) \vee Q(x, x) \vee Q(x, y)$

4. $\neg Q(c, c)$

5. $Q(x, x) \vee Q(x, c)$ підстановка $\{c/y\}$, резольвента 1, 3

6. $Q(c, c)$ підстановка {с/x} в 5, склейка 5

7. □ резольвента 4, 6

Логічне слідування виконано, тобто цирульник повинен голити самого себе.

Поставимо протилежне питання: цирульник не повинен голити самого себе: $\neg Q\ (c, c)$. І побудуємо резолютивний висновок:

1. $P(c)$

2. $\neg P(y) \vee \neg Q(x, x) \vee \neg Q(x, y)$

3. $\neg P(y) \vee Q(x, x) \vee Q(x, y)$

4. $Q(c, c)$

5. $\neg Q(x, x) \vee \neg Q(x, c)$ підстановка $\{c/y\}$, резольвента 1, 2

6. $\neg Q(c, c)$ підстановка {с/x} в 5, склейка 5

7. □ резольвента 4, 6

Логічне слідування також виконано, тобто цирульник не повинен голити самого себе.

У цьому завданні на два протилежних питання ми отримуємо однакову відповідь. Цей приклад показує, що резолютивний висновок не є спільною розв'язною процедурою, і причина тут полягає не в недоліках методу резолюцій, а у властивостях теорії предикатів першого порядку.

Взагалі-то, в цьому завданні використання будь-якого виведення призведе до порожнього диз'юнкту, якщо зауважити, що множина посилок, що складають умову, є суперечливою.

Візьмемо тільки посилки без всякого заперечення будь-якого виведення і додамо факт існування y=c (існування цирюльника – це є незмістовною частиною заперечення висновку):

1. $P(c)$

2. $\neg P(y) \vee \neg Q(x, x) \vee \neg Q(x, y)$

3. $\neg P(y) \vee Q(x, x) \vee Q(x, y)$

4. $\neg Q(x, x) \vee \neg Q(x, c)$ підстановка $\{c/y\}$, резольвента 1, 2

5. $Q(x, x) \vee Q(x, c)$ підстановка $\{c/y\}$, резольвента 1, 3

Ми можемо застосувати узагальнюючу склейку для 4 і 5, і отримаємо:

6. $\neg Q(x, y)$

7. $Q(x, y)$

Тоді в результаті застосування правила резолюцій Робінсона для 6 і 7 отримаємо в якості резольвенти □, що говорить про те, що множина диз'юнктів суперечлива і може служити адекватною постановкою завдання.

В процесі резолютивного виведення можуть породжуватися зайві резольвенти. Для скорочення кількості розрахунків застосовуються спеціальні стратегії, що скорочують перебори: спрощення, очищення і впорядкування.

1. стратегія упорядкування:

множина літер всередині диз'юнктів упорядковуються за абеткою (це полегшує перевірку);

2. стратегія очищення:

записуємо S_0 диз'юнкт, отриманий із заперечення висновку; також зручно починати будувати висновок з фактів (висловлювання) – однолітерних диз'юнктів;

3. стратегія спрощення:

в множині посилок можуть з'явитися тавтології - вони викреслюються з посилок і резольвент; викреслюються диз'юнкти, що містять унікальні літери (крім диз'юнктів, що поглинаються).

Метод резолюцій - це дуже сильний метод пошуку доказів загальнозначущості формул (в іншому формулюванні - логічних висновків). Саме тому він породив нову парадигму програмування - логічне програмування. Найбільш поширеною мовою логічного програмування є ПРОЛОГ. У логічному програмуванні будь-яке завдання ставиться як задача доведення логічного слідування деякої пропозиції із заданих

посилок. Виконання програми полягає в пошуку доказів методом резолюцій, тобто пошуку порожнього диз’юнкту.

Питання до розділу 2

1. Основна ідея методу резолюції. Правило резолюцій Робінсона. Отримання резольвент, побудова резолютивного виведення. Приклад застосування методу резолюцій для перевірки логічного слідування в логіці висловлювань. Особливості застосування методу резолюцій для логіки 1-го порядку.
2. Метод резолюцій для логіки 1-го порядку: підстановка, уніфікація. Найбільш загальний уніфікатор.
3. Визначення диз'юнкту. Визначення порожнього диз'юнкту. Визначення множини диз'юнктів. Визначення здійсненності і нездійсненності множини диз'юнктів.
4. Попереджені нормальні форми (ПНФ). Доказ теореми про приведення формули до ПНФ.
5. Скулемовські стандартні форми (ССФ). Приведення до ССФ. Теорема про суперечливість формули при суперечливості її стандартної форми.
6. Ербрановській універсум множини диз'юнктів, основний приклад. Теорема Ербрана (без доведення).

3. ЛОГІЧНЕ ПРОГРАМУВАННЯ

3.1. Логіка та управління

Логічне програмування – парадигма програмування[4], заснована на автоматичному доказі теорем, а також розділ дискретної математики, що вивчає принципи логічного висновку інформації на основі заданих фактів і правил виведення. Логічне програмування засноване на теорії та апараті математичної логіки з використанням математичних принципів резолюцій.

Технологія логічного програмування дозволяє досить близько наблизитися до втілення декларативного ідеалу, згідно з яким системи повинні конструюватися шляхом подання знань на деякій формальній мові, а задачі розв'язуватися шляхом застосування процесів логічного висновку до цих знань. Такий ідеал виражений в наступному рівнянні Роберта Ковальського:

Алгоритм = Логіка + Управління

Свої дослідження Ковальський почав в області автоматичних доказів, відомий його внесок в розвиток логічного програмування, починаючи з процедурної інтерпретації Хорна. Ковальський був одним з перших розробників абдуктівной логіки програмування, де логічні програми доповнені обмеженнями цілісності і з невизначеними, абдуктивними предикатами[5].

Розробка мови Prolog почалася в 1970 році Аланом Кулмером і Філіпом Русселом. Вони хотіли створити мову, яка змогла би робити логічні висновки на основі заданого тексту. Назва Prolog є скороченням від «PROgramming in LOGic». Ця мова була розроблена в Марселі в 1972 році. Принцип резолюції Ковальського здавався підходящою моделлю, на основі якої можна було розробити механізм логічних висновків. З обмеженням резолюції на диз'юнкт Хорна уніфікація привела до ефективної системи, де нездоланний недетермінізм оброблявся за допомогою процесу відкату, який міг бути легко реалізований. Алгоритм резолюції дозволяв створити виконувану послідовність, необхідну для реалізації специфікацій, подібних до наведеного вище відношенню.

---

[4] **Парадигма програмування** - це сукупність ідей і понять, що визначають стиль написання комп'ютерних програм (підхід до програмування). Це спосіб концептуалізації , що визначає організацію обчислень і структурування роботи, що виконується комп'ютером. Парадигма програмування не визначається однозначно мовою програмування.

[5] **Абдукція** - пізнавальна процедура висунення гіпотез. Абдукція є вид редуктивного виведення з тією особливістю, що з посилки, яка є умовним висловлюванням, і висновку випливає друга посилка.

Prolog (ПРОЛОГ) є найбільш поширеною логічною мовою. Він служить для вирішення завдань символічних маніпуляцій, таких як написання компіляторів і синтаксичних аналізаторів текстів на природній мові; для написання експертних систем для юридичних, медичних, фінансових та інших проблемних областей.

Отже, ПРОЛОГ орієнтований на логічне уявлення знань, а виконання програм на пролозі засноване на методі резолюцій.

3.2. Структура логічної програми

Сама логічна програма на Пролозі складається із сукупності фактів, які утворюють базу знань. Завдання формулюється як питання про деякі факти або відношення, визначених у базі знань. Виконання завдання - це перевірка логічного слідування (з фактів і правил, визначених у базі знань). Висновок щодо логічного слідування здійснюється методом резолюцій, який реалізується виконанням самої програми. Сам метод резолюцій прихований.

Логічна програма включає в себе: предикати, які вводяться при формалізації (блок - predicates); сукупність фактів і правил (блок - clauses); запит до бази знань (goal? - ВИРАЗ), за яким відбувається перевірка логічного слідування.

Програма видає відповідь ТАК чи НІ (виконується чи ні логічне слідування), або результат уніфікації (підстановки), яка саме би і привела до отримання □.

Програма на Пролозі виглядає наступним чином:

- Оголошення е предикатів

Predicates
<Список імен предикатів>

- База знань

Clauses
<Список фактів>
<Список правил>

- Ціль, мета (запитання до бази знань)

Goal ? – <вираз>.

ПРОЛОГ є яскравим представником декларативних мов. Текст програми на пролозі не містить в явному вигляді процедури формування відповіді на запит. Зрозуміло, виконавча система Пролог таку процедуру містить (метод резолюцій).

По суті, складання логічної програми полягає в формалізації знань про предметну область на мові логіки предикатів першого порядку.

3.2.1. Складання логічної програми

Розглянемо основні принципи Пролог на прикладі складання логічної програми про родинні стосунки. Будемо використовувати узагальнений синтаксис, не прив'язуючись ні до якої конкретної реалізації мови ПРОЛОГ.

Оголошення предикатів

Predicates

Батько (*x*, *y*)

Мати (*x*, *y*)

Дід (*x*, *y*)

Визначення фактів.

Факт – це висловлювання про об'єкти предметної області. Факт записується як предикат, в який на місце змінних поставлено конкретні об'єкти з предметної області.

Факти можуть бути позитивними, наприклад: Марія – мати Івана, або негативним, наприклад: Іван не є студентом. Для запису негативних фактів і заперечень виразів в логічній програмі використовуємо службове слово *not*: *not* (студент (Іван)).

Запишемо множину фактів щодо родинних зв'язків:

clauses

Мати (Марія, Іван).

Батько (Петро, Іван).

Батько (Петро, Борис).

Батько (Іван, Анна).

Визначення правил.

Змістовно правила Пролог - це імплікації «*якщо ... то*», посилки яких - кон'юнкції предикатів. У численні предикатів вони виражалися б формулами виду

$$\forall x \forall y \ldots \forall z(P_1 \& P_2 \& \ldots \& P_n \rightarrow R),$$

де кожна змінна у формулах P_1, P_2, ..., P_n і R, пов'язана квантором загальності, тобто формулами в стандартній скулемівській формі.

У методі резолюцій така формула приводиться до диз'юнктів

$$\neg P_1 \vee \neg P_2 \vee \ldots \vee \neg P_n \vee R,$$

в якому тільки одна літера позитивна (не містить заперечення).

Диз'юнкт, що містить не більше однієї позитивної літери, називається ***хорнівським диз'юнктом***[6].

---

[6] **Хорнівський диз'юнкт** - вивчений Альфредом Хорном став основою для мови логічного програмування Пролог. Диз'юнкти Хорна пов'язані з доказом теорем через резолюції першого порядку, так як резолюція двох хорнівських диз'юнктів є хорнівським диз'юнктом. В автоматичному доведенні теорем це може давати велику ефективність.

3.2.2. Правила, хорнівські диз'юнкти

Хорнівські диз'юнкти можуть приймати один з наступних видів:

1. A
2. ¬A∨¬B
3. A∨¬B∨¬C

Диз'юнкт Хорна з рівно одним позитивним літералом є ***визначеним диз'юнктом або точним*** (3); диз'юнкт Хорна без позитивних литералов іноді називається метою або запитом (2). Диз'юнкт, що містить тільки **одну позитивну** літеру, називають **фактом** (1) або одиночним Хорнівським диз'юнктом, а що містить **позитивні і негативні** літери - **правилом**. ***Формула Хорна*** - кон'юнкція диз'юнктів Хорна, тобто формула в кон'юнктивній нормальній формі, все диз'юнкт якої є хорнівськими.

Процедура резолюцій найбільш ефективна, якщо кожен диз'юнкт - Хорнівський, тобто диз'юнкт, що містить не більше однієї позитивної літери.

Якщо правило має негативну форму:

$$\forall x \forall y \ldots \forall z(P_1 \& P_2 \& \ldots \& P_n \rightarrow \neg R),$$

то йому відповідає хорнівський диз'юнкт

$$\neg R \vee \neg P_1, \vee \neg P_2 \vee \ldots \vee \neg P_n ,$$

який не містить позитивних літер.

У пролозі вирази, що описують правила, називаються *clause (клоз)*. Їх синтаксис відрізняється від традиційного синтаксису числення предикатів.

Позитивний *clause* має вигляд:

$R : - P_1, P_2, ..., P_n,$

що читається як висловлювання: R , якщо $P_1 \& P_2 \& ... \& P_n$ (висновок стоїть перед посилкою).

Негативний *clause* має вигляд:

$: - R , P_1, P_2, ..., P_n,$

Наприклад, формула

$\forall x (P (x) \& Q (x) \rightarrow R (x))$

у вигляді *clause* запишеться як

$R (x): - P (x), Q (x).$

Таким чином, *clause* є різновид записи диз'юнкта, в якому позитивні і негативні літери розділені символами : - , причому, позитивні літери пишуться зліва, а негативні - справа.

диз'юнкт	***clause***
A	A :– або A
A∨¬B	A :– B

¬A∨¬B	:– A, B
¬B	:– B
A∨¬B∨¬C	A :– B, C
A∨B∨¬C	A, B :– C
A∨B	A, B :–

Оголошений предикат (відношення) Дід (*x*, *y*) визначимо за допомогою правила, яке в логіці предикатів запишеться так:

$\forall x \forall y \forall z((\text{батько}(x,z)\&\text{мати}(z,y) \vee \text{батько}(x,z)\&\text{батько}(z,y) \rightarrow \text{дід}(x,y))$.

Диз'юнкція двох умов в посилці записується як два *clause* з однаковим висновком:

дід (*x* , *y*) : – батько (*x* , *z*), мати (*z* , *y*).

дід (*x* , *y*) : – батько (*x* , *z*), батько (*z* , *y*).

Цей розділ програми утворює опис бази знань.

Заключний розділ програми *Goal* (мета) буде містити питання до бази знань. Він має синтаксис:

Goal ? – <вираз>.

Отже, наша програма має вигляд:

Predicates

Батько (*x* , *y*)

Мати (*x* , *y*)

Дід (*x* , *y*)

clauses

Мати (Марія, Іван).

Батько (Петро, Іван).

Батько (Петро, Борис).

Батько (Іван, Анна).

дід (*x* , *y*)): – батько (*x* , *z*), мати (*z* , *y*).

дід (*x* , *y*)): – батько (*x* , *z*), батько (*z* , *y*).

***Goal* ? –**

Ми тут в прикладі не привели ніякого виразу, яке могло б служити метою. Ми розглянемо нижче різні приклади запитів. Взагалі, в пролозі існує таке поняття як внутрішня і зовнішня мета. Внутрішня мета - це питання, яке стоїть прямо в тілі програми. Зовнішня - сама програма не містить службове слово *Goal* , але після запуску в робочому вікні з'являється *Goal* ? – , і питання задаються в інтерактивному режимі.

3.2.3. Блок Goal

Розглянемо можливі цілі, тобто питання до нашої бази знань.

Найпростіший питання - про наявність в базі знань деяких фактів, наприклад:

Goal **? – Батько (Петро, Іван).**

(Чи є Петро батьком Івана?).

Відповідь: так.

Знак « –» перед виразом в питанні означає, що при пошуку відповіді від цього виразу береться заперечення, а потім виконується резолютивний висновок на множині фактів, правил і заперечення виразу, записаного в якості запитання. Якщо ця множина виявиться суперечливою, тобто буде отримано порожній диз'юнкт, то логічне слідування виконано, і програма відповість «так» (*yes)*, в іншому випадку вона відповість «ні» (*no*).

Наприклад, на питання:

Goal **? – Батько (Петро, Анна).**

Чи є Петро батьком Анни?

Відповідь: ні.

Інші типи питань.

Goal **? – Батько (** ***x*** **, Анна).**

Хто є батьком Анни?

Відповідь: *x* = Іван.

Відповіддю є та уніфікація - підстановка константи (імені), яка призводить до отримання порожнього диз'юнкту.

Goal **? – Батько (Петро,** ***y*****).**

Чий батько Петро?

Викличе послідовність відповідей:

Відповідь: *y* = Іван

y = Борис

Будуть знайдені всі значення, що задовольняють істинності предиката Батько (Петро, *y*).

Питання ***Goal*** **? – Батько (Петро, _)** означає:

Чи є діти у Петра?

Відповідь: так.

Відповідь «так» буде отримано, якщо буде знайдена хоча б одна підстановка, яка призведе до порожнього диз'юнкту.

Goal **? – Батько (** ***x*** **,** ***y*** **).**

Хто чий батько?

Призведе до відповідей:

x = Петро, *y* = Іван.
x = Петро, *y* = Борис.
x = Іван, *y* = Анна.
***Goal* ? – Батько (__, __).**
Чи є люди, що перебувають у відношенні *x* батько *y*?
Відповідь: так.

3.3. Особливості мови програмування Prolog

Програми на мові Prolog представляють собою множини певних виразів, записаних в системі позначень, яка трохи відрізняється від використовуваної стандартної логіки першого порядку. У мові Prolog великі літери застосовуються для позначення змінних, а малі - для позначення констант. Вирази записуються з головою, що передує тілу; символ : - служить для позначення імплікації, направленої ліворуч, коми поділяють літерали в тілі, а точка позначає кінець висловлювання, як показано нижче.

criminal(X) :- american(X), weapon(Y), sells(X, Y, Z), hostile(Z).

Мова Prolog включає «синтаксичні спрощення» (syntactic sugar) для позначення списків і арифметичних виразів. Наприклад, нижче приведена програма Prolog для предиката append (X, Y, Z), яка виконується успішно, якщо список Z являє собою результат доповнення списку Y списком X.

append ([], Y, Y).

append ([A [X], Y, A [Z]]): - append (X, Y, Z).

Природною мовою ці вирази можна прочитати так: по-перше, доповнення списку Y порожнім списком призводить до отримання того ж списку Y, і, по-друге, [A | Z] - це результат доповнення списку Y списком [A | X], за умови, що Z - це результат доповнення списку Y списком X. Таке визначення предиката append на перший погляд здається досить подібним відповідному визначенню мовою Lisp, але фактично є набагато більш потужним. Наприклад, в систему можна ввести запит append (A, B, [1, 2]) - які два списки можна доповнити один іншим, щоб отримати [1, 2]? Система поверне наступні рішення:

A = [] B = [1, 2]

A = [1] B = [2]

A = [1, 2] B = []

Виконання програм Prolog здійснюється за принципом зворотного логічного висновку з пошуком в глибину, при якому спроба застосування виразів виконується в тому порядку, в якому вони записані в базу знань. Але деякі описані нижче особливості мови Prolog виходять за рамки стандартного логічного висновку.

- У ньому передбачена множина вбудованих функцій для виконання арифметичних операцій. Літерали, в яких використовуються відповідні функціональні символи, «доводяться» шляхом виконання коду, а не здійснення подальшого логічного висновку. Наприклад, мета «X is 4 + 3» досягається успішно після зв'язування змінної X зі значенням 7. З іншого боку, спроба досягнення мети «5 is X + Y» закінчується невдачею, оскільки ці вбудовані функції не забезпечують вирішення довільних рівнянь (слід зазначити, що якби в деякій програмі Prolog були передбачені аксіоми Пеано, то такі цілі могли бути вирішені за допомогою логічного висновку).
- У мові передбачені вбудовані предикати, що викликають при їх виконанні побічні ефекти. До них відносяться предикати введення-виведення і предикати assert / retract для модифікації бази знань. Такі предикати не мають аналогів в логіці і можуть породжувати деякі ефекти, що викликають плутанину, наприклад, якщо факти підтверджуються (і вводяться в базу знань) деякої гілкою дерева доказів, яка в кінцевому підсумку закінчується невдачею.
- У мові Prolog допускається певна форма заперечення - заперечення як недосягнення мети. Заперечувальна мета not P вважається доведеною, якщо системі не вдається довести P. Таким чином, наступний вислів:

Alive (X): = not dead (X) можна прочитати так: «Будь-якого слід вважати живим, якщо не можна довести, що він мертвий».

- У мові Prolog передбачений оператор рівності, «=», але він не володіє всією міццю логічного рівності. Мета з оператором рівності досягається успішно, якщо в ній два терми є уніфікуючими, а в іншому випадку спроба її досягнення закінчується невдачею. Таким чином, мета X + Y = 2 + 3 досягається успішно, після зв'язування змінної X зі значенням 2, а Y – зі значенням 3, а спроба досягнення мети morningstar = eveningstar закінчується невдачею (в класичній логіці остання рівність може бути чи не бути істинною). Не можуть бути підтверджені (введені в базу знань) будь-які факти або правила, що стосуються рівності.
- З алгоритму уніфікації Prolog виключена перевірка входження. Це означає, що можуть бути зроблені деякі суперечливі логічні висновки; така проблема виникає рідко, за винятком тих ситуацій, коли мова Prolog використовується для доказу математичних теорем.

3.4. Практична частина

Розглянемо, як виглядають логічні програми для наших улюблених задач з попередніх лекцій: про птахів з рибами, пінгвінів і бджолярів з хіміками. Також покажемо, як перевіряється логічне слідування для цих шаблонів методом резолюцій.

Пример 1. $F1$: $\forall x(D(x) \to L(x))$,

$F2$: $\forall x(Q(x) \to \neg L(x))$.

Висновок G: $\forall x(D(x) \to \neg Q(x))$.

Приведемо посилки до ПНФ:

$\forall x(\neg D(x) \vee L(x))$

$\forall x(\neg Q(x) \vee \neg L(x))$

Через те, що посилки не містять кванторів існування, ПНФ і ССФ в даному випадку співпадають.

Візьмемо заперечення висновку $\forall x(D(x) \to \neg Q(x))$:

$\neg\ \forall x(D(x) \to \neg Q(x)) = \exists x\ \neg(\neg\ D(x) \vee \neg Q(x)) = \exists x\ (D(x)\ \&\ Q(x))$.

Це – ПНФ. Для отримання ССФ треба елімінувати квантор існування. Через те, що йому не передує жодний з кванторів загальності, можемо просто змінити x на a: $D(a)\ \&\ Q(a)$.

Отримаємо наступну множину диз'юнктів:

$\{D(a),\ Q(a),\ \neg D(x) \vee L(x),\ \neg Q(x) \vee \neg L(x)\}$. Перевіримо множину на несуперечливість:

1. $D(a)$
2. $Q(a)$
3. $\neg D(x) \vee L(x)$
4. $\neg Q(x) \vee \neg L(x)$
5. $L(a)$ резольвента (1,3) {a/x}
6. $\neg Q(a)$ резольвента (4,5) {a/x}
7. □ резольвента (2,6)

Порожній диз'юнкт отримано, множина із запереченням висновку є суперечливою. Отже, сам висновок є логічним слідуванням з множини посилок-умов задачі.

<u>Побудуємо логічну програму:</u>

Візьмемо множину посилок, що відповідає умові задачі (тільки посилок, без висновку або його заперечення) у вигляді диз'юнктів: $\{\ \neg\ D\ (x) \vee L\ (\ x\ ),\ \neg\ Q\ (x) \vee \neg\ L\ (\ x\ )\ \}$.

УВАГА! Заперечення висновку не беремо, це зробить за нас виконавча процедура ПРОЛОГУ.

Розглянемо висновок $\forall x\ (D\ (x) \to \neg Q\ (x))$. Наша програма не містить жодної константи, тобто жодного об'єкта для ініціалізації і початку потоку обчислень. А що, якщо поставити питання таким чином: чи буде для будь-якого $x=a$ з області визначення (Ербрановскій базис) при істинному ***D* (a) слідувати істинність НІ *Q* (a)?** Т аким чином ми можемо занести ***D* (a)** в область фактів, а базі знань поставити питання щодо **НІ *Q* (a).**

Зверніть увагу, висновок ми ніяк не перетворюємо, а розбиваємо на факт і питання. Антецедент - це факт, а консеквент - питання. Предикати беруться

в такому ж вигляді, як вони записані в висновку, тільки підставляється константа замість змінної.

$$\forall x(D(x) \rightarrow \neg Q(x))$$

факт ***D*(a)** питання **not (*Q*(a))**

Predicates *Q*(x), *D*(x), *L*(*x*)
Clauses
***D*(a)**
L(*x*) :– *D*(x)
:– *Q*(*x*), *L*(*x*)
Goal ? – **not (*Q*(a)).**

Факти записуються перед правилами; правила - форма запису диз'юнктів; знак заперечення ¬ в логічній програмі як символ відсутній.

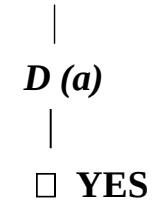
Побудуємо дерево логічного висновку, тобто проімітуємо роботу ПРОЛОГУ (метод резолюції).

Програма ПРОЛОГ починає роботу завжди з заперечення питання. І таким чином, якщо повернутися в попередній розділ - метод резолюцій, то можна побачити, що в даній задачі в множині диз'юнктів, які перевірялися на суперечливість, в результаті заперечення висновку і перетворювання його в ПНФ, а потім в ССФ, були додані два однолітерних предиката з константою (як наслідок елімінування квантора існування) - *Q* (a) , *D* (a). Саме ці факти і з'являються у нас в дереві логічного висновку.

Слід зауважити, що ПРОЛОГ переглядає набір *clauses* на кожному кроці з самого початку.

Clauses
***D*(a)**
L(*x*) :– *D*(x)
:– *Q*(*x*), *L*(*x*)

Q (a)
|
:– Q(x), L(x)
{a/x}
|
:- L (a)
|
L(x) :– D(x)
{a/x}
|
:-D (a)



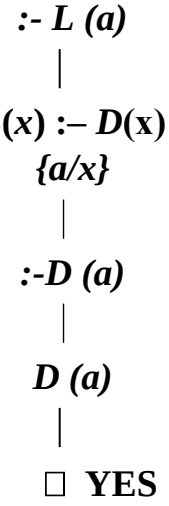
Приклад 2. . *F*1: ∀x(*D*(x) → *L*(*x*)),
*F*2: *D*(a)
Висновок *G*: *L*(*a*).

Приведемо посилки до ПНФ (ССФ), візьмемо заперечення висновку та складемо множину диз'юнктів: { ¬*L*(*a*) , *D*(a) , ¬*D*(x) ∨ *L*(*x*) }.

1. ¬*L*(*a*)
2. *D*(a)
3. ¬*D*(x) ∨ *L*(*x*)
4. ¬*D*(a) резольвента (1,3) {a/x}
5. □ резольвента (2,4)

Для побудови логічної програми візьмемо тільки множину посилок: { *D* (a) , ¬*D* (x) ∨*L* (*x*) }. З висновком тут все просто - висновок і є питання до бази знань *L* (*a*). І константа для ініціалізації вже є - *a* - пінгвін. Побудова дерева логічного висновку почнеться з заперечення ПИТАННЯ (не висновку, а саме питання до бази знань, хоча в цій задачі вони співпадають).

Predicates D(x), *L*(*x*)
Clauses
D(a)
L(*x*) :– *D*(x)
Goal ? – ***L* (a).**



Приклад 3. *F*1: ∃x(H(x)&A(x))
*F*2:∀x(C(x)→¬H(x))
Висновок *G*: ∃x(¬C (x)&A(x))

Перша посилка вже знаходиться в ПНФ, отримаємо ССФ - H(a)&A(a).

Для другої гипотези ПНФ та ССФ буде однаковими: ∀x(¬C(x) ∨ ¬H(x)).

Візьмемо заперечення висновку:

$\neg\exists x(\neg C(x)\&A(x)) = \forall x\neg(\neg C(x)\&A(x)) = \forall x\,(C(x) \vee \neg A(x))$.

Множина диз'юнктів: { $C(x) \vee \neg A(x)$, $H(a)$, $A(a)$, $\neg C(x) \vee \neg H(x)$ }.

Побудуємо резолютивний висновок:

1. $C(x) \vee \neg A(x)$
2. $H(a)$
3. $A(a)$
4. $\neg C(x) \vee \neg H(x)$
5. $\neg A(x) \vee \neg H(x)$ резольвента (1,4)
6. $\neg A(a)$ резольвента (2,5) {a/x}
7. □ резольвента (2,4)

Для побудови логічної програми візьмемо множину диз'юнктів від умови задачі: { H (a) , A (a), ¬C (x) ∨ ¬H (x) }. Константа в задачі вже є.

Висновок представляється в якості питання до бази знань: чи знайдеться такий x, який буде не хіміком і при цьому художником? У дереві логічного висновку заперечення цього питання буде носити узагальнений характер, що і відбувалося в методі резолюцій, коли при отриманні ССФ ми переходили до квантору загальності

Predicates H(x), A(x), C(x)
Clauses
H(a)
A(a)
:– C(x), H(x)
Goal ? – ***not(C(x)), A(x).***

C(x) :– A(x)
|
A (a)
{a/x}
|
C (a)
|
:– C(x), H(x)
{a/x}
|
:– H(a)
|
H (a)
|
□ **YES**

Приклад 4. Кожен раз, отримуючи в кінці порожній диз'юнкт, ми робимо висновок про істинність логічного слідування, яке перевіряли. Ходом свого дослідження ми спростовуємо ідею про можливість існування заперечення виведення при істинності посилок (і при їх несуперечливої природі).

Розглянемо задачу при суперечливій множині початкових посилок. Порожній диз'юнкт міститься в самоій умові. Висновок можна отримати з «неправди» - будь-який.

Всі птахи літають. Існує птах страус, яка не літає.

1. ∀x(B(x)→L(x))

2. B(s)&¬L(s)

Якщо ми перевіримо множину посилок на несуперечливість, то ми отримаємо:

1. ¬B(x) ∨ L(x)
2. B(s)
3. ¬L(s)
4. L(s) резольвента (1,2) {s /x}
5. □ резольвента (3,4)

Або такий варіант: *Всі птахи літають. Страус не літає. Виходить, що страус - не птах?*

1. ∀x(B(x)→L(x))

2. ¬L(s)

Висновок: ¬B(s).

1. ¬B(x) ∨ L(x)
2. ¬L(s)
3. B(s) – заперечення висновку
4. L(s) резольвента (1,3) {s /x}
5. □ резольвента (2,4)

Ось так результат! А в чому парадокс?

Приклад 5.

Відповідно до принципу Пітера, *службовець просувається по службових сходах до тих пір, поки він не досягне свого рівня некомпетентності.* Чи випливає з цього, що *не існує компетентних начальників?*

Перевіримо цей висновок, використовуючи метод резолюцій. Виберемо предикати: *C* (*x*): *x - службовець, K* (*x*): *x - компетентний, N* (*x*): *x - начальник, P* (*x*) : *x просувається по службових сходах.* Формалізуємо свої знання про проблему. У якості першої посилки візьмемо твердження про те, що *компетентний службовець просувається по службових сходах*:

$\forall x\,(C(x)\ \&\ K(x) \rightarrow \square P(x))$.

Другий посилкою може служити твердження про те, що *начальник перестає просуватися по службових сходах*:

$\forall x\,(N(x) \rightarrow \neg P(x))$.

Врахуємо також той факт, що *начальник - теж службовець*:

$\forall x\,(N(x) \rightarrow C(x))$.

Тоді висновок, який потрібно перевірити, можна сформулювати так:

«*Всі начальники некомпетентні*»:

$\forall x\,(N(x) \rightarrow \neg K(x))$.

Приведемо посилки і заперечення висновку до ССФ і побудуємо резолютивний висновок:

1. $N(a)$ - *від заперечення висновку*
2. $K(a)$ - *від заперечення висновку*
3. $\neg C(x) \vee \neg K(x) \vee P(x)$
4. $\neg N(x) \vee \neg P(x)$
5. $\neg N(x) \vee C(x)$
6. $\neg C(a) \vee P(a)$ *{a / x} в 3, резольвента 2,3*
7. $\neg N(a) \vee P(a)$ {a / x} в 5, резольвента 5,6
8. $\neg N(a)$ {a / x} в 4, резольвента 4,7
9. □ резольвента 1,8

Отже, ми отримали, що компетентних начальників не існує.

Побудуємо формальний висновок ($\forall x\,(N(x) \rightarrow \neg K(x))$):

1. $\forall x\,(C(x) \,\&\, K(x) \rightarrow P(x))$
2. $\forall x\,(N(x) \rightarrow \neg P(x))$
3. $\forall x\,(N(x) \rightarrow C(x))$
4. $C(y) \,\&\, K(y) \rightarrow P(y)$ *УК1*
5. $N(y) \rightarrow \neg P(y)$ *УК2*
6. $N(y) \rightarrow C(y)$ *УК3*
7. $P(y) \rightarrow \neg N(y)$ *ПК 5*
8. $C(y) \,\&\, K(y) \rightarrow \neg N(y)$ *сил.4,7*
9. $C(y) \rightarrow (K(y) \rightarrow \neg N(y))$ *екв.8*
10. $N(y) \rightarrow (K(y) \rightarrow \neg N(y))$ *сил.6,9*
11. $\neg N(y) \vee K(y) \vee \neg N(y)$ *екв.10*
12. $N(y) \rightarrow \neg K(y)$ *екв.11*
13. $\forall x\,(N(x) \rightarrow \neg K(x)$ *Gen 12*

Розглянемо інший, більш елегантний спосіб:

7. $N(y) \rightarrow \neg P(y) \,\&\, C(y)$ *введення &, дистрибутивність 5,6*
8. $\neg C(y) \vee \neg K(y) \vee P(y) = C(y) \,\&\, \neg P(y) \rightarrow \neg K(y)$ *екв.4*
9. $N(y) \rightarrow \neg K(y)$ сил.7,8
10. $\forall x\,(N(x) \rightarrow \neg K(x)$ Gen 9

Питання до розділу 3

1. Логічне програмування – основні поняття.
2. З яких блоків складається логічна програма?
3. Хорновскій диз'юнкт. Факт. Визначення клауз.
4. Дерево резолютивного виведення – з чого починається побудова, як обираються диз’юнкти для наступного крока?
5. Відмінність логічного висновку та мети логічної програми. Як формується запиання до бази знань?
6. Особливості логічного програмування (запис фактів, правил; що означає питання -?; у якому випадку отримують відповідь «так»,«ні»?).

4. ПРИМІТИВНО РЕКУРСИВНІ ТА РЕКУРСИВНІ ФУНКЦІЇ

Успіхи розвитку дедуктивного методу формалізації міркувань породили велику кількість робіт по формалізації основних розділів математики. Ідея створення універсальної мови для всієї математики і формалізації математичних доказів засобами цієї мови висувалася ще Лейбніцем. Однак, не все відбувалося гладко на цьому шляху. Так, наприклад, виявилося неможливим вивести з чисто логічних аксіом існування нескінченних множин. Відкриття парадоксів, пов'язаних з основними поняттями теорії множин, ще більше похитнуло впевненість математиків в досягненні поставленої мети. Основна проблема полягала в доказі несуперечності обраної системи аксіом. Д.Гільберт поставив перед собою завдання розвитку теорії доказів. Заслуга Гільберта полягає в тому, що він вказав шлях для дослідження несуперечності аксіоматичних систем. Несуперечливість теорії означає, що в ній не можна вивести протиріччя. Тоді для доказу несуперечності формальної теорії потрібно довести невиводимість в ній будь-яких тверджень. Таким чином математична теорія, несуперечливість якої потрібно довести, стає об'єктом вивчення іншої математичної науки, яку Гільберт назвав метаматематикою або теорією доказів. Двотомна монографія Д. Гільберта і П. Бернайса «Підстави математики. Логічні обчислення і формалізація арифметики», що вийшла в 30-х р.р., підвела підсумок процесу становлення математичної логіки як самостійної математичної дисципліни.

У 1930 р австрійський математик Курт Гедель довів теорему про повноту числення предикатів, згідно з якою множина логічно загальнозначущих формул логіки предикатів збігається з множиною теорем числення предикатів. Однак уже в 1931 р з'явилася інша робота Геделя: «Про формально нерозв'язні вирази Principia Mathematica і споріднені системи». Доведена в ній теорема про неповноту формальної арифметики, визнана одним з найбільших досягнень сучасної математики. Геделем в цей час було 25 років. Відповідно до цієї теореми, якщо формальна система, що містить арифметику, несуперечлива, то в ній існують істинні, але не виведені з аксіом цієї теорії вирази. Звідси випливає, що формальна теорія, засобами якої можна побудувати арифметику, істотно неповна, тобто жодного розширення її системи аксіом не зробить її повною, так як в новій системі все одно знайдуться істинні, але не виведені її засобами вирази. Інший результат Геделя полягає в тому, що не можна довести несуперечність формальної теорії першого порядку, що включає формальну арифметику, засобами цієї теорії. Для доказу несуперечності необхідне залучення засобів, що виходять за рамки формальної арифметики. Такі докази несуперечності арифметики були отримані пізніше Г. Генценом і П.С. Новіковим. Результати, отримані Геделем, показали, що можливості аксіоматичного методу певним чином обмежені, причому обмеження такі,

що навіть арифметика натуральних чисел не може бути повністю аксіоматизована. Відкриття Геделя зруйнували надії логіків про повну формалізації математики. Однак роботи Геделя збагатили математичну науку абсолютно новими методами міркувань і мали величезне значення для розвитку філософії науки. Результати Геделя показують, що можливості нашого мислення не зводяться до повністю формалізованих процедур дедуктивних міркувань. Людське мислення і способи людських міркувань значно багатші, і для їх формалізації (навіть часткової) необхідне залучення засобів, що виходять за рамки дедуктивних міркувань.

4.1. Примітивно рекурсивні функції

Визначення 4.1. *Арифметичними функціями* називаються функції, у яких область визначення і множина значень складаються з натуральних чисел, а *арифметичним відношенням* називаються відношення, задані на множині натуральних чисел.

Так, наприклад, множення є арифметичною функцією з двома аргументами, а вираз $x + y < z$ визначає арифметичне відношення з трьома аргументами.

Визначення 4.2.

1. Наступні функції називаються *вихідними функціями.*

(*I*) **Нуль-функція**: $Z(x) = 0$ для кожного x.

(*II*) **Додаток одиниці** (*наступний за*): $N(x) = x + 1$ для кожного x.

(*III*) **Проектуючі функції**: $U_i^n(x_1, ..., x_n) = x_i$ при всіх $x_1, .., x_n$ ($i = 1, ..., n$; $n = 1, 2, ...$).

2. Наступні два правила: підстановка і примітивна рекурсія, – служать для отримання нових функцій, виходячи з уже наявних.

(*IV*) **Підстановка** . Кажуть, що функція f отримана за допомогою підстановки з функцій $g(y_1, .., y_m)$, $h_1(x_1, ..., x_n)$, ..., $h_m(x_1, ..., x_n)$, якщо $f(x_1, ..., x_n) = g(h_1(x_1, ..., x_n), ..., h_m(x_1, ..., x_n))$.

(*V*) **Схема примітивної рекурсії.** Функція f отримана з функцій g і h за допомогою рекурсії, якщо

$f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$,

$f(x_1, ..., x_n, y + 1) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y))$.

При цьому виключається випадок $n = 0$, для якого окремо:

$f(0) = k$ (де k – фіксований ціле невід'ємне число),

$f(y + 1) = h(y, f(y))$.

Зауважимо, що функція f цілком визначена: значення $f(x_1, ..., x_n, 0)$ визначається з першої рівності, а якщо ми вже знаємо значення $f(x_1, ..., x_n, y)$, то з другого рівності ми можемо знайти значення $f(x_1, ..., x_n, y + 1)$.

3. Функція f називається *примітивно рекурсивної* , якщо вона може бути отримана з вихідних функцій за допомогою кінцевого числа підстановок (*IV*) і рекурсій (*V*), тобто якщо існує така кінцева послідовність функцій $f_1, ..., f_n$, що $f_n = f$ і для кожного i , $0 \le i \le n$, функція f_i або вихідна, або може бути отримана з деяких попередніх їй в цій послідовності функцій за допомогою застосування правила (*IV*) (підстановки) або правила (*V*) (рекурсії).

Теорема 4.1 . Наступні функції є примітивно рекурсивними.

(a) $x + y$ (додавання).

Доведення.

За правилом рекурсії (*V*):

$x + 0 = x$, тобто $f (x , 0) = U_1^1 (x) = x$.

$x + (y + 1) = N (x + y)$, тобто $f (x , y + 1) = N (f (x , y))$.

(b) $x \cdot y$ (множення).

Доведення.

$x \cdot 0 = 0$, тобто $g (x , 0) = Z (x)$.

$x \cdot (y + 1) = (x \cdot y) + x$, тобто $g (x , y + 1) = f (x, g (x , y))$, де f є функція додавання.

(c) x^y (піднесення до степеня).

Доведення.

$x^0 = 1$, $x^{y+1} = (x^y) \cdot x$.

(d) $\delta (x) = x - 1$, якщо $x > 0$ і $\delta(x) = 0$, якщо $x = 0$ (віднімання одиниці).

Доведення.

$\delta (0) = 0$, $\delta (y + 1) = y$.

(e) $x \div y = x - y$, якщо $x \ge y$ і $x \div y = 0$, якщо $x < y$ (обмежена різниця).

Доведення.

$x \div 0 = x$, $x \div (y + 1) = \delta (x \div y)$.

(f) $| x - y | = X - y$, якщо $x \ge y$, $| x - y | = y - x$, якщо $x < y$ (модуль різниці).

Доведення.

$| x - y | = (x \div y) + (y \div x)$ (підстановка).

(g) $sg (x) = 0$, якщо $x = 0$, $sg (x) = 1$, якщо $x \ne 0$ (Сігнум x).

Доведення.

$sg (0) = 0$, $sg (y + 1) = 1$.

(h) $unsg\ (x) = 1$, якщо $x = 0$, $unsg\ (x) = 0$, якщо $x \neq 0$;

Доведення.

$unsg\ (x) = 1 \div sg\ (x)$.

$(I)\ x!$.

Доведення.

$0! = 1$, $(y + 1)! = (Y!) \cdot (y + 1)$.

(j) $min\ (x, y)$ = *найменшому з чисел* x *і* y ;

Доведення.

$min\ (x, y) = x \div (x \div y)$.

(k) $min\ (x_1, ..., x_n)$.

Доведення.

Припустимо, що функція $min\ (x_1, ..., x_n)$ – примітивно рекурсивна. Для $min\ (x_1, ..., x_{n+1})$ маємо $min\ (x_1, ..., x_{n+1}) = min\ (min\ (x_1, ..., x_n), x_{n+1})$.

(l) $max\ (x, y)$ = *найбільшому з чисел* x *і* y .

Доведення.

$max\ (x, y) = y + (x \div y)$.

(m) $max\ (x_1, ..., x_n)$;

Доведення.

$max\ (x_1, ..., x_{n+1}) = max\ (max\ (x_1, ..., x_n), x_{n+1})$.

(n) $rm\ (x, y)$ = *залишку від ділення* y *на* x, *якщо* $x \neq 0$, *і* y , *якщо* $x = 0$.

Доведення.

$rm\ (x, 0) = 0$, $rm\ (x, y + 1) = N\ (rm\ (x, y)) \cdot sg\ (|x - N\ (rm\ (x, y))|)$.

(o) $qt\ (x, y)$ = *частці від ділення* y *на* x .

Доведення.

$qt\ (x, 0) = 0$, $qt\ (x, y + 1) = qt\ (x, y) + unsg\ (|x - N\ (rm\ (x, y))|)$.

4.2. Рекурсивні функції

Введемо ще одне правило отримання нових функцій (до визначення 4.2).

(*VI*) **μ-оператор (оператор мінімізації)**. Нехай функція $g\ (x_1, ..., x_n, y)$ така, що для будь-яких $x_1, .. ., x_n$ існує по крайній мірі одне значення y, при якому $g\ (x_1, ..., x_n, y) = 0$. Позначимо через $\mu y\ (g\ (x_1, ..., x_n, y) = 0)$ найменше

значення y, при якому $g(x_1, ..., x_n, y) = 0$. Тоді *функція f отримана з функції g з допомогою* μ-*оператора*, якщо $f(x_1, ..., x_n) = \mu y (g(x_1, ..., x_n, y) = 0)$.

При застосуванні μ-оператора можна використовувати не тільки відношення рівності, а й інші: $<$, $\leq$, $>$, $\geq$. Взагалі, для будь-якого відношення $R(x_1, ..., x_n, y)$ будемо позначати через $\mu y R(x_1, ..., x_n, y)$ то найменше значення y, при якому $R(x_1, . .., x_n, y)$ істинно, якщо взагалі такі значення існують.

4. Функція f називається *рекурсивною*, якщо вона може бути отримана з вихідних функцій за допомогою кінцевого числа застосувань підстановки (*IV*), рекурсії (*V*) і μ-оператора (*VI*).

Це останнє визначення відрізняється від визначення примітивно рекурсивної функції лише додатковим дозволом застосовувати μ-оператор. Тому будь-яка примітивно рекурсивна функція є також і рекурсивної функцією. Зворотне, втім, не завжди вірно. (В літературі замість терміна «рекурсивний» іноді вживається термін «частково рекурсивний»).

Визначення 4.3. *Обмежений μ-оператор* визначимо так:
$\mu y_{y<z} R(x_1, ..., x_n, y)$ дорівнює найменшому y такому, що $y<z$ і $R(x_1, ..., x_n, y)$ істинно, якщо таке y існує, і z в іншому випадку.

Необмежений μ-оператор (визначений у (*VI*)) задає наступну схему обчислень n - місцевої функції $f(x_1, ..., x_n)$. Для фіксованих значень аргументів $x_1,...,x_n$ знаходиться рішення рівняння $g(x_1,...,x_n, y) = x_{n+1}$ для $y=0$, 1, 2, ... Найменше значення $y = b$, для якого $g(x_1, ..., x_n, b) = x_{n+1}$ і є значення функції $f(x_1, ..., x_n) = \mu y (g(x_1, ..., x_n, y) = x_{n+1})$, отриманої з функції g з допомогою μ-оператора.

Цей процес знаходження значення функції f буде тривати нескінченно, якщо: (1) значення $g(x_1, ..., x_n, 0)$ не визначене; (2) значення $g(x_1, ..., x_n, y)$ визначені для $y = 0, 1, ..., b - 1$, але відмінні від x_{n+1}, а $g(x_1, ..., x_n, b)$ не визначене; (3) значення $g(x_1, ..., x_n, y)$ визначені для всіх y, але відмінні від x_{n+1}. У всіх цих випадках обчислення тривають нескінченно і значення функції $f(x_1, ..., x_n)$ вважається невизначеним.

Таким чином, застосування μ-оператора дозволяє отримати частково певні функції, що і пояснює термін «частково рекурсивні» функції. Застосування обмеженого μ-оператора дозволяє зупинити процес обчислень, ввівши верхню межу перебору $y < z$, і довизначити функцію $f(x_1, ..., x_n)$, присвоївши їй деяке довільне значення, наприклад, z.

μ-оператор є зручним засобом для побудови зворотних функцій. Якщо задана функція g одномісна, то $f(x)=g^{-1}(x)=\mu y(g(y)=x)$. Для багатомісних функцій запис g^{-1} не вживається.

Приклади .

1. Для функції $sg(x)$ зворотну функцію можна визначити як $sg^{-1}(x) = \mu x\,(sg(x)=1)$. Тоді $sg^{-1}(x)=x$, якщо $x= 0, 1$; і $sg^{-1}(x)$ не визначене, якщо $x > 1$.

2. Визначимо функцію $y = [\, z / x \,]$ (частка від ділення z на x) як функцію, зворотну множенню: $z = y \cdot x$. Тоді $y = \mu y_{\,y \le z}(|\, y \cdot x - z \,| = 0)$, тобто це буде найменше значення y, при якому існує рішення рівняння $|\, y \cdot x - z \,| = 0$. Функція в повному обсязі не визначена, тому що не кожні два числа діляться один на одного без залишку.

3. Функція $y = [\, \sqrt{x}\,]$ (ціла частина $\sqrt{x}$) може бути визначена за допомогою обмеженого μ-оператора як $y = \mu y_{\,y \le x}(\ ^2 = x\ )$. Ця функція також є частково визначною функцією.

4.3. Рекурсивні відношення

Визначення 4.4. *Характеристичної функцією* відношення $R\,(\,x_1, ..., x_n\,)$ називається функція $C_R\,(\,x_1, ..., x_n\,)$, що задається умовами:

$$C_R(x_1, ..., x_n) = \begin{cases} 1, \text{ якщо } R(x_1, ..., x_n) = T. \\ 0, \text{ якщо } R(x_1, ..., x_n) = F. \end{cases}$$

Визначення 4.5. Відношення $R\,(\,x_1, ..., x_n\,)$ називається *примітивно рекурсивним* (*рекурсивним*) відношенням, якщо примітивно рекурсивною (відповідно рекурсивною) є його характеристична функція $C_R\,(\,x_1, ..., x_n\,)$.

Оскільки кожному відношенню можна поставити у відповідність предикат, це визначення є також визначенням примітивно рекурсивного (рекурсивного) предиката.

Приклади.

1. Відношення $x_1 = x_2$ примітивно рекурсивно, так як характеристична функція його збігається з функцією $unsg\,(|\, x_1 - x_2 \,|)$, яка примітивно рекурсивна, в силу тез (f), (g) теореми 4.1.

2. Примітивно рекурсивна функція $unsg\,(x_1 \div x_2)$ служить характеристичної функцією відношення $x_1 < x_2$, яке, таким чином, примітивно рекурсивно.

3. Відношення $x_1 \vdots x_2$ (x_1 ділиться на x_2 без залишку) примітивно рекурсивно, так як його характеристичної функцією є примітивно рекурсивна функція $unsg\,(rm\,(x_1, x_2))$.

4. Відношення $Pr\,(x)$, тобто «X є просте число», примітивно рекурсивно, так як $C_{Pr}(x) = unsg\,((D(x) \div 2) + unsg\,(|\, x - 1 \,| + unsg\,(|\, x - 0 \,|))$, де функція $D\,(\,x\,)$ дорівнює 1, якщо $x = 0$, і числу дільників x, якщо $x > 0$.

Функція $D(x)$ примітивно рекурсивна, так як *(сума по x від 1 до y)* $D(x)=\sum_{x=1}^{y} unsg(rm(y,x))$.

Теорема 4.2. Відношення, які можна отримати з примітивно рекурсивних (або рекурсивних) за допомогою пропозиційних зв'язок і обмежених кванторів, також примітивно рекурсивні (відповідно, рекурсивні); застосування обмежених μ-операторів $\mu y_{\ y < z}$ або $\mu y_{\ y \le z}$ до примітивно рекурсивних (рекурсивних) відношень призводить до примітивно рекурсивних (рекурсивних) функцій.

Доведення.

Нехай відношення $P(x_1, \ldots, x_n)$ і $Q(x_1, \ldots, x_n)$ примітивно рекурсивні (рекурсивні). Це означає, що примітивно рекурсивними (рекурсивними) є їхні характеристичні функції C_P і C_Q. Тоді примітивно рекурсивними (рекурсивними) є і відношення: $\neg P\ (x_1, \ldots, x_n)$ (його характеристична функція: $C_{\neg P} = 1 \div C_P$), $P(x_1, \ldots, x_n) \vee Q(x_1, \ldots, x_n)$ (характеристична функція: $C_{P \vee Q} = sg\ (C_P + C_Q)$), $P(x_1, \ldots, x_n)\ \&\ Q(x_1, \ldots, x_n)$ (характеристична функція: $C_{P\ \&\ Q} = C_P \cdot C_Q$). Звідси неважко отримати характеристичні функції для формул: $R1 = \forall y_{\ y < z} P(x_1, \ldots, x_n, y)$ і $R2 = \exists y_{\ y < z} P(x_1, \ldots, x_n, y)$ із допомогою обмеженої суми і обмеженого добутку: $C_{R2} = sg\ (\sum_{y < z} C_P(x_1, \ldots, x_n, y))$ і $C_{R1} = \Pi_{y < z} C_P(x_1, \ldots, x_n, y)$[7].

Отже, термін **рекурсивні функції** в теорії обчислюваності використовують для позначення трьох множин функцій

- примітивно рекурсивні функції;
- загальнорекурсивні функції;
- частково рекурсивні функції.

Множина частково рекурсивних функцій включає в себе множину загальнорекурсивних функцій, а загальнорекурсивні функції включають в себе примітивно рекурсивні функції. Частково рекурсивні функції іноді називають просто рекурсивними функціями.

Визначення поняття ***примітивно рекурсивної функції*** *складається з вказівки класу базових примітивно рекурсивних функцій і двох операторів (*підстановки *і* примітивної рекурсії*), що дозволяють будувати нові примітивно рекурсивні функції на основі вже наявних.*

До числа базових примітивно рекурсивних функцій відносяться функції наступних трьох видів:

- *Нульова функція Z* одного змінного, що зіставляє кожному натуральному числу значення 0.

---

[7] Оскільки квантор загальності - узагальнення кон'юнкції, а квантор існування - узагальнення операції диз'юнкції, для отримання нових характеристичних функцій можна скористатися операторами максимуму і мінімуму.

- *Функція наступного N* одного змінного, що зіставляє кожному натуральному числу *x* безпосередньо наступне за ним натуральне число *x* + 1.
- Функції U^m_n, де $0<m\leq n$, від n змінних, що зіставляють будь-якого впорядкованого набору $x_1, \ldots, x_n$ натуральних чисел число x_m з цього набору.
- **Оператор підстановки** . Нехай *g* - функція від m змінних, а $h_1, \ldots, h_m$ - упорядкований набір функцій від *n* змінних кожна. Тоді результатом підстановки функцій h_k в функцію *g* називається функція *f* від *n* змінних, що зіставляє кожному впорядкованому набору натуральних чисел число

  $$f(x_1, x_2, \ldots, x_n) = g(h_1(x_1, x_2, \ldots, x_n), \ldots, h_m(x_1, x_2, \ldots, x_n)).$$
- **Оператор примітивної рекурсії** . Нехай *g* - функція від n змінних, а *h* - функція від *n* + 2 змінних. Тоді результатом застосування оператора примітивної рекурсії до пари функцій *h* і *g* називається функція *f* від *n* + 1 змінної виду

  $$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n),$$
  $$f(x_1, \ldots, x_n, y + 1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)).$$

Множина **примітивно рекурсивних функцій** - це мінімальна множина, що містить всі базові функції і замкнута щодо зазначених операторів підстановки і примітивної рекурсії.

Приклади

Ще раз покажемо кілька широко відомих арифметичних функцій, які є примітивно рекурсивними.

- Функція *Додавання* двох натуральних чисел (***Sum(a, b)=a+b***) може бути розглянута як примітивно рекурсивної функції двох змінних, одержуваної в результаті застосування оператора примітивної рекурсії до функцій U^1_1 і F, друга з яких виходить підстановкою основної функції U^3_3 в основну функцію *N*:

  $$Sum(x, 0)=x;$$
  $$Sum(x, y+1)=F(x, y, Sum(x, y));$$
  $$F(x, y, z) = N(U^3_3(x, y, z)).$$
- *Множення* двох натуральних чисел (***Mul(a, b)=a***x***b***) може бути розглянуто як примітивно рекурсивна функція двох змінних, одержана в результаті застосування оператора примітивної рекурсії до функцій *Z* і *G*, друга з яких виходить підстановкою основних функцій U^1_3 і U^3_3 в функцію додавання:

  $$Mul(x, 0) = Z(x); Mul(x, y+1)=G(x, y, Mul)x, y));$$
  $$G(x, y, z) = Sum(U^1_3(x, y, z), U^3_3(x, y, z)).$$

Частково рекурсивні функції *визначаються аналогічним чином, тільки до двох операторів підстановки і примітивної рекурсії додається ще оператор мінімізації аргументу.*

- **Оператор мінімізації аргументу**. Нехай *f* - функція від n натуральних змінних. Тоді результатом застосування оператора мінімуму аргументу до функції *f* називається функція *h* від *n* - 1 змінної, що задається наступним визначенням:

$$h(x_1, \ldots, x_{n-1})=min(y), \text{ за умови } f(x_1, \ldots, x_{n-1}, y)=0.$$

Тобто функція *h* повертає мінімальне значення останнього аргументу функції *f* , при якому її значення дорівнює 0.

Цікаво відзначити аналогію з імперативними мовами програмування. Примітивні функції відповідають програмним функціям, в яких використовується тільки арифметичні операції, а також умовний оператор і оператор арифметичного циклу (оператор циклу, в якому число ітерацій відомо на момент початку циклу).

Якщо ж програміст починає використовувати оператор циклу *while*, в якому число ітерацій заздалегідь невідомо, і в принципі, може бути нескінченним, то він переходить в клас частково рекурсивних функцій.

Важливим моментом є те, що частково рекурсивні функції для деяких значень аргументу можуть бути не визначені, тому що оператор мінімізації аргументу не завжди коректно визначається, зокрема, функція f може не дорівнювати нулю ні при яких значеннях аргументів.

Власне, тому, що частково рекурсивні функції можуть мати значення лише на частині аргументів, і пішла їх назва.

З точки зору програміста результатом частково рекурсивної функції може бути не тільки число, а й виключення (Exception) або «зависання», відповідне невизначеному значенню.

Загальнорекурсивні функції - це підмножина частково рекурсивних функцій, визначених для всіх значень аргументів. Проблема визначення того, чи є частково рекурсивна функція з даними описом загальнорекурсивною чи ні, алгоритмічно нерозв'язна .

Легко зрозуміти, що будь-яка примітивно рекурсивна функція є частково рекурсивної, так як за визначенням оператори для побудови частково рекурсивних функцій включають в себе оператори для побудови примітивно рекурсивних функцій.

Також зрозуміло, що примітивно рекурсивна функція визначена всюди і тому є загальнорекурсивною функцією (у примітивно рекурсивною функції немає приводу «зависати», так як при її побудові використовуються оператори, що визначають всюди існуючі функції).

4.4. Геделева нумерація

Гедель запропонував спосіб кодування символів і виразів будь-якої формальної теорії так, що кожному символу, виразу і послідовності виразів однозначно відповідає деяке натуральне число. Це спосіб кодування отримав назву «геделевої нумерації».

Курт Фрідріх Гедель (нім. *Kurt Friedrich Gödel*, 1906 - 1978) – австрійський логік, математик і філософ математики, найбільш відомий через сформульовану і доведену їм теорему про неповноту.

Біографія:

Курт Гедель народився 28 квітня 1906 г . в австро-угорському (моравському) місті Брюнн (нині Брно , Чехія), в німецькій родині. У 18 років Гедель вступив до Віденського університету . Там він два роки вивчав фізику, але потім переключився на математику.

Зазвичай Геделя вважають австрійцем, але за своє життя він неодноразово змінював громадянство. Народжений підданим Австро-Угорщини, він в 12 років прийняв громадянство Чехословаччини після того, як Австро-Угорська імперія припинила своє існування. У 23 роки Гедель став громадянином Австрії, а в 32 роки, після захоплення Австрії Гітлером автоматично став підданим німецького Рейху, порядки якого були йому абсолютно неприйнятні. У 1940 він їде в США, причому через небезпеку шляху через Атлантику під час війни він їде через СРСР і Японію. У США він отримує роботу в знаменитому Інституті перспективних досліджень (Institute for Advanced Study) в Прінстоні.

Наприкінці життя у Геделя розвинувся психічний розлад - параноїдальний страх отруєння. Він брав їжу тільки з рук дружини, а після її смерті в 1977 р. відмовився від їжі. Вчений помер від недоїдання 14 січня 1978 р. в Прінстоні, штат Нью-Джерсі.

Наукова спадщина

Гедель був логіком і філософом науки. Найбільш відоме досягнення Геделя - це сформульовані і доведені ним теореми про неповноту, опубліковані в 1931 р.. Одна з них свідчить, що будь-який засіб (мова), досить сильний для визначення натуральних чисел, наприклад, логіка другого порядку (*логіка другого порядку* - розширює логіку першого порядку, дозволяючи проводити квантифікацію загальності і існування не тільки над атомами, але і над предикатами; логіка другого порядку не спрощується до логіки першого порядку) або природня мова, є неповними. Тобто містять висловлювання, які не можна ні довести, ні спростувати, виходячи з аксіом мови. Доведені Геделем теореми мають широкі наслідки як для математики, так і для філософії (зокрема, для онтології і філософії науки).

Крім того Геделю належать роботи в області диференціальної геометрії і теоретичної фізики. Зокрема він написав роботу по загальній теорії відносності в якій запропонував варіант вирішення рівнянь Ейнштейна з якого випливає, що будова всесвіту може мати такий устрій, в якому перебіг часу є закільцьованим (метрика Геделя), що теоретично допускає подорожі в часі. Більшість сучасних фізиків вважають це рішення правильним лише математично і таким, що не має фізичного сенсу.

Перша теорема Геделя про неповноту (1931 р)

У всякій досить багатій несуперечливій теорії першого порядку (зокрема, у всякій несуперечливій теорії, що включає формальну арифметику), існує така замкнута формула F , що ні F , ні $\neg F$ не є виведеними в цій теорії.

Інакше кажучи, в будь-який досить складній несуперечливій теорії існує твердження, яке засобами самої теорії неможливо ні довести, ні спростувати. Наприклад, таке твердження можна додати до системи аксіом, залишивши її несуперечливої. При цьому для нової теорії (зі збільшеною кількістю аксіом) також буде існувати недовідне і незаперечне твердження.

Друга теорема Геделя про неповноту

У всякій досить багатій несуперечливій теорії першого порядку (зокрема, у всякій несуперечливій теорії, що включає формальну арифметику), формула, яка стверджує несуперечність цієї теорії, не є виведеної в ній.

Іншими словами, несуперечливість досить багатою теорії не може бути доведена засобами цієї теорії. Однак цілком може виявитися, що несуперечливість однієї конкретної теорії може бути встановлена засобами іншої, більш потужної формальної теорії. Але тоді постає питання про несуперечності цієї другої теорії, і т.п.

Геделева нумерація

Кожному символу u довільної теорії першого порядку K наступним чином поставимо у відповідність позитивне число $g(u)$, зване *геделевим номером* символу u:

g (() = 3;
g ()) = 5;
g (,) = 7;
g ($\neg$) = 9;
g ($\rightarrow$) = 11;
g (x_k) = 5 + 8 k для k = 1, 2, ...;
g (a_k) = 7 + 8 k для k = 1, 2, ...;
g (f_k^n) = 9 + 8 ($2^n 3^k$) для k , $n \geq 1$;
g (A_k^n) = 11 + 8 ($2^n 3^k$) для k , $n \geq 1$;
при цьому покладемо g ($\forall x_i$) = g ((x_i)).

Таким чином, різним символам поставлені у відповідність різні геделеві номера, що є натуральними числами. Наприклад, $g(x_2)$=21, $g(a_4)$=39, $g(f_1^2)$=105, $g(A_2^1)$=155.

Нехай дано вираз $u_0 u_1 ... u_r$, що представляє, наприклад, формулу теорії першого порядку. Геделевим номером $g(u_0 u_1 ... u_r)$ цього виразу визначимо як $2^{g(u0)} 3^{g(u1)} ... p_r^{g(ur)}$, де p_i є i -е просте число і p_0 =2.

Наприклад, $g(A_1 \rightarrow (A_2 \rightarrow A_1)) = 2^{g(A1)} 3^{g(\rightarrow)} 5^{g(()} 7^{g(A2)} 11^{g(\rightarrow)} 13^{g(A1)} 17^{g())}$.

Зауважимо, що, в силу єдиності розкладання натуральних чисел в добутки степенів простих чисел, різні вирази отримують при цьому різні геделеві номери. Крім того, геделеві номери виразів є парними числами, і тому відмінні від геделевих номерів символів.

Нарешті, геделев номер довільної послідовності e_0, ..., e_r виразів (наприклад, виведення формули e_r в теорії K) визначимо наступним чином: $g(e_0, ..., e_r) = 2^{g(e0)} ... 3^{g(e1)} ... p_r^{g(er)}$. Як і раніше, різні послідовності виразів мають різні геделеві номери, а так як ці останні парні і, крім того, мають парний показник ступеня при 2, то вони відмінні і від геделевих номерів символів, і від геделевих номерів виразів.

Таким чином, функція g взаємно однозначно відображає множину всіх символів, виразів і кінцевих послідовностей виразів в множину цілих додатних чисел.

Множина значень функції g не збігається, проте, з множиною всіх натуральних чисел, так як не всі натуральні числа є геделевими номерами, наприклад, число 12 не є геделевим номером.

Така нумерація символів, виразів і послідовностей виразів була запропонована Геделем в 1931 р. з метою *арифметизації* математики, тобто з метою заміни тверджень про формальну систему еквівалентними висловлюваннями про натуральні числа з наступним уявленням цих висловлювань у формальній системі. Оскільки кожному виразу обчислення приписаний Геделем номер, то кожне метаматематичне висловлювання про вирази обчислення і відношення між ними можна виразити як висловлювання про відповідні геделеви номери і відношення між ними. Таким чином метаматематика виявляється повністю «арифметизованою», і мова арифметики стає мовою метаматематики. Вивчення метаматематичних питань зводиться до вивчення співвідношень і властивостей деяких чисел.

Теорема Геделя про неповноту

Трохи доповнивши мову логіки першого порядку для забезпечення можливості застосовувати схему математичної індукції в арифметиці, Гедель зумів показати в своїй теоремі про неповноту, що існують справжні арифметичні висловлювання, які не можуть бути доведені.

Повне доведення цієї теореми про неповноту в своєму безпосередньому вигляді займає 30 сторінок, але тут наведемо його схематично. Почнемо з логічної теорії чисел. У цій теорії існує єдина константа, 0, і єдина функція, N (функція визначення наступного). У наміченої моделі інтерпретації цієї теорії N(0) позначає 1, N(N (0)) позначає 2 і т. д., тому в даному описі мовою є імена для всіх натуральних чисел. Крім того, словник мови включає функціональні символи +, X і Expt (піднесення до степеня), а також звичайну множину логічних зв'язок і кванторів. Перш за все, слід зазначити, що множина висловлювань, які можуть бути записані на цій мові, може бути пронумерована. (Для цього достатньо уявити собі, що визначений алфавітний порядок символів, а потім в алфавітному порядку розташовано кожне з множин висловлювань з довжиною 1, 2 і т. д.) Потім можна позначити кожне висловлювання α унікальним натуральним числом # α (яке називається геделевим номером). Це - найважливіший момент доведення; в ньому стверджується, що теорія чисел включає окреме ім'я для кожного з її власних

висловлювань. Аналогічним чином, за допомогою геделева номера g(P) можна пронумерувати кожне можливе доведення P, оскільки будь-яке доведення - це просто кінцева послідовність висловлювань.

Тепер припустимо, що є рекурсивно перелічувана множина A висловлювань, які представляють собою істинні твердження про натуральні числа. Нагадаємо, що висловлювання з множини A можна назвати за допомогою заданої множини цілих чисел, тому можна уявити собі, що на нашій мові записується висловлювання α(j, A) такого роду:

∀i i – ні геделевий номер доведення висловлювання, геделевим номером якого є j, де в доведенні використовуються тільки передумови з множини A.

Потім припустимо, що σ є висловлювання α (# σ, A), тобто висловлювання, в якому стверджується його власна недовідність з A. Твердження про те, що таке висловлювання завжди існує, є істинним, але його не можна назвати повністю очевидним[8].

Тепер застосуємо наступний дотепний аргумент: припустимо, що висловлювання σ доказово з A; в такому випадку висловлювання σ помилково (оскільки в висловлюванні σ стверджується, що воно не може бути доведено). Але це означає, що є деяке хибне висловлювання, яке доказово з A, тому A не може складатися тільки з істинних висловлювань, а це суперечить нашій передумові. Тому висловлювання σ не доказовою з A. Але саме це і стверджує про самого себе висловлювання σ, а це означає, що σ – справжнє висловлювання.

Отже, ми довели (і заощадили 29 з половиною сторінок), що для будь-якої множини дійсних висловлювань теорії чисел і, зокрема, для будь-якої множини базових аксіом, існують інші істинні висловлювання, які не можуть бути доведені з цих аксіом. Очевидно, що це відкриття мало для математики дуже важливе значення. Значимість цього відкриття для штучного інтелекту була предметом широких обговорень, починаючи з роздумів самого Геделя.

Питання до розділу 4

1. Арифметичні функції.
2. Схема примітивної рекурсії. Примітивно-рекурсивні функції.
3. Оператор мінімізації, рекурсивні функції.
4. Рекурсивні відношення.
5. Геделева нумерація. Теорема Геделя про повноту.

[8] «Це висловлювання - хибно»

5. ТЕОРІЯ АЛГОРИТМІВ

5.1. Інтуїтивне поняття алгоритму

Одним з найбільш істотних факторів, що визначають значення математики і місце її в сучасній науці, є точність і висока ступінь універсальності образотворчих засобів, якими вона володіє. Ці засоби дозволяють будувати уточнені моделі різноманітних за своєю природою фундаментальних наукових понять, в тому числі і математичних.

Розробка такого роду уточненої моделі була проведена у другій половині 30-х років минулого століття до одного з найважливіших і вживаними понять математики – поняттю "алгоритма" (тобто, алгоритм, метод, спосіб).

Функція $f(x_1, x_2, ..., x_n)$ називається ефективно обчислюваною, якщо для кожного набору аргументів $a_1, a_2, ..., a_n$ з її області визначення може бути обчислено значення функції $f(a_1, a_2, ..., a_n)$. Якщо функція ефективно обчислювана, то кажуть, що існує *алгоритм* її обчислення.

Поняття алгоритму інтуїтивно зрозуміло і часто використовується в математиці. Додавання і множення чисел "стовпчиком", яке відоме ще зі школи, формула знаходження коренів квадратного рівняння, множення двох матриць за правилом "рядок на стовпець" є алгоритмами. Безліч прикладів можна продовжити. У сучасній практиці під алгоритмом часто розуміють програму, а критерієм існування алгоритму вважають можливість його запрограмувати. Однак це не зовсім так. Спочатку дамо *поняття* алгоритму.

Під алгоритмом розуміють точне розпорядження про виконання в певному порядку точно зазначеної послідовності операцій для вирішення всіх завдань з даного класу задач.

Наведене вище поняття алгоритму не є чітким математичним визначенням. Це поняття характеризується набором властивостей, притаманних алгоритму.

Властивості алгоритму

- Дискретність інформації. Кожен алгоритм має справу з даними - вхідними, проміжними і вихідними. Ці дані подаються у вигляді кінцевих слів в деякому алфавіті.
- Дискретність роботи алгоритму. Алгоритм виконується крок за кроком і при цьому на кожному кроці виконується тільки одна операція.
- Здійснимість операцій. В алгоритмі не повинно бути нездійсненних операцій. Наприклад, не можна в програмі присвоїть змінній значення "нескінченність", - така операція була б нездійсненною. Кожна операція обробляє певну ділянку в оброблюваному слові.
- Скінченність алгоритму. Скінченність алгоритму означає, що опис алгоритму має бути кінцевим. Крім цього, робота самого алгоритму

також повинна бути кінцева, тобто після кінцевого числа кроків алгоритм повинен закінчувати свою роботу. Для програм, зокрема це означає, що алгоритм не повинен "зациклюватися". Робота алгоритму завершується видачею результату. Тому скінченність алгоритму тісно пов'язана з його результативністю: здатністю видавати результат.

- Детермінованість алгоритму. Кожен крок алгоритму строго визначений. Після кожного кроку точно вказується, який крок зробити далі, або вказується, що алгоритм повинен закінчити свою роботу на даному етапі. До кожної ділянки слова на кожному кроці може бути застосована тільки одна операція.
- Масовість алгоритму. Масовість алгоритму означає, що алгоритм повинен вирішувати всі завдання з даного класу задач. Якщо знайдеться хоча б одна задача, для якої алгоритм не застосуємо, то цю послідовність операцій не можна вважати алгоритмом. Так, наприклад, метод резолюції не є алгоритмом доказу загальнозначущості формул логіки предикатів. Існують завдання, для яких метод резолюцій не може дати однозначної відповіді, наприклад, чи повинен відомий нам цирульник голити самого себе.

5.2. Алфавіти і слова

Визначення 5.1. Будемо називати довільну кінцеву множину *алфавітом A*, а елементи цієї множини *буквами* алфавіту *A*. Тоді довільні кінцеві послідовності букв називаються *словами* в даному алфавіті.

Будь-який алфавіт містить порожній символ. Будемо позначати його символом Λ. Порожній символ є і порожнім словом.

Множину слів алфавіту *A* будемо позначати *A**.

Визначення 5.2. Кількість букв в слові називається його *довжиною* і позначається вертикальними дужками. Наприклад, довжина слова *аааа* дорівнює | *аааа* | = 4, довжина порожнього слова | Λ | = 0.

Основною операцією над словами є *конкатенація.*

Визначення 5.3. *Конкатенацію* двох слів *P* і *Q* назвемо слово, отримане дописуванням слова *Q* після *P*: *PQ* .

Наприклад, конкатенація слів *aab* і *ba* є слово *aabba*. Операція конкатенації некомутативна, але асоціативна.

Визначення 5.4. Якщо *x* - деяке слово алфавіту *A* і якщо *x* представимо у вигляді конкатенації *PQ* , то *P* і *Q* називають *підсловами* слова *x*.

Наприклад, слово *abb* містить підслова *a, b, ab, bb.* Якщо $x \in A^*$ і $X = P_1 Q P_2 Q P_3$, то говорять про перше, друге і т.д. *входження* слова *Q* в слово *X*. Слова, складені з послідовності однакових букв, будемо іноді

записувати *aaabb* = $a^3 b^2$, де ступінь позначає кількість входжень літери в слово.

Кодування та нумерація

Визначення 5.5. Нехай дано два алфавіти *A* і *B*. Кодуванням слів алфавіту *A* словами в алфавіті *B* є функція $\varphi(p)$, яка здійснює відображення множини слів в алфавіті А в множину слів в алфавіті В: $\varphi(p)$: А* $\rightarrow B^*$, де $p \in A^*$, $\varphi(p) \in B^*$.

Якщо кожному слову в алфавіті *A* відповідає слово в алфавіті *B*, то відображення однозначне.

Приклад. Нехай *A* = { *a* , *b* }, *B* = { *a* , *b* , *c* }. Відображення $\varphi(p)$: $p \rightarrow cpc$ визначає кодування слів в алфавіті *A* словами в алфавіті *B*, наприклад, $ab \rightarrow cabc$.

Визначення 5.6. Кодування називається *блоковим,* якщо кожній букві алфавіту А відповідає слово в алфавіті В.

Приклад. Відображення $\varphi(a)$: $a \rightarrow ac$, $\varphi(b)$: $b \rightarrow bc$ є блоковим кодуванням. Наприклад, $\varphi(ab)=acbc$.

Будь-яку множину слів в деякому алфавіті можна впорядкувати в лексикографічному порядку: якщо $R_1 \alpha R_2$, $R_1 \beta R_2$ і $\alpha < \beta$, то $R_1 \alpha R_2 < R_1 \beta R_2$.

Приклад лексикографічного впорядкування для слів алфавіту *A* = { *a* , *b* }:

Λ , *a* , *b* , *aa* , *ab* , *ba* , *bb* , *aaa* , *aab* , *aba* , *abb* , *baa* , *bab* , *bba* , *bbb* , і т.д.

5.3. Проблема еквівалентності слів

Словникові примітивно рекурсивні функції

Вихідні функції для алфавіту *A:*

Нуль-функція $0(x) = \Lambda$;

Додавання символу ζ: $P_\zeta(x) = x\zeta$, де $\zeta \in A$;

Проектуюча функція $U_i^j(x) = x_i$, де *j* - кількість букв в слові *x* , x_i - *i-а* буква слова *x*.

Схема примітивної рекурсії для алфавіту A = { a, b).

Нехай $g(x_1, ..., x_n)$, $h_1(x_1, ..., x_{n+2})$, $h_2(x_1, ..., x_{n+2})$, - вихідні або примітивно рекурсивні функції. Тоді $f(x_1, ..., x_{n+1})$ - нова примітивно рекурсивна функція, якщо

$f(x_1, ..., \Lambda) = g(x_1, ..., x_n)$,

$f(x_1, ..., a) = h_1(x_1, ..., x_n, a, f(x_1, ..., x_{n+1}))$,

$f(x_1, ..., b) = h_1(x_1, ..., x_n, b, f(x_1, ..., x_{n+1}))$.

Позначимо операцію конкатенації $con(x_1, x_2)$, $x_1, x_2 \in A^*$, $A = \{a, b\}$ і покажемо, що вона є примітивно рекурсивної.

$con(x_1, \Lambda) = x_1 = U_1^2(x) = x_1$;

$con(x_1, x_2, a) = x_1x_2a = P_a(con(x_1x_2))$;

$con(x_1, x_2, b) = x_1x_2b = P_b(con(x_1x_2))$.

У загальному випадку

$con(x_1, x_2, \xi) = x_1x_2\xi = P_\xi(con(x_1x_2))$.

Асоціативні обчислення

Визначення 5.7. *Підстановка* слова P замість Q в слові W означає заміну Q на P в слові W. Підстановка позначається: $P \to Q$.

Приклад. Нехай дано алфавіт A = { a , b } і підстановка $ab \to ba$. Тоді результат виконання підстановки над словом *ababbbab* буде наступним:

1. *ab abbbab*
2. *ba ab bbab*
3. *b ab abbab*
4. *bba ab bab*
5. *b b ab abab*
6. *bbba ab ab*
7. *bbb ab aab*
8. *bbbbaa ab*
9. *bbbba ab a*
10. *bbbb ab aa*
11. *bbbbbaaa*

Визначення 5.8. Сукупність усіх слів алфавіту A разом з кінцевою системою допустимих підстановок називається *асоціативним обчисленням*.

Проблема еквівалентності слів

Визначення 5.9. Якщо слово R може бути перетворено в слово S за допомогою одноразового застосування допустимої підстановки, то, очевидно, що слово S може бути перетворено в слово R за допомогою застосування зворотної підстановки. Такі слова називаються *суміжними* словами.

Визначення 5.10. Послідовність суміжних слів R_1, R_2, ..., R_n називають *дедуктивним ланцюжком* від R_1 до R_n.

Визначення 5.11. Якщо існує дедуктивний ланцюжок від R до S, то існує і дедуктивний ланцюжок від S до R. Такі два слова називаються *еквівалентними* словами.

Проблема еквівалентності слів полягає в знаходженні дедуктивного ланцюжка від слова R до слова S. Це означає, що необхідно знайти таку множину допустимих підстановок, щоб, застосовуючи їх до слова R, в кінці дедуктивного ланцюжка отримати слово S.

Надалі ми покажемо, що проблема еквівалентності слів для будь-якого довільно взятого асоціативного обчислення алгоритмічно нерозв'язна.

Питання до розділу 5

1. Інтуїтивне поняття алгоритму, властивості алгоритму.
2. Алфавіти і слова. Асоціативні обчислення. Проблема еквівалентності слів.

6. АЛГОРИТМІЧНІ СХЕМИ

6.1. Машина Тюрінґа

Алан Матисон Тюрінґ (англ. Alan Mathison Turing, 23 червня 1912 - 7 червня 1954) - англійський математик, логік, криптограф, винахідник машини Тюрінґа.

Син британського чиновника в Індії, Алан вчився у Франції, Англії та в США. Тоді багато математиків намагалися створити алгоритм для визначення істинності висловлювань. Але Геделю вдалося довести, що будь-яка корисна математична система аксіом неповна в сенсі того, що в ній існує вислів, істинність якого не можна ні спростувати, ні підтвердити. Це спонукало Тюрінґа довести, що немає загального методу визначення істинності і, таким чином, математика завжди буде містити недоведені висловлювання.

У своїй роботі Тюрінґ запропонував проект простого пристрою, що має всі основні властивості сучасної інформаційної системи: програмне управління, пам'ять, і покроковий спосіб дій. Ця уявна машина, що отримала назву «машини Тюрінґа», використовується в теорії автоматів або комп'ютерів.

Коли Тюрінґ з США повернувся в Англію, почалася друга світова війна. Одним з найважливіших озброєнь цієї війни була ЕОМ «Колос» за проектом «Ультра», що почала в 1943 році зламувати надскладні шифри німців.

Після війни в 1945 Алан очолив проект створення комп'ютера «ТУЗ» (ACE, Automatic Computing Engine), а в 1948 Тюрінґ став працювати з «МАДАМ» (MADAM, Manchester Automatic DigitAl Machine) - комп'ютером з найбільшою пам'яттю в світі на той час. Роботи Алана зі спорудження перших ЕОМ і розвитку методів програмування мали неоціненну важливість, давши основу більшості досліджень в області штучного інтелекту. Він вважав, що комп'ютери, врешті-решт, зможуть мислити, як людина, і запропонував просту перевірку, відому як тест Тюрінґа, що оцінює здатність машини мислити: поговоріть з ЕОМ, і нехай вона переконає вас, що вона - людина.

Одна з щорічних нагород Асоціації обчислювальної техніки називається Премія Тюрінга. Премія заснована Асоціацією обчислювальної техніки в честь Алана Тюрінґа, як видатного вченого, який отримав перші глибокі результати щодо обчислюваності задовго до появи перших електронних обчислювальних машин.

Премія щорічно вручається одному або декільком фахівцям в галузі інформатики та обчислювальної техніки, чий внесок в цю область справив сильний та тривалий вплив на комп'ютерне співтовариство. Премія може бути присуджена одній людині не більше одного разу. У сфері інформаційних технологій премія Тюрінґа має статус, аналогічний Нобелівської премії в академічних науках. Вперше Премія Тюрінга була присуджена в 1966 році Алану Перліс за розвиток технології створення компіляторів.

У 2000-і роки преміальний фонд спонсорувався корпораціями Intel і Google, щорічний розмір премії становив $ 250 тис.. З 2014 року щорічний призовий фонд збільшено до $ 1 млн, а компанія Google стала єдиним спонсором премії.

За традицією, лауреат премії Тюрінґа при врученні її виступає з доповіддю, що має назву «Тюрінґівська лекція». У цій «лекції» зазвичай йде мова про тих питаннях комп'ютерної науки, теорії та практики використання обчислювальної техніки, які лауреат вважає досить важливими, щоб поділитися своєю думкою про них з якомога більшою кількістю фахівців.

Серед лауреатів премії були:

• Дональд Кнут (аналіз алгоритмів),

• Едсгер Дейкстра (теорія мов програмування),

• Хоар, Чарльз Ентоні Річард (теоретик мов програмування, винахідник алгоритму Quick Sort),
• Ніклаус Вірт (творець мови Паскаль) та багато інших видатних науковців.

6.1.1. Визначення машини Тюрінґа

Машина Тюрінґа була першою алгоритмічної схемою, запропонованою в якості математичного визначення алгоритму в 1937 році.

Машина Тюрінґа є гіпотетичною машиною: Тюрінґ не ставив завдання сконструювати цей пристрій. Концепція обчислювальної машини належить фон Нейманому, але її основою послужила машина Тюрінґа. Машина Тюрінґа є знакопереробним пристроєм. Завдання машини Тюрінґа - переробляти вхідне слово $W \in A^*$ в якесь вихідне слово $W^* \in A^*$, де A - зовнішній алфавіт.

Машина Тюрінґа складається з трьох частин.

1) Нескінченна в обидві сторони стрічка, розбита на клітинки, в кожній з яких може перебувати один (і тільки один) символ зовнішнього алфавіту $A=\{ \alpha, \beta, \gamma, ... \}$. Інформація записується на стрічці у вигляді слова в алфавіті A.

2) Голівки, що зчитує, яка в один дискретний момент часу оглядає одну клітинку і може перебувати в одному з внутрішніх станів $q_i \in Q$, де Q - алфавіт внутрішніх станів. При цьому вона розпізнає записаний в чарунці символ зовнішнього алфавіту, записує замість нього інший символ (можливо, той же самий), переходить в інший стан (можливо – залишається в поточному), після чого зсувається вправо, вліво або залишається на місці (П, Л, Н) (див. рис. 6.1.).

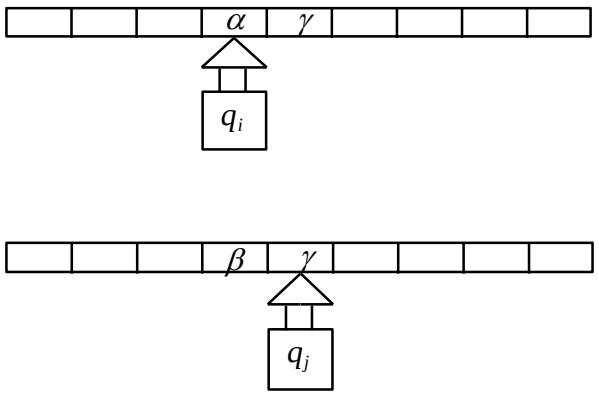


Рис.6.1. Схематичне зображення машини Тюрінґа

Таким чином, множина внутрішніх станів Q являє собою пам'ять машини Тюрінґа: коли машина в різних станах бачить один і той же самий символ, вона може виконати різні дії.

3) Наступна частина машини Тюрінґа - це програма, що складається з команд виду:

$\alpha q_i \rightarrow \beta q_j \xi$, $\xi \in \{Л, П, Н\}$, $q_i, q_j \in Q$, $\alpha, \beta \in A$.

В результаті виконання цієї команди символ α на стрічці буде замінений символом β, головка змінить своє положення в залежності від ξ і внутрішній стан зміниться з q_i на q_j.

Послідовність команд задає алгоритм переробки слова. На вхід машини Тюрінґа подається вихідне слово W в алфавіті A і задається початковий стан q_0: початкова конфігурація. Робота машини Тюрінґа відбувається по *тактах,* або

по *кроках*. При роботі машини Тюрінґа можливе розширення зовнішнього алфавіту *A* допоміжними символами, які після переробки слова замінюються символами алфавіту *A* або витираються. У число символів алфавіту *A* обов'язково входить порожній символ Λ. Вважається, що спочатку все чарунки стрічки заповнені порожніми символами.

В процесі роботи можливі ситуації:

1. Машина Тюрінґа переробляє вихідне слово *P* в *Q* і зупиняється; і можна сказати, що дана машина може бути застосована до слова *P*.
2. Машина Тюрінґа, починаючи роботу з слова *P*, ніколи не зупиниться; тоді - дана машина Тюрінґа не може бути застосована до слова *P*.

Розглянемо **приклад**.

Нехай заданий алфавіт *A* = {|, *}, вихідне слово в якому може мати вигляд: *P* = | | | * | * | |. У **початковому стані q_0 машина оглядає крайній лівий символ**. Машина Тюрінґа повинна перетворити це слово в порожнє слово, тобто повинна реалізовувати алгоритм обчислення нуль-функції: *0(x)*= Λ. Для цього вона повинна, рухаючись зліва направо, замінити кожен символ на стрічці на порожній символ і зупинитися, як тільки побачить крайній правий символ.

A \ *Q*	q_0
|	Λ $q_0 П$
*	Λ $q_0 П$
Λ	!

Програму для машини Тюрінґа зазвичай записують у вигляді таблиці.

Це дуже простий алгоритм.

Розглянемо програму для алгоритму додавання символу | до слова в тому ж алфавіті. Нехай вихідне слово має вигляд: *P* = | | |. Це слово можна розглядати як число 3, а додавання символу | - як обчислення функції *f(x)=x+1* в унарній системі числення. Програма для машини Тюрінґа насправді дуже проста: головка машини пропускає все символи |, рухаючись зліва направо, і, дійшовши до крайнього правого порожнього символу Λ дописує туди ще один символ |, після чого зупиняється в своєму заключному стані.

A \ *Q*	q_0
|	| $q_0 П$
Λ	| ! *H*

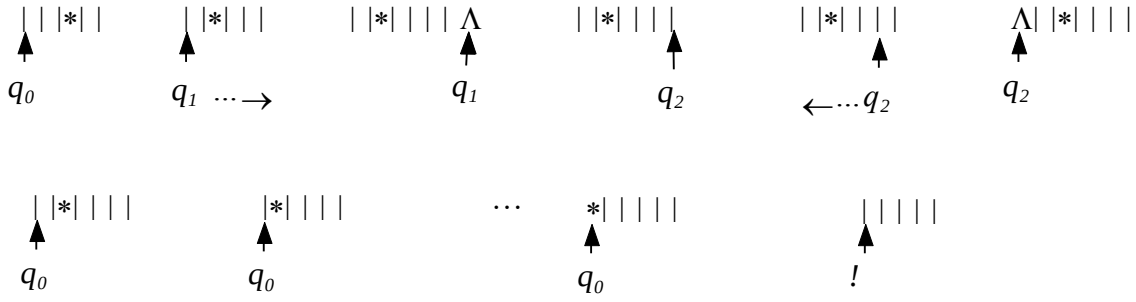
Очевидно, що точно так само можна скласти програму для додавання будь-якого символу алфавіту *A* в кінець слова, тобто такий алгоритм реалізовує вихідну рекурсивную функцію $P_{\zeta}(x)= x\,\zeta$, де $\zeta \in A$.

Третя вихідна функція: функція проекції $U_i^j(x)=x_i$, є не що інше, як елементарна дія машини Тюрінґа - розпізнавання символу.

Розглянемо тепер обчислення більш складної функції: *F (x, y) = x + y*, де *x*, *y* - слова, задані в унарній системі числення. Нехай вихідна конфігурація на

стрічці задана так, що в початковому стані головка оглядає крайній лівий символ | (див. Рис.6.2.), Словом на стрічці є два числа, розділені символом *.

Складемо машину Тюрінґа, що працює за наступним алгоритмом. У початковому стані q_0 машина бачить крайню ліву паличку, витирає її і, перейшовши в стан q_1, рухається вправо, поки не побачить крайній праворуч порожній символ. Тоді на місці порожнього символу машина записує паличку, переходить в новий стан q_2 і рухається вліво в тому ж стані q_2, пропускаючи всі символи, поки не дійде до крайнього зліва порожнього символу. Тоді машина залишає його на місці, зсувається вправо і переходить в стан q_0. Після цього процес повторюється до тих пір, поки в стані q_0 машина не побачить символ *. Це означає, що вже всі палички перенесені зліва направо, машина може витерти символ * і зупинитися, тобто перейти до заключного стану.



$A \setminus Q$	q_0	q_1	q_2
|	$\Lambda\ q_1 П$	$\|\ q_1 П$	$\|\ q_2 Л$
*	$\Lambda\ !\ П$	$*\ q_1 П$	$*\ q_2 Л$
Λ	$\Lambda\ !\ Н$	$\|\ q_2 Л$	$\Lambda\ q_0 П$

Неважко помітити, що процес обчислення суми рекурсивний: він заснований на попередньому алгоритмі, коли до одного з доданків (в даному випадку, до правого слова) додається стільки одиниць, скільки їх міститься в іншому доданку. Для порівняння відтворимо процес обчислення суми двох доданків за допомогою рекурсивної функції:

$f(x, 0) = x,$

$f(x, y + 1) = f(x, y) + 1.$

$f(2, 3) = f(2, 2) + 1 = (f(2, 1) + 1) + 1 = ((f(2, 0) + 1) + 1) + 1.$

На цьому завершується прямий хід рекурсії: складена рекурсивна схема обчислення функції. Тепер зворотний хід рекурсії обчислює її значення:

$((2 + 1) + 1) + 1 = (3 + 1) + 1 = 4 + 1 = 5.$

$A \setminus Q$	q_0	q_1	
|	$\Lambda\ q_1 П$	$\|\ q_1 П$	
*	$\Lambda\ Н\ !$	$\|\ Н\ !$	

Звісно можна написати програму для машини Тюрінґа, яка просто витирає першу паличку і записує її на місце *. Але така програма, хоча і дуже ефективна, не буде рекурсивною і не буде містити уніфіковані дії.

6.1.2. Машина Тюрінґа, що розпізнає

Машина Тюрінґа, що розпізнає – це машина, яка обчислює предикат *P(W)* таким чином, що якщо *P(W)=T*, то машина зупиняється в стані *так!*, а якщо *P(W)=F*, то в стані *ні!*.

A \ *Q*	q_0	q_1
|	| q_1 П	| q_0 П
Λ	Λ *так*!	Λ *ні*!

Наприклад, машина, що розпізнає парність числа, представленого в унарній системі числення, наведена нижче в таблиці. Рухаючись зліва направо, машина зупиняється на порожньому символі в стані *так!*, якщо число парне, і в стані *ні!,* якщо непарне.

6.1.3. Композиція машин Тюрінґа

Можна виділити деякий набір елементарних алгоритмів, з яких можна отримувати більш складні алгоритми за правилами композиції машин Тюрінґа. Ми вже показали, що елементарні функції

- Нуль-функція *0 (x)* = Λ;
- Додавання символу ζ*:* $P_{\zeta}(x)=x\zeta$, де $\zeta \in A$;
- Проектуюча функція $U_i^j(x)=x_i$;
- Тотожне перетворення *T*(*x*)=x;
- Алгоритм копіювання слова *Copy* (*x*)=*x***x*;
- Змінюючий алгоритм $Rep_{x_j}^{\ x\ i}(x)=x_j$, де x_i, $x_j \in A \cup B$, де *B* – допоміжний алфавіт.

З цих елементарних алгоритмів можна утворювати більш складні за допомогою композиції машин Тюрінґа.

1. Послідовна композиція.

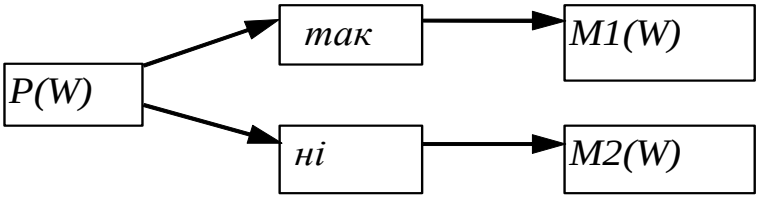
Нехай M1 і M2 - дві машини Тюрінґа. Тоді, ототожнив заключний стан машини M1 з початковим станом машини M2 і, при необхідності, перенумерувавши внутрішні стани машини M2, отримаємо нову машину Тюрінґа, яка, починаючи роботу з слова *W*, спочатку буде виконувати над ним перетворення, що виконуються машиною M1, а потім буде працювати як машина M2. Композицію машин Тюрінґа можна позначити: *M1°M2=M2(M1(W))*. Таким чином, послідовна композиція машин Тюрінґа здійснює суперпозицію функцій.

2. Паралельна композиція.

Якщо на стрічці записано слово *W*, яке представимо як конкатенація двох слів *P||Q*, можна скласти таку машину Тюрінґа, яка буде працювати як машина *M1* над підсловом *P* і як машина *M2* над підсловом *Q*, а потім здійснить конкатенацію результатів: *M*1(*P*) || *M*2(*Q*).

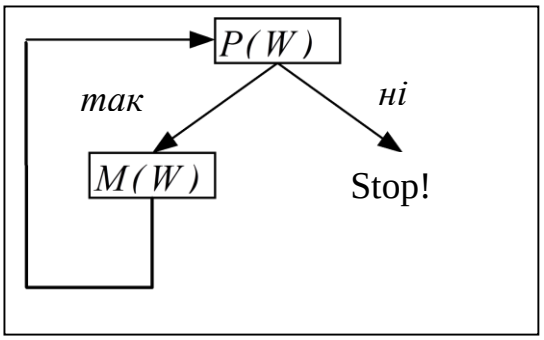
3. Машина Тюрінґа, що розпізнає

Якщо існують машини Тюрінґа *M1* і *M2* і машина Тюрінґа *P*, що розпізнає, то можна скласти таку машину Тюрінґа *M*, що, починаючи роботу зі слова *W*, машина Тюрінґа працює спочатку як така, що розпізнає – *P* (*W*). Якщо вона закінчує обробку вихідного слова в стані *так*, то далі працює машина *M1(W),* а якщо в стані *ні*, то машина *M2(W).*



4. Циклічна машина Тюрінґа.

Можна побудувати таку машину Тюрінґа М, яка, починаючи роботу з слова W_0, спочатку працює як розпізнавальна машина *P*, що обчислює предикат $P(W_0)$. Якщо вона завершує свою роботу в стані *так!,* то далі вона працює як машина Тюрінґа М1 над словом W_0 і завершує свою роботу з вихідним словом W_1. Далі управління передається розпізнавальній машині *P*, що обчислює предикат $P(W_1)$, і так далі, до тих пір, поки що розпізнавальна машина Тюрінґа *P* не зупиниться в стані *ні!* на слові W_k. Тоді машина Тюрінґа М зупиняється в своєму заключному стані.



Було математично доведено, що для реалізації будь-яких алгоритмів досить цих чотирьох структур. Композиція машин Тюрінґа містить в собі основну ідею структурного програмування, і саме Тюрінґу належить доказ повноти цих чотирьох базових алгоритмів для реалізації алгоритму будь-якої складності.

Розглянемо деякі приклади побудови машин Тюрінґа (МТ):

1) розпізнає, чи є слово паліндромом,
в алфавіті {a, b}:
aabbaa-> так; aabaa-> так; abbaa-> немає

A \ Q	q_0	q_1	q_2	q_3	q_4	q_5
a	$\Lambda q_1 П$	a $q_1 П$	a $q_2 П$	Λ $q_5 Л$	ні!	a $q_5 Л$
b	$\Lambda q_2 П$	b $q_1 П$	b $q_2 П$	ні!	Λ $q_5 Л$	b $q_5 Л$
Λ	так!	$\Lambda q_3 Л$	$\Lambda q_4 Л$	так!	так!	$\Lambda q_0 П$

2) копіювання слова в алфавіті {0,1}: 001101 -> 001101*001101

$A \setminus Q$	q_0	q_1	q_2	q_3	q_4
0	$0q_0П$	$0\ q_1\ Л$	a $q_3П$	$0q_3П$	$0q_4П$
1	$1q_0П$	$1\ q_1\ Л$	b $q_4П$	$1q_3П$	$1q_4П$
a		$0\ q_2П$			
b		$1\ q_2П$			
*		$*\ q_1Л$	!	$*q_3П$	$*\ q_4П$
Λ	$*q_1Л$	$Λq_2\ П$		$0q_1Л$	$1q_1Л$

6.2. Нормальний алгоритм Маркова

6.2.1. Робота алгоритму Маркова

Нормальні алгоритми Маркова (нормальні алгорифми) — формалізація поняття алгоритму, що є системою послідовних застосувань підстановок до слів певного алфавіту, введена математиком А. А. Марковим у 1956-му році.

Визначення 6.1. Нормальний алгоритм Маркова (НАМ) - це кінцевий впорядкований набір підстановок виду $P_i \to (.)\ Q_i$, $i = 1, 2, ..., n$. $P_i \to Q_i$ - звичайна підстановка, яка означає, що крайнє ліве входження подслова P_i в W замінюється на Q_i , $P_i \to .\ Q_i$ - кінцева підстановка, тобто після її виконання робота алгоритму завершується.

Початкове слово $W \in A^*$ переробляється за допомогою алгоритму наступним чином. Починаючи з першої підстановки, алгоритм шукає перше ліве входження подслова P_i в слові W. Якщо воно знайдено, то воно замінюється на Q_i, після чого список підстановок проглядається з самого початку. Якщо ж входження P_i не було знайдено, то вибирається наступна підстановка. Якщо була виконана заключна підстановка (з крапкою), то процес переробки слова завершується і тоді говорять, що алгоритм застосуємий до даного слова. Може виявитися, що процес не завершується, тобто жодна із заключних підстановок не застосовувалася в процесі переробки слова. У цьому випадку говорять, що алгоритм не застосуємий до даного слова.

Приклад 1. Нормальний алгоритм Маркова для складання двох чисел в унарною системі, тобто алгоритм обчислює функцію $f(x, y) = x + y$ в алфавіті $A = \{|, *\}$. Список підстановок складається з двох (це приклад цілком рекусрсивного алгоритму).

1. $* \mid \to \mid *$ *[не рекурсивний НАМ*
2. $* \to . \Lambda$ *1.* $|* \mid \to . \mid \mid$ *]*

Процес переробки слова полягає в наступному (в дужках вказано номер застосовуваної підстановки):

| | | * | | | | ⇒ (1) | | | | * | | | ⇒ (1) | | | | | * | | ⇒ (1) | | | | | | * | ⇒ (1) | | | | | | | * ⇒ . (2)
| | | | | | |

Приклад 2. Алгоритм обернення слова в алфавіті *A={a, b, c,..., z}*. Наприклад, вихідне слово *abc* читається в зворотному порядку як слово *cba* . Список підстановок:

1. $\alpha\alpha \to \beta$
2. $\beta x \to x\beta$
3. $\beta\alpha \to \beta$
4. $\beta \to . \Lambda$
5. $\alpha yx \to x\alpha y$
6. $\Lambda \to \alpha$

де x, y ∈ *A*; α, β - допоміжні символи. Для даного слова процес роботи алгоритму показаний нижче.

Λabc ⇒ (6) αabc ⇒ (5) bαac ⇒ (5) bcαa ⇒ (6) αbcαa ⇒ (5) cαbαa ⇒ (6) αcα bα a ⇒ (6) ααcαbαa ⇒ (1) βcαbαa ⇒ (2) cβαbαa ⇒ (3) cβbαa ⇒ (2) cbβαa ⇒ (3) cbβa ⇒ (2) cbaβ ⇒ . (4) cba.

Приклад 3. Перевірити, чи є паліндромом слово з алфавіту *A={a, b, c, ..., z}*. Наприклад, аввва ⇒ yes, fddddf ⇒ yes, baba ⇒ no. Список підстановок при x, y ∈ A:

1. x*y–>yx*
2. xx*–>λ
3. yx*–>.no
4. x*–>yes (непарні)
5. λx–> x*
6. λ–>.yes (парні)

abbba⇒(5)a*bbba⇒(1)ba*bba⇒(1)bba*ba⇒(1)bbba*a⇒(1)bbbaa*⇒(2)bbb⇒(5)b*bb⇒(1)bb*b⇒(1)bbb*⇒(2)b⇒(5)b*⇒.(4)yes

*cdffdc⇒(5)c*dffdc⇒(1)dc*fdc⇒...⇒(1)dffdcc*⇒(2)dffd⇒(5)d*ffd⇒ (1) fd*fd⇒...(1)ffdd*⇒(2)ff⇒(5)f*f⇒(1) ff*⇒(2)* λ*⇒.(6)yes*

aabba⇒(5)a*abba⇒(1)aa*bba⇒…(1)abbaa*⇒(2)abb⇒(5)a*bb⇒(1) ba*b⇒(1)bba*⇒.(3)no

Приклад 3 схеми нормального алгоритму можна навести наступну схему в алфавіті з п'яти літер, яка реалізовує унарне множення:

$$\begin{cases} |b & \to & ba| \\ ab & \to & ba \\ b & \to & \\ *| & \to & b* \\ * & \to & c \\ |c & \to & c \\ ac & \to & c| \\ c & \to & \bullet \end{cases}$$

При застосуванні алгоритму за наведеною вище схемою до слова |*|| будуть отримуватися слова:

1. $|b*|$,
2. $ba|*|$,
3. $a|*|$,
4. $a|b*$,
5. $aba|*$,
6. $baa|*$,
7. $aa|*$,
8. $aa|c$,
9. aac,
10. $ac|$
11. $c||$,
12. $||$.

Результатом застосування буде слово ||, що узгоджується з |*||=||.

Приклад 4.

Даний алгоритм перетворює двійкові числа в «унарні» (в яких записом цілого невід'ємного числа N є рядок з N паличок). Наприклад, двійкове число 101 перетвориться в 5 паличок: |||||.

Алфавіт { 0, 1, |, Λ }.

Правила:

1. $1 \rightarrow 0|$,
2. $|0 \rightarrow 0||$,
3. $0 \rightarrow .\ \Lambda$

Покрокове виконання алгоритму при застосуванні алгоритму за наведеною вище схемою до слова 101 буде отримувати слова:

1. 0|01
2. 0|00|
3. 00||0|
4. 00|0|||
5. 000|||||
6. 00|||||
7. 0|||||
8. |||||

6.2.2. Еквівалентність нормальних алгоритмів і машини Тюрінґа

Теорема 6.1. Нехай M - машина Тюрінґа із зовнішнім алфавітом A і внутрішнім алфавітом Q. Тоді існує нормальний алгоритм Маркова з алфавітом $A \cup Q$, еквівалентний даній машині Тюрінґа.

Доведення. Нехай дана машина Тюрінґа M із зовнішнім алфавітом $A = \{ x_i \}$, де $i = 1, 2, ..., n$, і внутрішнім алфавітом $Q = \{ q_i \}$, $j = 0, 1, ..., n$. Двовимірну конфігурацію на стрічці можна записати як $x_1 x_2 ... q_j x_i x_i$

$_{+1} \ldots x_n$, де символ поточного стану q_j стоїть перед символом, який оглядає головка машини Тюрінґа в даний момент часу. Ми отримаємо слово в алфавіті $A \cup Q$. Тоді можна замінити кожну команду машини Тюрінґа на послідовність підстановок наступним чином:

	МТ		НАМ	
1.	$q_j\alpha \to \beta q_r H$	$\Leftrightarrow$	$q_j\alpha \to q_r\beta$	
2.	$q_j\alpha \to \beta q_r П$	$\Leftrightarrow$	$q_j\alpha x_i \to \beta q_r x_i,$	$\forall x_i \in A$
3.	$q_j\alpha \to \beta q_r Л$	$\Leftrightarrow$	$x_i q_j\alpha \to q_r x_i\beta$	$\forall x_i \in A$
4.			$q_j \to .\ \Lambda\ \forall q_j \in Q$	
5.			$\Lambda \to q_0$	

Таким чином, якщо є деяка програма для машини Тюрінґа, то за допомогою цих п'яти правил її можна перетворити в алгоритм Маркова.

Приклад. Нехай задана програма для машини Тюрінґа в алфавіті A={|,*}.

	q_0
|	| q_0 П
Λ	| ! Н
*	* Q_0 П

Нормальний алгоритм Маркова, еквівалентний цій машині Тюрінґа має вигляд:

1. $q_0 | | \to | q_0 |$
2. $q_0 | * \to | q_0 *$
3. $q_0 | \Lambda \to | q_0 \Lambda$
4. $q_0 \Lambda \to . |$
5. $\Lambda \to q_0$

Теорема 6.2. (зворотна) Для кожного нормального алгоритму Маркова можна побудувати еквівалентну йому машину Тюрінґа.

Схема доведення.

1. Пошук подслова P в слові W (необхідно скласти машину Тюрінґа, яка шукає подслово P в слові W; головка машини Тюрінґа, в результаті, буде стояти на першому символі подслова P).
2. Побудувати машину Тюрінґа, яка замінює подслово P на слово Q (для кожної підстановки алгоритму Маркова можна побудувати машину Тюрінґа, що виконує її).

Отримана композиція машин Тюрінґа буде виконувати ту ж процедуру переробки слів, що і алгоритм Маркова.

Наведені вище дві теореми стверджують еквівалентність машини Тюрінґа і нормального алгоритму Маркова. Для нормального алгоритму Маркова можна визначити розпізнавальний алгоритм Маркова, який обчислює деякий предикат і має заключні

підстановки $P \to_{так} Q$ або $P \to_{ні} Q$; аналогічно можна визначити поняття композиції (за тими ж схемами, що і для машини Тюрінґа).

Доведена теорема показує, що якщо побудована машина Тюрінґа для вирішення будь-якої задачі, тобто, якщо функція обчислювана за Тюрінґом, то для неї існує і нормальний алгоритм Маркова, а, якщо вона обчислювана за Марковим, то і навпаки. Іншими словами, класи функцій, обчислюваних за Тюрінґом і за Марковим, збігаються. Кажуть, що нормальні алгоритми повні за Тюрінгом, тобто можуть описувати всі алгоритми, що можуть виконуватись будь-яким комп'ютером.

Доведено, що відносно виконуваних перетворень, нормальні алгоритми збігаються з іншими класами алгоритмів, введених для уточнення інтуїтивного поняття алгоритму.

Визначення алгоритмів у нормальному вигляді дуже схоже на числення, і це є дуже корисним у випадках, коли поняття числення в досліджуваному розділі математики або кібернетики широко застосовують, як, приміром, в математичній логіці або в математичній лінгвістиці.

Використовуючи поняття нормального алгоритму, Марков та інші дослідники довели нерозв'язність цілого набору алгоритмічних проблем.

6.3. Клас функцій, обчислюваних за Тюрінґом

Вище було зазначено, що, якщо побудована машина Тюрінґа для вирішення будь-якої задачі, тобто, якщо функція обчислювана за Тюрінґом, то для неї існує і нормальний алгоритм Маркова, тобто вона обчислювана і за Марковим, і навпаки. Іншими словами, класи функцій, обчислюваних за Тюрінґом і за Марковим, збігаються. Залишилося зрозуміти, який же клас функцій обчислюваний за Тюрінґом.

Теорема 6.3. Функція $F(x_1, x_2, \ldots x_n)$ рекурсивна (частково рекурсивна) тоді і тільки тоді, коли вона обчислювана за Тюрінґом.

Доведення.

1. Будь-яка рекурсивна функція обчислюваних за Тюрінґом.

- $Z(x) = \lambda$ (нуль функція - машина Тюрінґа, яка знищує будь-яке слово на стрічці)
- $N(x) = x + 1$ (машина Тюрінґа, яка до будь-якому слову на стрічці дописує заданий символ α: $W\alpha$)
- $U_j\ n(x) = x_i$ (машина Тюрінґа, яка при подачі на її вхід деякої послідовності слів $W_1 \# W_2 \# \ldots \# W_n$ вибирає слово W_i)
- $\varphi(x_1, x_2) = g(F_1(x_1, x_2), F_2(x_1, x_2))$ суперпозиція функцій, реалізується за допомогою композиції машин Тюрінґа:

$x_1 * x_2 \to x_1 * x_2 \# x_1 * x_2 \to F_1(x_1 * x_2) \# F_2(x1 * x_2) \to Mg(F_1(x_1 * x_2) \# F_2(x_1 * x_2)) \to \varphi(x_1, x_2)$

$M\varphi = Copy(x_1 * x_2)0(MF_1 || MF_2)0зам*_{\#} \bullet Mg$

- найпростіша схема примітивної рекурсії: $\varphi(0) = C$, $\varphi(x+1) = F(\varphi(x))$
  - M_D: $x \# M \# z \to x || M+1 || F(z)$
  - M_C: $x \# 0 \# C$
  - Ф: $x \# M \# z \to ?\ M < z \to$так!

    $M_C 0$ (поки Ф повторити M_D) - задає схему примітивної рекурсії.

- μ- оператор

Машина Тюрінґа, що обчислює μ -оператор, буде знаходити мінімальне слово в словниковому впорядкуванні слів, яке задовольняє даному предикату.

Оскільки всі шість пунктів визначення рекурсивних функцій реалізовані у вигляді композиції машин Тюрінґа, то будь-яка рекурсивна функція обчислювана на машині Тюрінґа. Другу частину теореми - якщо функція обчислювана на машині Тюрінґа, то вона рекурсивна, - приймемо без доведення[9].

Таким чином, було показано, що клас функцій, обчислюваних на машині Тюрінґа, є частково рекурсивним класом.

Крім цих алгоритмічних схем було знайдено і запропоновано багато інших, наприклад, машина фон Неймана, машина Посту, блок-схеми Посту, формальне числення рекурсивних функцій Ербрана - Геделя і інші. Виявилося, що всі вони еквівалентні між собою, і, отже, всі вони обчислюють рекурсивні або частково рекурсивні функції. Ця обставина послужила причиною того, що ряд вчених (Черч, Тюрінґ, Марков) практично одночасно висловили гіпотезу, яка відома як теза Черча.

6.3.1. Теза Черча

Кожна функція є ефективно обчислюваною тоді і тільки тоді, коли вона рекурсивна.

Іншими словами, теза Черча пропонує розуміти під ефективної обчислюваностю існування алгоритмічної схеми, а оскільки всі знайдені алгоритмічні схеми обчислюють тільки клас рекурсивних (частково рекурсивних) функцій, то під ефективної обчислюваності тоді розуміється рекурсивність.

Поняття алгоритму, яке було дано, не є точним математичним визначенням. Спроби знайти таке визначення привели до створення математично строгих алгоритмічних схем. Значення гіпотези полягає в тому, що вона уточнює інтуїтивно зрозуміле, але неточне і розпливчасте поняття алгоритму через більш спеціальне, але математично точне поняття алгоритмічної схеми. Тепер можна говорити про можливості розв'язання деякого класу задач в термінах існування машини Тюрінґа (або алгоритму Маркова, або якого-небудь іншого з відомих алгоритмічних схем).

Теза Черча не є теоремою, про його доведення мова не йде, - це твердження, що пропонує ототожнити ефективну обчислювальність з існуванням алгоритмічної схеми. Його можна приймати чи не приймати, однак, більшість подальших результатів в теорії алгоритмів і математичної логіки заснована на прийнятті тези Черча.

---

[9] З доведенням можна познайомитися, наприклад, в [Трахтенброт Б.А. «Алгоритми та обчислювальні автомати», 1974].

Впевненість в справедливості тези Черча заснована перш за все на досвіді: в результаті численних досліджень не вдалося знайти будь-якої іншої алгоритмічної схеми, що обчислює більш широкий клас функцій, ніж рекурсивний. Всі знайдені алгоритмічні схеми виявилися еквівалентними між собою і, отже, еквівалентними машині Тюрінґа. Тому обчислюваними функціями, згідно з тезою Черча, є ті і тільки ті, які є рекурсивними (частково рекурсивними). Тоді, якщо приймати тезу Черча, повинні існувати і необчислювальні функції[10].

6.3.2. Необчислювальні функції

Теорема 6.4. Множина функцій над словами деякого алфавіту еквівалентна множині функцій на множині натуральних чисел, а множина функцій на множині натуральних чисел незліченна.

Доведення. Припустимо, існує перерахування $L=F_1, F_2, ..., F_k,...$ всіх арифметичних функцій (тобто функцій, визначених на множині $\boldsymbol{N}$). Побудуємо функцію $U(n)$ у такий спосіб:

$$U(n) = \begin{cases} 1, \text{ якщо } F_n(n) \text{ не визначена,} \\ F_n(n) + 1, \text{ в іншому випадку.} \end{cases}$$

Тоді $U(n)$ – усюди визначена на $\boldsymbol{N}$ функція. Якщо множина арифметичних функцій злічена, тобто існує перерахування L, то побудована нами функція входить в цей перелік з яким-небудь номером, наприклад, m, тобто $U(n) \in L$ і $U(n) = F_m$. Тоді, обчислюючи значення цієї функції від m згідно її визначення, отримаємо: $U(m) = F_m(m)$: $F_m(m)=F_m(m)+1$, якщо $F_m(m)$ визначена, що неможливо, тобто ми прийшли до протиріччя.

Побудована нами функція, по-перше, доводить незліченність множини арифметичних функцій, а по-друге, за способом побудови вона є необчислювальної функцією.

Як приклад визначення такої функції розглянемо наступну задачу. Уявімо собі велику бібліотеку, в якій книги розставлені по розділах. Для кожного розділу складено каталог. Кожна книга має призначену їй ціну, а оскільки каталог розділу містить відомості про всі книги, він повинен бути найціннішою книгою. Тому його вартість визначена як функція від вартості найдорожчої книги в розділі: $C_к=C_{max}+1$. Книг в бібліотеці виявилося настільки багато, що всі каталоги були складені в окремий розділ, і був складений каталог каталогів. Йому також повинна бути призначена ціна за зазначеним вище правилом: вартість його повинна бути найвищою в цьому розділі, тобто $C_к^к=C_{Kmax}$, і повинна бути на одиницю більше ціни найдорожчої книги в розділі: $C_к^к= C_{Kmax}+1$. Але оскільки каталог каталогів сам є каталогом,

[10] Наприклад, функція Аккермана.

то його ціна повинна бути на одиницю більше його власної вартості: $C_к^к = C_к^к + 1$. Таким чином, виявилося, що вартість його неможливо обчислити по заданому правилу.

Таким чином, встановлено що множина всіх арифметичних функцій незліченна, тобто його потужність дорівнює $2^{\aleph_0} = C$. Постараємося з'ясувати, яка потужність множини всіх рекурсивних функцій. Тепер, коли встановлена еквівалентність всіх рекурсивних функцій і всіх машин Тюрінґа, для цього достатньо "перерахувати" всі можливі машини Тюрінґа.

Розглянемо множину символів, необхідних для зображення програми для машини Тюрінґа. Нехай зовнішній алфавіт складається з двох символів: $A = \{0, 1\}$ (можливість кодування будь-яких даних в двійковій системі числення ні у кого не викликає сумніву – адже саме так працюють сучасні обчислювальні машини). Нехай стани машини Тюрінґа позначаються десятковими числами, тобто внутрішній алфавіт $Q = \{0, 1, 2, 3, 4, ..., 9\}$, і три символи нам знадобляться для вказівки руху головки: {П, Л, Н}. Кожній клітці в таблиці, яка зображує програму машини Тюрінґа, поставимо у відповідність команду у вигляді: $a_i q_j \to a_л \zeta q_д$, де a_i, $a_k \in A$, $\zeta \in$ {П, Л, Н}, q_j, $q_д \in Q$. Будемо використовувати роздільники: , - для заміни символу $\to$ та ; - для відділення однієї команди від іншої. Тоді послідовність команд може бути зображена, наприклад, так: 10, 0П12; 01, 0П1; 11, 1Л2; і так далі. Запис 10, 0П12; означає, що в стані 0 машина бачить символ 1, записує на його місце символ 0, зсувається вправо і переходить в стан 12. Тоді для запису програми машини Тюрінґа досить тільки 15 символів (враховуючи, що символи зовнішнього алфавіту 0 і 1 входять і в алфавіт Q) . В результаті будь-яка програма може бути представлена, як число в п’ятнадцятковій системі числення.

Записана так, як зазначено вище, програма для машини Тюрінґа M називається *кодом* машини Тюрінґа і позначається $d(M)$, а п’ятнадцяткове число, яке можна поставити їй у відповідність, називається *індексом* машини Тюрінґа.

Таким чином, кожній машині Тюрінґа можна поставити у відповідність її індекс – число в п’ятнадцятковій системі числення. Але множина чисел в п’ятнадцятковій системі числення злічено (так як зліченою є будь-яка нескінченна послідовність кінцевих послідовностей символів), – значить, множина всіх машин Тюрінґа теж злічена. Отже, потужність множини всіх рекурсивних функцій (оскільки вони і тільки вони обчислювані на машині Тюрінґа) дорівнює $\aleph_0$, і $\aleph_0 < c$.

Ми показали, що всіх арифметичних функцій більше, ніж обчислюваних функцій, а це означає, що існують необчислювальні функції. У загальному вигляді це означає існування таких проблем (задач), для яких неможливо побудувати алгоритм, тобто машину Тюрінґа, – якщо приймати тезу Черча. Це не означає, втім, що ці проблеми взагалі не вирішується. Проблема вважається

алгоритмічно нерозв'язною, якщо існує хоча б одна задача з даного класу задач, для якої математично строго доведена неможливість побудови алгоритму, тобто машини Тюрінґа, або нормального алгоритму Маркова, або будь-якої іншої алгоритмічної схеми. Тюрінґ запропонував інструмент для доказу алгоритмічної нерозв'язності - універсальну машину Тюрінґа.

6.4. Алгоритмічно нерозв'язні проблеми

6.4.1. Універсальна машина Тюрінґа.

Будь-яка програма для машини Тюрінґа може бути закодована деяким кодом. Позначимо *d*(M) – код програми машини Тюрінґа. Тоді можна побудувати таку машину Тюрінґа M'(*d* (M) * *W*) на вхід якої буде подаватися код *d*(M) разом з вхідним словом *W*. Тоді машина Тюрінґа спочатку перевіряє, чи є *d*(M) синтаксично правильним програмним кодом для машини Тюрінґа. І якщо це так, то виконує програму *d*(M) над словом *W*. Якщо ж ні, то вона зупиняється в стані "ні!". Така машина Тюрінґа називається *універсальною машиною Тюрінґа* .

Універсальна машина Тюрінґа була математичною моделлю синтаксичного аналізатора і компілятора. Універсальна машина Тюрінґа є математичним апаратом для доказу алгоритмічної нерозв'язності проблем. Аналогічно можна побудувати і універсальний алгоритм Маркова.

6.4.2. Проблема зупинки для машини Тюрінґа

Історично першою алгоритмічно нерозв'язною проблемою, доведеною Тюрінґом, була проблема зупинки для машини Тюрінґа. Вона формулюється так: чи можна побудувати таку універсальну машину Тюрінґа M ′, що будучи застосовано. до слова (*d*(M)*W*), вона буде зупинятися в "так!"-стані, якщо машина Тюрінґа M застосовується до слова *W*, тобто зупиняється на цьому слові, і зупинятися в " *ні*!"-стані, якщо машина M не може бути застосована до слова *W*.

Теорема 6.5. Проблема зупинки для машини Тюрінґа алгоритмічно нерозв'язна.

Доведення. Припустимо, що така універсальна машина Тюрінґа M′ побудована. Це означає, що в таблиці програми універсальної машини Тюрінґа є дві команди, по яким вона в деякому стані *q′* на деякому символі σ зупиняється в "так!"-стані, і в деякому стані *q″* на деякому символі τ зупиняється в "ні!"-стані, тобто в її таблиці є клітини:

	q′	*q″*
σ	σ*так*!	
τ		τ*ні* !

Побудуємо тепер машину Тюрінґа M ′′ так, що змінимо в ній тільки одну команду: в стані q' на символі σ машина не зупиняється, а продовжує нескінченно писати символ σ в одну і ту ж чарунку. Таблиця для машини M ′′ показана нижче.

	q'	q''
σ	σ*так*Н	
τ		τ*ні* !

Тепер на вхід машини M ′ подамо її власний код, для якого вхідним словом буде код машини Тюрінґа M′′: M′(*d*(M′)**d*(M′′)). Чи зможе тепер машина M′ розпізнати, чи зупиниться вона сама на коді машини Тюрінґа M′′? Універсальна машина Тюрінґа працює так, як машина, код якої подано на її стрічку, отже, дійшовши до символу σ в стані q' (в коді машині M′′ - в слові, що переробляється), машина M′ зупиниться в стані " *так*!", в той час як M′′ не зупиняється! І навпаки, коли в стані q'' машина M ′′ бачить символ τ, вона видає повідомлення "ні" і зупиняється, а машина M ′ для цієї ситуації теж зупиняється в стані "ні", тобто вона видає повідомлення про те, що машина M ′′ не зупиняється!

Таким чином, машина Тюрінґа M ′ не розпізнає свою власну зупинку в коді машини M ′′. Сам факт можливості побудови такої машини Тюрінґа, яка не може розпізнати зупинку, доводить алгоритмічну нерозв'язність цієї проблеми.

Проблема алгоритмічної нерозв'язності пов'язана з проблемою *самозастосовності*.

6.4.3. Проблема самозастосовності

Будемо говорити, що машина Тюрінґа M самозастосовна, якщо вона зупиняється на своєму власному коді *d*(M), і не самозастосовна, якщо вона не зупиняється на своєму коді *d*(M).

Теорема 6.6. Проблема розпізнавання самозастосовності машини Тюрінґа алгоритмічно не розв'язна.

Доведення. Припустимо, що побудована машина Тюрінґа M′, яка зупиняється в "так!"-стані, якщо машина самозастосовна, і в "ні!" стані, якщо ні самозастосовна. Тоді в програмі машини Тюрінґа присутні дві команди виду: σq'→σ*так*!, τq''→τ*ні*!

Аналогічно до попереднього доведення, побудуємо машину M′′, змінивши одну команду: σq'→σ*так*Н (тобто машина не зупиняється, а нескінченно пише символ σ на стрічці). Тепер, якщо на вхід M′ подати *d*(M′′): M′(*d*(M′′)) машина буде видавати "так!", - тобто повідомляти, що M′′ самозастосовна, в той час як вона ні самозастосовна, так як не зупиняється, і, навпаки, вона буде видавати повідомлення "*ні!*", якщо машина M′′ не зупиняється, тобто якщо вона самозастосовна.

Таким чином, доказ нерозв'язності проблеми самозастосовності зводиться до доведення неможливості розпізнавання зупинки машини Тюрінґа. Звідси виник і загальний метод доведення алгоритмічної нерозв'язності інших задач: метод зведення проблем. У цьому методі нова проблема зводиться до деякої іншої проблеми, для якої вже доведена алгоритмічна нерозв'язність. Найчастіше проблема зводиться до проблеми самозастосовності. Як приклад розглянемо проблему еквівалентності слів в асоціативних обчисленнях.

6.4.4. Проблема еквівалентності слів в асоціативних обчисленнях

Теорема 6.7 (Маркова - Поста). Існує асоціативне обчислення, в якому проблема розпізнавання еквівалентності слів алгоритмічно нерозв'язна.

Доведення. Нехай машина Тюрінґа M починає роботу зі слова R з початковою конфігурацією $q_0\alpha$ і закінчує роботу словом S з кінцевою конфігурацією $q_k\beta$. Проблема розпізнавання еквівалентності слів R і S в такій постановці полягає в знаходженні алгоритму, який за кодом машини M розпізнає, чи зупиниться вона в стані q_k, починаючи роботу з конфігурації $q_0\alpha$, чи ні. Подамо код цієї машини разом зі словом R на вхід універсальної машини M′: M′($d(M)*R$). Для машини M′ проблема зупинки нерозв'язна. Звідси випливає, що проблема розпізнавання еквівалентності слів також алгоритмічно нерозв'язна. Покажемо, що вона нерозв'язна також і в деякому асоціативному обчисленні. Для цього скористаємося тією властивістю, що для будь-якої машини Тюрінґа можна побудувати еквівалентне їй асоціативне обчислення.

Остання властивість була доведена для нормальних алгоритмів Маркова. Нормальний алгоритм Маркова можна розглядати як асоціативне обчислення з односпрямованими підстановками, що безпосередньо випливає з визначення цих систем. При необхідності для кожної підстановки нормального алгоритму можна визначити зворотну підстановку, і отримати асоціативне обчислення з двонаправленими підстановками.

Побудуємо асоціативне обчислення S(M), яке виконує той же алгоритм, що і машина M, зокрема, переробляє слово R в слово S, і приєднаємо до нього систему підстановок виду $q_ka_i \to q_k$. Отримаємо нове числення S'(M). У цьому численні також переробляються слова виду R в слова виду S, але всі заключні конфігурації M в S'(M) еквівалентні. Тому в S(M) слова $q_0\alpha$ і q_k еквівалентні, якщо і тільки якщо машина M, почавши роботу зі слова $q_0\alpha$, зупиниться. З огляду на нерозв'язності проблеми зупинки для M′, проблема еквівалентності слів $q_0\alpha$ і q_k також нерозв'язна.

З цієї теореми випливає, що проблема розпізнавання еквівалентності слів в загальному випадку (тобто для всієї множини асоціативних обчислень) алгоритмічно нерозв'язна. Звідси випливає також, що проблема

еквівалентності алгоритмів нерозв'язна: за двома заданими алгоритмами неможливо визначити, обчислюють вони одну і ту ж функцію чи ні.

Історично алгоритмічна нерозв'язність спочатку була встановлена для проблем, що виникають в математичній логіці, - проблем виводимості в формальних теоріях. Доведення нерозв'язності проблем самозастосовності і еквівалентності показало, що алгоритмічні схеми можуть служити апаратом дослідження можливості розв'язання і повноти формальних теорій. Проблему виводимості в формальної теорії можна інтерпретувати як проблему розпізнавання еквівалентності слів: за допомогою заданої системи правил виведення необхідно визначити, чи виводиться формула з заданої системи аксіом і вихідної множини посилок. Тоді з алгоритмічної нерозв'язності розпізнавання еквівалентності слів слідує нерозв'язність (в загальному випадку!) проблеми виводимості. Ми знаємо, що числення висловлювань розв’язна теорія: побудова таблиці істинності формули є алгоритмом, за допомогою якого доводиться, що формула є (або не є) тавтологією логіки висловлювань, а, отже, і теоремою числення висловлень. Обчислення предикатів 1-го порядку нерозв'язне, що було доведено для формальної арифметики *S* (теорема Геделя про неповноту). Доведення теореми Геделя істотно ґрунтувалося на рекурсивності функцій і відношень, що представимі в *S*. Отже, той же самий результат можна отримати, використовуючи апарат алгоритмічних схем, зокрема, машини Тюрінґа. У 1936 році цей результат: проблема розпізнавання виводимості – алгоритмічно нерозв'язна, був отриманий Чёрчем.

6.5. Рекурсивні та рекурсивно перелічувані множини

Визначення 6.2. *Рекурсивно перелічуваними* множинами є або порожня множина, або множина значень деякої функції $F(x)$.

Беручи тезу Черча, це можна інтерпретувати так, що існує машина Тюрінґа, яка, починаючи роботу з порожньої стрічки, виписує на неї всі значення деякої функції $F(x)$.

Визначення 6.3. Множина X є *рекурсивною,* якщо існує функція $f(x)$ така, що

$$f(x)=\begin{cases}1, \text{ якщо } x\in X\\ 0, \text{ якщо } x\notin X\end{cases}$$

Якщо приймати тезу Черча, то в термінах рекурсивної обчислюваності це означає, що існує машина Тюрінґа, яка для будь-якого x зупиняється в "так!" стані якщо x належить даній множині, і в "ні!" стані, якщо x не належить даній множині.

Тоді в термінах машини Тюрінґа ми можемо довести наступні теореми:

Теорема 6.8. Якщо R і S рекурсивно перелічувані, то $R \cup S$ і $R \cap S$ також рекурсивно перелічувані.

Доведення. З умови теореми випливає, що існують дві машини M_R і M_S, які породжують елементи цієї множини. Тоді можна побудувати машину Тюрінґа $M_{R \cup S}$: $r_1, s_1, r_2, s_2, \ldots$ - яка буде породжувати ці слова, видаляючи повторювані слова. Та $M_{R \cap S}$: r_i лише якщо існує таке s_j, що $r_i = s_j$.

Теорема 6.9. S рекурсивно тоді і тільки тоді, якщо S і його доповнення $S`$ рекурсивно перелічувано.

Доведення.

1. Припустимо, що $S \subset N$ і S рекурсивно. Тоді існує машина Тюрінґа M_S, яка для кожного натурального числа $n \in N$ розпізнає, чи належить воно множині S, чи ні. Тоді можна побудувати таку композицію, що, якщо $M_S(n)$ зупиняється в стані «так!», тобто $n \in S$, то починає роботу машина Тюрінґа M1, яка виписує на свою стрічку всі елементи S, а, якщо $M_S(n)$ зупиняється в стані «ні!», тобто $n \notin S$, то починає працювати M2, яка виписує всі елементи, які не належать S, на свою стрічку. Тоді машина M1 породжує все елементи S, а M2 - його доповнення S'. Згідно з визначенням 6.2, це означає, що S і його доповнення S' рекурсивно перелічуванії.

2. Припустимо, що S і S` рекурсивно перелічуванії. Отже, існують породжучі їх машини $M_S$ і $M_{S`}$. Тоді можна побудувати M(x), яка для кожного x буде запускати по черзі M_S і $M_{S`}$, і порівнювати x з черговим елементом $s_i$. Якщо $x = s_i$, породженому машиною $M_S$, то машина M(x) зупиняється в «так!»-стані, а, якщо машиною $M_{S`}$ - то в «ні!»-стані. Тоді машина M(x) працює як розпізнавальна машина Тюрінґа для обчислення предиката $x \in S$. Існування такої машини доводить, що множина S рекурсивна.

Теорема 6.10. Існують рекурсивно перелічувані, але не рекурсивні множини. Це означає, що S рекурсивно перелічуване, але не рекурсивно, тобто $S`$ **не** рекурсивно перелічуване.

Доведення. Припустимо, що всі рекурсивно перелічувані множини ми можемо переписати у вигляді списку (занумерувати): $S_1, S_2, \ldots, S_r, \ldots$ Тоді існує машина Тюрінґа M(n), така, що для кожного натурального числа n = 1, 2, 3, ... вона перевіряє, чи належить число n, відповідне індексу множини S_n, самому цій множині (тобто, $n \in S_n$), чи ні. Тоді, якщо $n \in S_n$, вона видає повідомлення «так» і записує це число в множину U, а, якщо $n \notin S_n$, то вона видає «ні», і записує число в множину U.

Машина Тюрінґа M j і и $j \in S_i$ — так $j \in U$

— ні $j \in U'$

Таким чином U буде містити деяку множину натуральних чисел, і буде рекурсивно перелічуваним, і U' - доповнення U.

Доведемо тепер, що це множина не рекурсивно перелічувана. Якщо $U`$ рекурсивно перелічувана, то $U' = S_{к.}$

Тоді М (к):
$$\begin{cases} \text{якщо } k \in S_k \Rightarrow k \in U \Rightarrow k \notin U' \Rightarrow k \notin S_{k.} \\ \text{якщо } k \notin S_k \Rightarrow k \in U' \Rightarrow k \in S_k \end{cases}$$

Таким чином, U' не є рекурсивно перелічуваною, і U' *не* рекурсивно, тому що неможливо побудувати машину Тюрінґа, яка б визначала $k \in U'$ чи ні для кожного k.

Алгоритмічна нерозв'язність машини Тюрінґа в термінах рекурсивних і рекурсивно перелічуваних множин означає, що множина всіх машин Тюрінґа, які не зупиняються на деякому слові, є не рекурсивною. А рекурсивно перелічуваною є множина всіх машин, які зупиняються, тобто закінчують свою роботу за кінцеве число кроків. Тоді повинні існувати необчислювані функції. Побудуємо таку функцію.

Нехай i - індекс машини Тюрінґа. Визначимо предикат $T(i, a, x)$ - i - індекс машини Тюрінґа, яка будучи застосовна до слова а, закінчує роботу в момент часу x, обчислюючи при цьому функцію $\varphi_i(a)$, x - гарантія зупинки машини. Цей предикат буде істинним або вирішуваним. Можна побудувати таку машину Тюрінґа М', яка для будь-якої машини буде визначати, чи є i правильним кодом машини Тюрінґа. Якщо i не є кодом, то М' - ні! і $T(i, a, x) = F$, якщо i є кодом, то М`-так! і $T(i, a, x) = T$.

Якщо приймати тезу Черча, то цей предикат вирішуваний в строгому сенсі, тобто можна побудувати таку машину Тюрінґа:

$$\tau(i, a, x) = \begin{cases} 0, \text{ якщо } T(i, a, x) = F \\ 1, \text{ якщо } T(i, a, x) = T \end{cases}$$

Питання до розділу 6

1. Визначення машини Тьюринга. Композиція машин Тьюринга.
2. Нормальні алгоритми Маркова. Еквівалентність нормальних алгоритмів і машини Тьюринга.
3. Клас функцій, обчислюваних за Тьюрингом. Теза Черча. Необчислювальні функції.
4. Універсальна машина Тьюринга. Алгоритмічно нерозв'язні проблеми.

7. ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ[i]

7.1. λ - числення Черча

7.1.1. Історія функціонального програмування

Функціональне програмування – розділ дискретної математики і парадигма програмування, в якій процес обчислення трактується як обчислення значень функцій в математичному розумінні (тобто тих, чий єдиний результат роботи полягає в повернутому значенні, або іншими словами, обчислення яких не має побічного ефекту). Протиставляється парадигмі імперативного програмування (відноситься в тому числі і машина Тюрінґа), в якій виконавцю програми *пропонується* послідовність виконуваних дій, в той час, як у функціональному програмуванні спосіб вирішення завдання *описується* за допомогою залежності функцій один від одного (в тому числі можливі рекурсивні залежності), але без вказівки послідовності кроків.

Однією з близьких парадигм програмування є логічне програмування (Prolog), в якому програма являє собою множину пар (логічна умова, нові факти). У логічному програмуванні, також, як і у функціональному програмуванні, програміст залишається в невіданні про методи, які застосовуються при обчисленні і послідовності виконання елементарних дій. Велика частина відповідальності за ефективність обчислень в логічному і функціональному програмуванні перекладається на «плечі» транслятора використовуваної мови програмування.

Історія функціонального програмування. Перед початком опису безпосередньо функціонального програмування слід звернутися до історії програмування взагалі. У 1940-х роках з'явилися перші цифрові комп'ютери, які програмувались перемиканням різного роду перемикачів, дротів і кнопок. Число таких перемикань досягало порядку декількох сотень і росло з ускладненням програм. Тому наступним кроком розвитку програмування стало створення всіляких ассемблерних мов з простою мнемонікою .

Але навіть асемблери не могли стати тим інструментом, яким змогли б користуватися багато людей, оскільки мнемокод все ще залишався занадто складними, а всякий асемблер був жорстко пов'язаний з архітектурою, на якій виконувався. Кроком після асемблера стали так звані імперативні мови високого рівня: Бейсік, Паскаль, Сі, Ada і інші, включаючи об'єктно-орієнтовані. Імперативними («розпорядчими») такі мови названі тому, що орієнтовані на послідовне виконання інструкцій, які працюють з пам'яттю (присвоювання), і ітеративні цикли. Виклики функцій і процедур, навіть рекурсивні, що не позбавляли такі мови від явної імперативності.

У парадигмі функціонального програмування наріжний камінь – це функція. Згадавши історію математики, можна оцінити вік поняття «функція»: йому вже близько чотирьохсот років, і математики придумали дуже багато теоретичних і практичних апаратів для оперування функціями, починаючи від звичайних операцій диференціювання і інтегрування, закінчуючи незрозумілими функціональними аналізами, теоріями нечітких множин і функцій комплексних змінних.

Математичні функції виражають зв'язок між вихідними даними і підсумковим продуктом деякого процесу. Процес обчислення також має вхід і вихід, тому функція – цілком підходящий і адекватний засіб опису обчислень. Саме цей простий принцип покладено в основу функціональної парадигми і функціонального стилю програмування. Функціональна програма являє собою набір визначень функцій. Функції визначаються через інші функції або рекурсивно через самих себе. При виконанні програми функції отримують параметри, обчислюють і повертають результат, при необхідності обчислюючи значення інших функцій. На функціональній мові програміст не повинен описувати порядок обчислень. Потрібно просто описати бажаний результат як систему функцій.

Як відомо, теоретичні основи імперативного програмування були закладені ще в 1930-х роках Аланом Тюрінґом і Джоном фон Нейманом. Теорія, покладена в основу функціонального підходу, також народилася в 20-х – 30-х роках. У числі розробників математичних основ функціонального програмування можна назвати Мозеса Шейнфінкеля і Хаскелла Каррі, які розробили комбінаторних логіку (обидва вони запропонували один і той же прийом, який згодом назвали *каррінг*, незважаючи на те, що Шейнфінкель запропонував його раніше. Вибачення математиків зводилися, в основному, до того, що термін *шейнфінкелінг* незручний для написання і вимови), і Алонзо Черча, творця саме λ-числення, яке дозволяло описувати значну кількість математичних об'єктів, використовуючи всього дві операції – абстракції та застосування функції. Ця, в принципі, перша функціональна мова програмування, тільки програмували на ньому не на комп'ютері, а на папері, і була практичною відповіддю на питання – як з логічних виразів вивести алгоритм.

Якраз в середині 1930-х логіки билися над цією проблемою, Тюрінґ теж був логіком. І створив свою машину після Черча, десь на півроку-рік пізніше. Якщо порівнювати два підходи - лямбда-числення і машину Тюрінґа, - то лямбда-числення моделює людини-обчислювача, у якого є нескінченний олівчик і нескінченна кількість паперу. Машина Тюрінґа ж до знарядь людини додає ще й ластик, тому він може дещо непотрібне стирати. Тюрінґ і Черч листувалися, і довго письмово розшаркувалися, кажучи

(Тюрінґ - Чёрчу) "як елегантна і мінімальна ваша побудова!" і (Черч - Тюрінґу) "як інтуїтивно зрозуміла ваша машина!"

Алонзо Черч (англ. *Alonzo Church*; 14 червня 1903, Вашингтон, США - 11 серпня 1995, Хадсон, Огайо, США) - американський математик і логік, який зробив внесок в основи інформатики. Отримав ступінь бакалавра в Прінстонському університеті в 1924 році, і кандидатську в 1927. Черч став професором математики в Прінстоні в 1929 році.

Черч прославився розробкою теорії лямбда-числень, що послідувала за його знаменитою статтею 1936 року, в якій він показав існування так званих «нерозв'язних задач». Ця стаття передувала знаменитому дослідженню Алана Тюрінґа на тему проблеми зависання, в якому також було продемонстровано існування задач, нерозв'язних механічними способами. Згодом Черч і Тюрінґ показали, що лямбда-числення і машина Тюрінґа мали однакові властивості, таким чином доводячи, що різні «механічні процеси обчислень» могли мати однакові можливості. Ця робота була оформлена як теза Черча - Тюрінґа.

Черч залишався професором математики в Прінстоні до 1967 року, після чого він переїхав до Каліфорнії. Поміж іншим, його система лямбда-числень лягла в основу функціональних мов програмування, зокрема сімейства Лісп (наприклад, Scheme).

Теорія так і залишалася теорією, поки на початку 1950-х років Джон МакКарті не розробив мову Лісп, яка стала першою майже функціональною мовою програмування і багато років залишалася єдиною такою. Хоча Лісп все ще використовується (як, наприклад, і Фортран), він вже не задовольняє деяким сучасним запитам, які змушують розробників програм звалювати якомога більшу ношу на компілятор, полегшивши так свою працю. Потреба в цьому, звичайно ж, виникла через все більш зростаючої складності програмного забезпечення.

У зв'язку з цією обставиною все більшу роль починає грати типізація. В кінці 70-х – початку 80-х років XX століття інтенсивно розробляються моделі типізації, які підходять для функціональних мов. Більшість цих моделей включали в себе підтримку таких потужних механізмів як абстракція даних і поліморфізм. З'являється безліч типізованих функціональних мов: ML, Scheme, Hope, Miranda, Clean і багато інших. До того ж постійно збільшується число діалектів.

Найбільш відомими мовами функціонального програмування є:

- LISP (Джон МакКарті, 1958, безліч його нащадків, найбільш сучасні з яких - Scheme і Common Lisp)
- ML (Робін Мілнер, 1979, з нині використовуваних діалектів відомі Standard ML і Objective CAML)
- Miranda (Девід Тернер, 1985, який згодом дав розвиток мови Haskell).

В результаті вийшло так, що практично кожна група, що займається функціональним програмуванням, використовувала власну мову. Це перешкоджало подальшому поширенню цих мов і породжувало багато більш дрібних проблем. Щоб виправити становище, об'єднана група провідних дослідників в області функціонального програмування вирішила відтворити гідності різних мов в новій універсальній функціональній мові. Перша реалізація цієї мови, названої Haskell в честь Хаскелла Каррі, була створена на початку 90-х років.

Більшість функціональних мов програмування реалізуються як інтерпретуємі, дотримуючись традицій Лиспа. Такі є зручними для швидкого налагодження програм, виключаючи тривалу фазу компіляції, скорочуючи звичайний цикл розробки. З іншого боку, інтерпретатори в порівнянні з компіляторами зазвичай програють по швидкості виконання. Тому крім інтерпретаторів існують і компілятори, які генерують непоганий машинний код (наприклад, Objective Caml) або код на Ci/Ci++ (наприклад, Glasgow Haskell Compiler). Що показово, практично кожен компілятор з функціональної мови реалізований цією ж самою мовою.

7.1.2. Лямбда-числення

Лямбда-числення (*λ-числення*) – розділ дискретної математики, що вивчає процес обчислення, як математичний процес; є теоретичним механізмом для багатьох прикладних наук, в першу чергу, для функціонального програмування. Основна методика вивчення – дослідження *застосовності* (калька з англійського - *application*) операндів до функції, при цьому останні представляються у вигляді λ-термів:

λxy.(x + y) - функція складання двох своїх операндів.

λx.xx λx.xx - класичний приклад для ілюстрації нескінченної редукції. Під редукцією розуміється власне процес обчислення результату функції після застосування до неї всіх необхідних операндів.

Лямбда-числення – це правила побудови і обчислення безіменних функцій. Ось запис функції, яка отримує на вхід число і обчислює значення, більше його на одиницю:

(λx.x + 1)

Сама функція записана в дужках: до точки - аргументи функції, а після точки -вираз. У Haskell замість λ використовується символ '\'.

Застосування функцій до аргументів записується ось так:

(λx.x + 1) 2

Ми можемо спрощувати записаний нами вираз, підставляючи замість змінних їх значення (бета-спрощення, бета-редукція):

(λx.x+1)2 – застосовуємо бета-редукцію, підстановку значень аргументів -> (2 + 1) – застосовуємо дельта-редукцію, тобто, обчислення функції від аргументів -> 3

Розглянемо функцію від двох аргументів – суму x і y:

(λ x y.x + y)

(λ x y.x + y) 1 - застосування бета-редукції, замінивши x на 1 -> (λ y.1 + y)

Отримали іншу функцію, від меншої кількості змінних.

Запишемо: *(λ x y.x + y) еквівалентно (λ x. (λ y.x + y)).*

Саме цей прийом називається *каррінг* (currying) - звуження простору змінних.

7.1.3. Обчислення функціональних програм

У програмуванні нормальна стратегія відповідає викликом на ім'я. Тобто аргумент виразу не обчислюється до тих пір, поки до нього не виникне звернення в тілі вираження. Аплікативна редукционная стратегія відповідає викликом за значенням.

Робота інтерпретатора описується декількома кроками:

1. У вираженні необхідно виділити деяке звернення до рекурсивної або вбудованої функції з повністю зазначеними аргументами. Якщо виділене звернення до вбудованої функції існує, то відбувається його виконання і повернення до початку першого кроку.

2. Якщо виділене на першому кроці звернення до рекурсивної функції, то замість нього підставляється тіло функції з фактичними параметрами (тому що вони вже означена). Далі відбувається перехід на початок першого кроку.

3. Якщо більше звернень немає, то відбувається зупинка.

В принципі, обчислення в функціональній парадигмі повторюють кроки редукції, але додатково містять обчислення вбудованих функцій.

Можна бачити, що будь-яке рекурсивне визначення деякої функції f можна представити в такому вигляді: f = F f

Цей вислів можна трактувати як рівняння, в якому рекурсивна функція f є нерухомою точкою функціоналу F. Відповідно інтерпретатор функціональної мови можна в такому ж ключі розглядати як чисельний метод вирішення цього рівняння.

Можна зробити припущення, що цей чисельний метод (тобто інтерпретатор) в свою чергу може бути реалізований за допомогою деякої функції Y, яка для функціоналу F знаходить його нерухому точку (відповідно визначаючи шукану функцію): f = Y F.

Програми на функціональних мовами зазвичай коротше і простіше, ніж ті ж самі програми на імперативних мовах. Як приклад розглянемо *швидке сортування Хоара на мові Haskell:*

```
quickSort [] = []
quickSort(x:xs)=quickSort[y| y <- xs, y <= x] ++ [x] ++ quickSort [y | y <- xs, y> x]
```

Це читається так:

1) Якщо список порожній, то результатом також буде порожній список.

2) Інакше виділяється голова (перший елемент) і хвіст (список з решти елементів, який може бути порожнім). У цьому випадку результатом буде конкатенація відсортованого списку з усіх елементів хвоста менших або рівних голові, списку з самої голови і списку з усіх елементів хвоста більших голови.

Як видно, навіть на такому простому прикладі функціональний стиль програмування виграє і за кількістю написаного коду, і по його людяності.

Крім того, всі операції з пам'яттю виконуються автоматично. При створенні будь-якого об'єкта під нього автоматично виділяється пам'ять. Коли об'єкт виконає своє призначення, він незабаром буде також автоматично знищений збирачем сміття, який є в будь-якій функціональній мові.

Ще один засіб, що дозволяє скоротити програму, – вбудований механізм зіставлення зі зразком. З ним можна описувати функції як індуктивні визначення.

Визначення N-ого числа Фібоначчі

fibb (0) = 1

fibb (1) = 1

fibb (N) = fibb (N - 2) + fibb (N - 1)

Завдяки механізму зіставлення зі зразком функціональні мови виходять на більш абстрактний рівень, ніж традиційні імперативні мови (тут не розглядається об'єктно-орієнтована парадигма і її розширення).

У функціональних мовах, так само як і взагалі в мовах програмування і математиці, функції можуть бути передані іншим функціям як аргументи або повернуті в якості результату. Функції, які беруть функціональні аргументи, називаються функціями вищих порядків або функціоналами. Найбільш, мабуть, відомий функціонал – функція map. Вона застосовує деяку функцію до всіх елементів списку, формуючи з результатів заданої функції інший список. Наприклад, визначивши функцію зведення цілого числа в квадрат як:

square (N) = N * N

Можна скористатися функцією map для зведення в квадрат всіх елементів деякого списку:

squareList = map (square, [1, 2, 3, 4])

Результатом буде список [1, 4, 9, 16].

В імперативних мовах функція в процесі свого виконання може читати і змінювати значення глобальних змінних і здійснювати операції введення або виводу. Тому, якщо викликати одну й ту ж функцію двічі з одним і тим же аргументом, може статися так, що в якості результату будуть обчислені два різних значення. Така функція називається функцією з побічними ефектами.

У чистому функціональному програмуванні оператор присвоювання відсутній, об'єкти не можна змінювати і знищувати, можна тільки створювати нові шляхом розбору і збору існуючих.

Крім спрощення аналізу програм є ще одна корисна властивість - паралелізм. Раз всі функції для обчислень використовують тільки свої параметри, ми можемо обчислювати незалежні функції в довільному порядку або паралельно, на результат обчислень це не вплине. Причому паралелізм цей може бути організований не тільки на рівні компілятора мови, а й на рівні архітектури.

Приклади. Розглянемо кілька прикладів представлення функції за допомогою λ-абстрактора, що дозволяє задавати функцію, вказуючи її ім'я і аргумент такий спосіб: $\lambda x.\ f(x)$. λ-абстрактор вказує, від якої змінної залежить функція (Черч запропонував розглядати функції як правила, при цьому перехід від аргументу до значення функції здійснюється за допомогою певного процесу). Значення функції обчислюється (дельта-редукція) за допомогою підстановки (бета-редукція):

Наприклад, нехай $f(x) = ax + b$.

$\lambda b\ .\ ax + b$ визначає цю функцію як залежну від b.

Підстановка ($\lambda b\ .\ ax + b$) (3) дає множину функцій ax +3.

$\lambda a\ .\ ax + b$ визначає цю функцію як залежну від a.

Підстановка (λa .a$Ax + b$) (3) дає множину функцій $3x + b$.

$\lambda x\ .\ ax + b$ визначає цю функцію як залежну від x.

Підстановка ($\lambda x\ .\ ax + b$) (3) дає множину функцій $3\ a + b$.

Можна визначити функцію від декількох змінних, наприклад,

$\lambda x\ .\ (\lambda a\ .\ (\lambda b\ .\ ax + b))$ визначає цю функцію, як залежну від x, a, b.

Підстановка ($\lambda x\ .\ (\lambda a\ .\ (\lambda b\ .$ a$x + b)\ (5))\ (2))\ (3)$ дає значення $2_x3 + 5 = 11$.

За допомогою λ-абстрактора можна задавати і предикати, наприклад, $\lambda x\ .\ (x > 1)$.

7.1.4. Основні особливості функціонального програмування

1. *Виділяється базовий набір функцій*. Наприклад, елементарні арифметичні функції, предикати рівності і нерівності. Далі ми розглянемо базовий набір функцій для обробки списків.
2. *Нові функції конструюються з базових* в допомогою λ-абстрактора, операцій аплікації (застосування функції до аргументу), *композиції функцій і рекурсії*.
3. Функції обчислюються шляхом *підстановки аргументів замість змінних*, зазначених λ-абстракторе.
4. Кожен вираз єдиним чином визначається значенням його складових частин (тобто не залежить від контексту, відсутній побічний ефект, коли значення аргументу змінюється після обчислення функції).
5. *Складні структури даних можуть виступати як одиниці даних.* Наприклад, список може є одиницею даних в мові Lisp. Точно так само

можна визначити функціональний мову, де одиницею даних будуть матриці, множини і т.п.

6. *У функціональному програмуванні відсутній оператор присвоювання.* Функції визначаються за допомогою λ-абстрактора, при цьому вони можуть бути іменованими чи неіменованими. Значення функції обчислюються за допомогою підстановки (причому результатом не обов'язково є число, це може бути нова функція), наприклад,

$g = \lambda x .2 * x +3$ (функції присвоєно ім'я g).

g (10) – обчислюється значення g: 2 * 10 + 3 = 23.

Та ж функція, але неіменована:

($\lambda x .2 * x +3$) (10) обчислює те ж саме значення.

Підстановка функції замість аргументу: ($\lambda x .2 * x +3$) (5 * y) дає нову функцію: 2 * 5 * y +3. Ця операція називається *аплікацією* функцій.

7. *У функціональному програмуванні використовується умовний вираз виду*:

if <вираз> *then* <вираз> *else* <вираз>;

if <вираз> *then* <вираз>.

наприклад:

$h = \lambda x$. *if* ($x > 10$) *then* g (x) *else* f (x)

$f = \lambda x . x + y$

$g = \lambda x .2 * x +3$

Тоді h (5) – це функція f (5), тобто 5+ y.

8. Можна визначити *композицію функцій*, використовуючи λ-абстрактор по іменах функцій: $\lambda f . (\lambda g (\lambda x. F(g(x)))$.

Тоді при підстановці в це вираження визначених раніше функцій

($\lambda f . (\lambda g . (\lambda x . F (g (x))))$ (f, g) отримаємо нову функцію $\lambda x .2 * x +3+ y$.

9. *Оператор циклу відсутній* в строго функціональній мові. Замість нього *застосовується рекурсія.*

Рекурсія буває двох видів: висхідна і спадна.

Розглянемо їх на прикладі функції факторіалу:

$$F(x) = x! = \begin{cases} 1, \text{ якщо } x = 0, \\ x * f(x-1), \text{ якщо } x > 0. \end{cases}$$

Функціональна програма:

Factorial = λx . *if* $x = 0$ *then* 1 *else* x * *Factorial* (x -1).

Процес обчислення:

Factorial(3) = 3 * *Factorial* (2) = 3 * 2 * *Factorial* (1) = 3 * 2 * 1 * *Factorial* (0) = = 3 * 2 * 1 * 1 = 6.

Це *спадна рекурсія*. Проміжні значення заносяться в стек, а коли буде обчислено термінальне значення (умова закінчення), то результати вибираються з стека і обчислюється остаточний результат.

Висхідна рекурсія використовує накопичувальний параметр. Вона визначається за допомогою двох функцій. Одна функція задає початкове значення накопичувального параметра:

$F1 = \lambda x\ .\ fact\ (x, 1)$,

інша функція рекурсивно обчислює факторіал:

$fact\ (x, y) = \lambda x\ .\ \lambda y\ .\ if\ x = 0\ then\ y\ else\ fact\ (x - 1, y * x)$.

Для обчислення 3! потрібно викликати функцію $F1(3)$, $F1$ присвоїть накопичувальному параметру y значення 1 і викличе функцію $fact(x, y)$, передавши в неї значення $x = 3$, $y = 1$:

$F1(3) = fact(3, 1) = fact(2, 1 * 3) = fact(1, 3 * 2) = fact\ (0, 6 * 1) = 6$.

7.2. Дерево представлення виразу та польський запис

7.2.1. Дерево розбору арифметичного виразу

У зв'язку з акцентованим увагою до виконання функцій, розглянемо *представлення складної функції у вигляді суперпозиції простих:*

*f = a + b * c - a / (a + b) = add (a, minus (mult (b, c), del (a, add (a, b))))*.

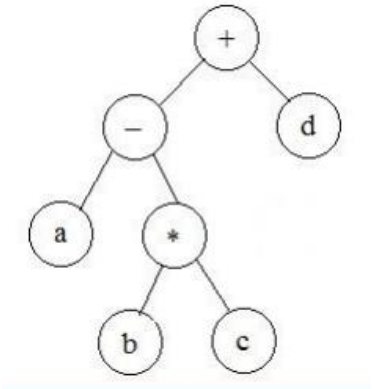
Операції - бінарні, будемо вважати, що ділення існує; можна було побудувати суперпозицію і таким чином:

minus (add (a, mult (b, c)), del (a, add (a, b))).

Можна уявити арифметичний вираз у вигляді дерева. Згадаймо, що дерево - це граф T = (V, E) без циклів, тут V - множина вершин, E – множина ребер. Одна вершина $v \in V$ в дереві виділена, і називається кореневою вершиною. Лист - вершина, яка не має нащадків.

Бінарне дерево (дерево, в якому кожна вершина має не більше двох дуг, що виходять) якраз є одним з варіантів зберігання арифметичного виразу - так як будь-яка проста арифметична операція є бінарної. Листя такого дерева містять операнди (числа або змінні), а внутрішні вершини і корінь - знаки операцій.

Бінарне дерево для вираження a - b * c + d представлено на рисунку. Обчислення виразу здійснюється за алгоритмом, що формується на підставі обходу дерева.

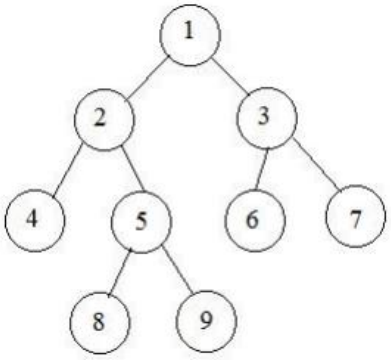


Обхід дерева – це спосіб послідовного відвідування вузлів дерева, при якому кожен вузол відвідується тільки один раз.

*Дерева можна обходити різними способами***.** Їх можна обходити в глибину або ширину[11].

Пошук називаються пошуком *в глибину*, якщо по дереву пошуку просуватися дуже глибоко для кожного нащадка, перед тим, як перейти до наступного листа.

Дерева можна обходити також в порядку рівнів, де відвідують кожен вузол на одному рівні перш ніж перейти на наступний рівень. Такий пошук (обхід) називається пошуком *в ширину*.



Алгоритм *обходу дерева в ширину* заснований на структурі даних черга.

Крок 0. Помістити в чергу корінь дерева.

Крок 1. Вилучити з черги чергову вершину. Помістити в чергу її дочірні вершини по порядку зліва направо (справа наліво).

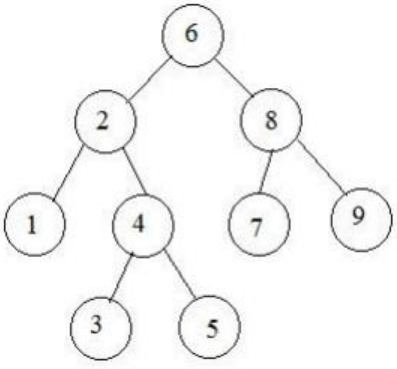
Крок 2. Якщо черга порожня, то кінець обходу, інакше перейти на крок 1

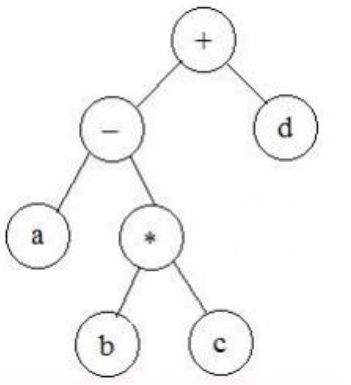
Обходи в глибину

Існує три способи обходу дерева в глибину:

1. прямий (корінь-ліво-право), Pre_order, префіксний запис;

2. поперечний – серединний, центрований (ліво-корінь-право), In_order; інфіксний запис (традиційний);





Поперечний обхід виразу

a–b*c+d;

+-a*bcd

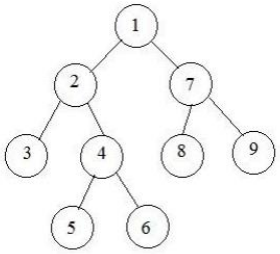
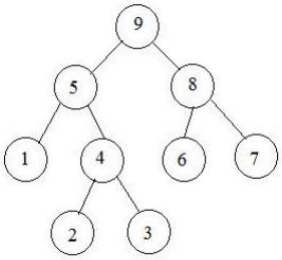
abc*-d+

відповідає інфіксной формі запису.

3. зворотний (ліво-право-корінь), Post_order; постфіксний запис.

[11] Існують також три алгоритму обходу, що не класифікуються ні як пошук в глибину, ні як пошук в ширину. Один з таких алгоритмів - метод Монте-Карло , який зосереджується на аналізі найбільш обіцяють ходів, грунтуючись на розширенні дерева пошуку при випадковому виборі простору пошуку.

Продемонструємо обхід зверху вниз (прямий обхід - 1) і від низу до верху (зворотний обхід - 3).

Порядок обходу вершин при прямому обході	Порядок обходу вершин при зворотному обході
	

З варіантами обходу дерева обчислення арифметичного виразу пов'язані поняття прямого і зворотного польського запису.

7.2.2. Польські нотації Лукасевича

Польський запис - це альтернативний спосіб запису арифметичних виразів, перевага якого полягає у відсутності дужок. Існує *два типи польської записи: пряма і зворотна, також відомі як префіксна і постфіксна*. Відмінність їх від класичного, *інфіксне* способу полягає в тому, що знаки операцій пишуться не між, а, відповідно, до або після аргументів.

Інфіксний запис	*Прямий (префіксний) польський запис*	*Зворотний (постфіксний) польський запис*
(3 + 5) * 7	* + 3 5 7	3 5 +7 *
(3 * 5) + 7	+ * 3 5 7	3 5 * 7 +
3 + (5 * 7)	+ 3 * 5 7	3 5 7 * +
3 * (5 + 7)	* 3 +5 7	3 5 7 + *

Зворотній польський запис видається більш простий для розуміння, але більш складнм для реалізації, ніж прямий.

Польський логік Ян Лукасевич[12] винайшов *пряму польську нотацію* (вірніше він назвав її просто польської, а називатися прямою або префіксною вона стала після винаходу зворотного постфіксного запису) приблизно в 1920, щоб спростити пропозіціональному логіку. Характерна риса такого запису - *оператор розташовується зліва (перед);* цьому виду запису відповідає *обхід дерева зверху вниз.*

Алонзо Черч згадував цю нотацію в своїй класичній книзі по математичній логіці як гідну уваги систему. Незважаючи на те, що польський запис не використовується в математиці, він широко застосовується в інформатиці.

Таблиця, наведена нижче, демонструє ядро запису, запропонованого Яном Лукасевичем для пропозіціональної логіки. Деякі літери польського запису означають конкретні слова польською мовою:

поняття	*логічна нотація*	*польська нотація*	*польське слово*
заперечення	¬φ	Nφ	*negacja*
кон'юнкція	φ&ψ	Kφψ	*koniunkcja*
диз'юнкція	φVψ	Aφψ	*alternatywa*
імплікація	φ→ψ	Cφψ	
рівність	φ≡ψ	Eφψ	*ekwiwalencja*
штрих Шеффера	φ|ψ	Dφψ	*dysjunkcja*
можливість	◊φ	Mφ	*możliwość*
необхідність	□φ	Lφ	
квантор загальності	∀φ	Πφ	
квантор існування	∃φ	Σφ	

Зворотний польський запис (англ. Reverse Polish notation, RPN) - форма запису математичних і логічних виразів, в якій *операнди розташовані перед знаками операцій*. Також іменується як зворотний бездужковий

[12] Ян Лукасевич (пол. Jan Łukasiewicz, 21 грудня 1878 Львів - 13 листопада 1956, Дублін) – польський логік і математик, член Польської академії наук (1937), один з головних представників львівсько-варшавської школи.

Працював в області логічних проблем індукції та причинності і логічних підстав теорії ймовірностей. Побудував першу систему багатозначної логіки, а з її допомогою – систему модальної логіки. Розробив оригінальну мову для формалізації логічних виразів (т.з. польський запис, який послужив основою для більш відомого зворотного польського запису). За філософських поглядах – позитивіст.

запис, *постфіксний* нотація, бездужкова символіка Лукасевича, польський інверсний запис, полізім. Автоматизація обчислення виразів в зворотної польської нотації заснована на *використанні стека;* цьому виду запису відповідає *обхід дерева від низу до верху.*

Зворотна польська нотація (ЗПН) була розроблена австралійським філософом і фахівцем в області теорії обчислювальних машин Чарльзом Хемблін в середині 1950-х рр. на основі польської нотації, яка була запропонована в 1920 році польським математиком Яном Лукасевичем.

Наприклад, розглянемо обчислення виразу, що задано в ЗПН: 7 2 3 * – (еквівалентний вираз в інфіксной нотації: 7 – 2 * 3).

1. Перший по порядку знак операції - «*», тому першою виконується операція множення над операндами 2 і 3 (вони стоять останніми перед знаком). Вираз при цьому перетвориться до виду 7 6 – (результат множення 6, замінює трійку «2 3 *»).

2. Другий знак операції - «–». Виконується операція віднімання над операндами 7 і 6.

3. Обчислення закінчено. Результат останньої операції дорівнює 1, це і є результат обчислення виразу.

Очевидне розширення зворотного польського запису на унарні, тернарниі і операції з будь-якою іншою кількістю операндів: при використанні знаків таких операцій в обчисленні виразу операція застосовується до відповідного числа останніх операндів.

Особливості зворотної польської записи наступні[13]*:*

• Порядок виконання операцій однозначно задається порядком проходження знаків операцій у виразі, тому відпадає необхідність використання дужок і введення пріоритетів і асоціативності операцій.

• На відміну від інфіксного запису, неможливо використовувати одні і ті ж знаки для запису унарних і бінарних операцій. Так, в інфіксному запису вираз 5 * (-3 + 8) використовує знак «мінус» як символ унарною операції (зміна знака числа), а вираз (10 - 15) * 3 застосовує цей же знак для позначення бінарної операції (віднімання). Конкретна операція визначається тим, в якій позиції знаходиться знак. Зворотний польський запис (і прямий теж) не дозволяє цього: запис 5 3 - 8 + * (умовний аналог першого виразу) буде інтерпретований, як помилковий, оскільки неможливо визначити, що «мінус» після 5 і 3 позначає не віднімання; в результаті буде зроблена спроба обчислити спочатку 5 - 3, потім 2 + 8, після чого з'ясується, що для операції множення не вистачає операндів. Щоб все ж записати цей вислів, доведеться або переформулювати його (наприклад, записавши замість виразу - 3 вираз 0 - 3), або ввести для операції зміни знака окреме позначення, наприклад, «±» 5 3 ± 8 + *.

---

[13] Власне кажучи, для прямої теж.

• Так само, як і в інфіксной нотації, в ЗПН одне і те ж обчислення може бути записано в кількох різних варіантах. Наприклад, вираз (10 - 15) * 3 в ЗПН можна записати як 10 15 - 3 *, а можна - як 3 10 15 - *

• Через відсутність дужок зворотний польський запис коротше інфіксной. За цей рахунок при обчисленнях на калькуляторах підвищується швидкість роботи оператора (зменшується кількість натискань на клавіші), а в програмованих пристроях скорочується обсяг тих частин програми, які описують обчислення.

Едсгер Дейкстра винайшов *алгоритм для перетворення виразів з інфіксной нотації в ЗПН*. Алгоритм отримав назву «сортувальна станція», за схожість його операцій з тими, що відбуваються на залізничних сортувальних станціях.

Приклади.

інфіксний запис	*прямий (префіксний) польський запис*	*зворотний (постфіксний) польський запис*
(1+2)+3*4*(5/6)-(7-8)	++1 2 –* 3 * 4 /5 6 –7 8	1 2+ 3 4 5 6 /* * 7 8 – – +
$(a+b)/(c+d)$	/+ a b + c d	$ab+cd+/$
$a+b*c-a/(a+b)$	–+ a* b c / a + a b	$abc*+aab+/-$
(a+b)*c^2 або (a+b)*c^2	*+a b * c c або *+a b ^ c 2	a b + c c * * або a b + c 2 ^ *

Дерева розбору арифметичного виразу для наведених прикладів пропонується побудувати самостійно.

7.3. Функціональна мова обробки списків

7.3.1. Базові оператори та функції

Визначення 7.1. Назвемо *атомом* послідовність символів (рядок) або число. Наприклад: *test* , 15, чорний, дерево, -25 - атоми.

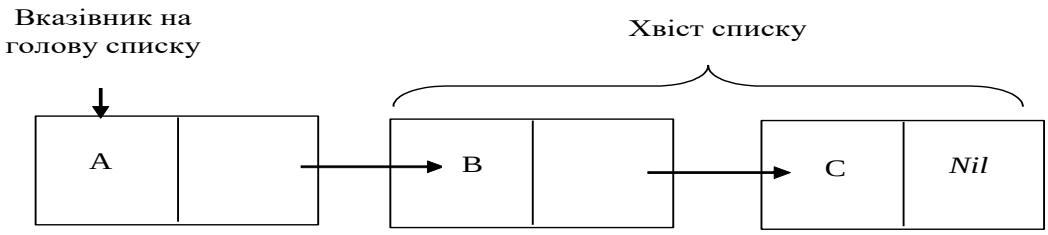
Визначення 7.2.

1) Атом, укладений в дужки, є *список;*

2) Послідовність *списків* , укладена в дужки, є *списком.*

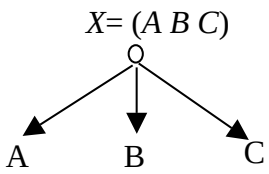
Таким чином: (25 17 16 15 2), (дерево), ((білий пішак) (4 7)) - списки.

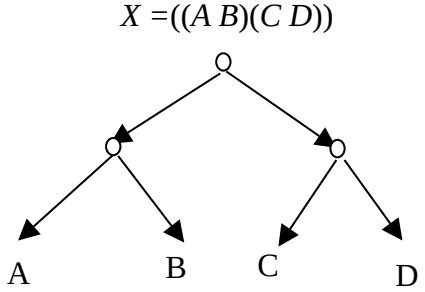
В пам'яті машини список можна представити як послідовність пов'язаних одна з одною чарунок, що складаються з двох частин: інформаційної частини і вказівника на наступний елемент. Перший елемент списку називається його головою, інші елементи - хвостом. Список *X* = (*A B C*) можна схематично зобразити так:



При обробці списків як аргумента списку передається вказівник на голову списку. Якщо значення, що повертається, теж список, то це вказівник на список, в інших випадках це може бути значення інформаційної частини (атом).

Список можна зобразити як дерево:

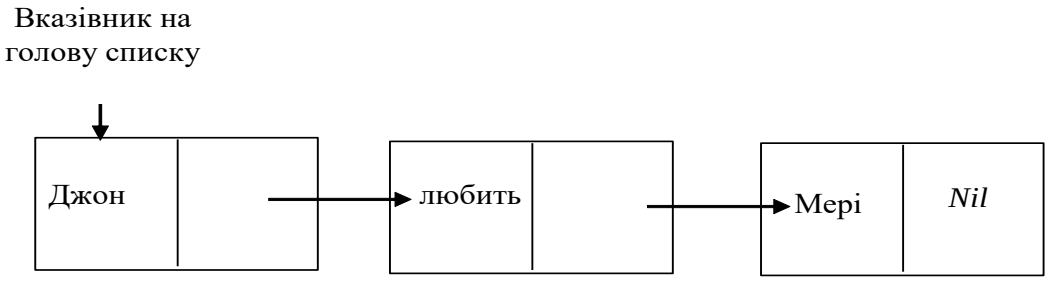




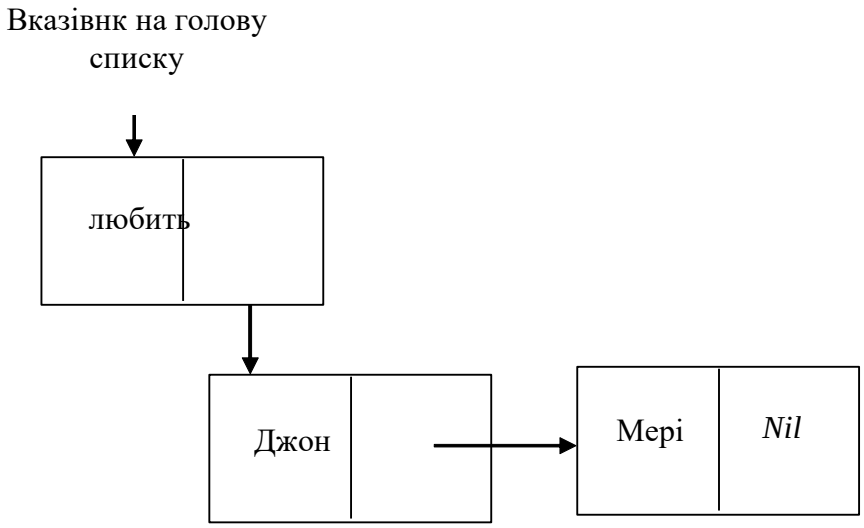
Пусте значення вказівника позначається *Nil*. Якщо список не містить елементів, то значення вказівника на голову списку – *Nil*. Такий список називається порожнім і позначається: *X = Nil*.

Для чого потрібні списки? Для подання послідовностей символів і слів, які використовуються для представлення знань.

Наприклад, висловлювання «Джон любить Мері» представимо двомісним предикатом «*x любить y*». Це висловлювання можна представити лінійним списком:



Але це не дуже вдала структура. Набагато ефективніше буде подання у вигляді дерева:



Слід, однак, відзначити, що списки використовувалися як основна структура при поданні знань (як в Ліспі, так і в Прологі) тільки до тих пір, поки не з'явилися об'єктно-орієнтовані мови. Коли з'явилися С ++, середа Delphy і інші об'єктно-орієнтовані середовища, про Ліспі і Пролог успішно забули.

Елементарними функціями обробки списків є такі.

1. Функція ***Car (X)*** вибирає зі списку **1-й** елемент.

X	Car (X)
(*A B*)	*A*
((*A B*) (*C D*))	(*A B*)
(*A*)	*A*

Аргумент *X* може бути тільки списком, а результат - 1-й елемент – може бути або атомом або списком .

Функція *Car* (*X*) для аргументу-атома не визначена.

Приклади використання функції наведені в таблиці. Якщо *X* – список (*A B*), то *Car*(*X*) – голова списку – інформаційне значення *A*. Якщо *X* – список, складений зі списків (тобто дерево) ((*A B*) (*C D*)), то *Car* (*X*) – голова списку – список (піддерево) (*A B*).

2. Функція ***Cdr (X)*** від списку є список, отриманий відкиданням 1-го члена. Функція *Cdr* (*X*) повертає вказівник на другий елемент списку, тобто на **хвіст списку**.

X	Cdr (X)
(*A B*)	(*B*)
((*A B*) (*C D*))	((*C D*))
(*A*)	*Nil*

Cdr (*X*) для атома не визначена. *Car* і *Cdr* – це функції-селектори.

Використовуючи аплікацію цієї функції до себе: *Cdr* (... (*Cdr* (*X*))), можна отримати вказівник на другий, третій і т.д. елементи списку. Таким чином можна отримати доступ до будь-якого елементу списку. Як бачимо, список – це структура з послідовним доступом. Наприклад, вираз *Car* (*Cdr* (*Cdr* (*X*))) матиме значенням 3-й елемент списку. Можна визначити нову функцію: *третій* (*X*): = *Car* (*Cdr* (*Cdr* (*X*))).

3. Функція ***Cons* (*X, Y*)** – конструктор для створення списків. Він **додає аргумент *X* в голову списку *Y*.**

X	Y	Cons (X, Y)
A	(*B C*)	(*A B C*)
(*A B*)	((*C D*))	((*A B*) (*C D*))
(*A B*)	(*C D*)	((*A B*) *C D*)
A	*Nil*	(*A*)

Аргумент *X* може бути списком або атомом, *Y* - обов'язково список.

Результат - список, отриманий з *Y* додаванням ***X* в якості першого елемента**. Таким чином, довжина результату дорівнює *n* +1, якщо *Y* - список довжини *n*. Основний концепт: **Cons (Car (X), Cdr (X)) = X.**

4. Функція ***Atom* (*X*)** – предикат. Він дорівнює *T*, якщо *X* є атомом, і *F*[14], якщо *X* – список.

X	Atom (X)
A	*T*
(*A*)	*F*
(*A B C*)	*F*
Nil	*T*
127	*T*
(127)	*F*

Служить для розпізнавання списків і атомів.

Наприклад, у функції *f* (*x*): = *if atom* (*x*) *then Nil else car* (*x*), – *f* (*x*) видає 1-й елемент *x* , якщо *x* - список, і видає *Nil* , якщо *x* - атом.

Таким чином, якщо *Car* (*x*) - частково-визначена функція, то *f* (*x*) - усюди визначена функція.

5. Функція ***EQ* (*x , y*)** – предикат, який порівнює атоми.

X	Y	EQ (X , Y)
A	*A*	*T*

[14] Взагалі в самому ЛІСП існують такі позначення: істинна - T, хибно - Nil. Тобто Nil - це і хибно, і порожній список, і порожній вказівник.

A	*B*	*F*
(*A, B*)	*B*	*F*
Nil	*Nil*	*T*
A	(*B*)	*F*
(*A*)	(*B*)	не визначено

Він дорівнює *T*, якщо два атома рівні, *F* – якщо атоми різні або один з аргументів є списком.

Якщо *X, Y* – обидва списки, то результат не визначений.

6. Функція ***NULL (X)*** – предикат, який повертає *T*, якщо аргумент - список порожній, і *F*, якщо список не порожній. Для атома – не визначений.

X	NULL (X)
(*A B*)	*F*
Nil	*T*
( )	*T*
A	не визначено

Тільки для числових атомів будемо використовувати операції: +, –, *, *div*, *mod*, і предикати $x \le y$, $x = y$ *і т.і.*.

З цих елементарних функцій можна будувати більш складні функції обробки списків.

Розглянемо деякі з них.

7.3.2. Рекурсивні функції обробки списків

Приклад 1.

Визначимо функцію *Довжина* (X), яка обчислює кількість елементів у списку. Потрібно розглянути два випадки: $X = Nil$ і $X \neq Nil$.

Якщо список порожній: $X = Nil$, то його довжина дорівнює 0.

Якщо у списку не порожній: $X \neq Nil$, то довжину його можна підрахувати, відкинувши спочатку 1-й елемент, потім 2-й, і т. д., до тих пір, поки *X* не стане дорівнювати *Nil,* при цьому потрібно підраховувати, скільки раз це було зроблено.

$$Довжина(X) = \begin{cases} =0, якщо\ X = Nil \\ = Довжина(Cdr(X))+1, якщо\ \ X \neq Nil \end{cases}$$

Функція обчислення довжини списку:

Довжина (*X*): = *if Null (X) then* 0 *else Довжина* (*Cdr* (*X*)) + 1.

Це був варіант з низхідній рекурсією. Побудуємо функцію, яка використовує висхідну рекурсію:

ДовжВ (X): = Довж (X, 0);

Довж (X, y): = if X = Nil then y else Довж (Cdr (X), y + 1).

Приклад 2.

Визначимо суму елементів числового списку.

$$Сума(X) = \begin{cases} =0, якщо\ X = Nil \\ = Car(X)+Сума(Cdr(X)), якщо\ \ X \neq Nil \end{cases}$$

Сума (*X*): = *if Null (X) then* 0 *else Car* (*X*) + *Сума* (*Cdr* (*X*)).

Наприклад, для списку *X* = (3 5 7) функція буде виконуватися так: *Сума* (3 5 7) = 3 + *сума* (5 7) = 3 + 5 + *Сума* (7) = 3 + 5 + 7 + *Сума* (*Nil*) = 3 + 5 + 7 + 0 = 15. Це – спадна, низхідна рекурсія.

Висхідна рекурсія:

СумаВ (X): = См (X, 0);

См (X, y): = if Null (X) then y else См (CDR (X), Car (X) + y).

СумаВ(3 5 7)=См((3 5 7), 0)=См((5 7), 3 + 0)=См((7), 5 + 3)=См(( ), 7 + 8)=15.

Приклад 3.

Визначимо функцію *З'єднати* (*X* , *Y*), де *X* , *Y* - списки (конкатенація). Її можна визначити по різному.

1 спосіб.

З'єднати(*X*, *Y*): = *if Null (X) then Y else Cons*(*Car*(*X*), *З'єднати*(Cdr(*X*), *Y*)).

Наприклад, обчислимо функцію для *X* = (*A B*), *Y* = (*D F*).

З'єднати(*X*, *Y*)=*Cons*(*A*, З'*єднати*((*B*), *Y*))=

=*Cons*(*A*,*Cons*(*B*, З'*єднати*(*Nil*, *Y*)) =

=*Cons*(*A*,*Cons*(*B*,*Y*))=*Cons*(*A*,(*B D F*))=(*A B D F*).

2 спосіб.

Можна ввести допоміжну функцію:

З'єднати (X, Y): = if Null (Y) then X else Конкат (X, Y);

Конкат (X, Y): = if Null (X) then Y else Cons (Car (X), Конкат (Cdr (X), Y)).

Тут застосовується принцип структурного програмування: виділення підфункцій, так що *Конкат* (*X*, *Y*) служить як би підпрограмою для *З'єднати* (*X*, *Y*) - це висхідна рекурсія, а попередній спосіб був реалізований за допомогою низхідній рекурсії.

Приклад 4.

Розглянемо функцію обернення списку. Нехай *X* – список, що обертається, а *Y* - параметр, що накопичує обернений список (це буде висхідна рекурсія).

Обр (*X, Y*): = *if Null (X) then Y else Обр* (*Cdr* (*X*), *Cons* (*Car* (*X*), *Y*));

Обернути (*X*): = *Обр* (*X* , *Nil*).

Робота функції: *Обернути* (*A B C D*) = *Обр* ((*A B C D*), *Nil*) = *Обр* ((*B C D*), (*A*)) = *Обр* ((C *D*), (*B A*)) = *Обр* ((*D*), (*C B A*)) = *Обр* (*Nil* , (*D C B A*)) = (*D C B A*).

Приклад 5.

Предикат *елемент* (*s* , *X*) = *T* , якщо *s* - елемент *X*, і *F*, якщо *s* - не є елементом *X (перевіряє входження елемента – атома s в список (слово) X.*

елемент (*s* , *X*): = *if Null (X) then F else*

if s = Car (X) then T else *елемент* (s, *Cdr* (*X*)).

Тут використовується вбудований умовний оператор; це – низхідна рекурсія. Спробуємо реалізувати функцію через висхідну рекурсію:

ЕЛЕМЕНТ (*s, X, y*): = *if Null (X) then y else*

if s = Car (X) then ЕЛЕМЕНТ (s, Nil, T)

else *ЕЛЕМЕНТ* (s, *Cdr* (*X*), y);

Елем (*s , X): = ЕЛЕМЕНТ (s , X, F).*

Виходить штучно і надумано. Якщо для прикладу 4 (обернути) висхідна рекурсія єдина можливість реалізувати її, то в даному випадку, низхідна рекурсія - оптимальний варіант.

Як працює функція (низхідна) перевірки входження літери (елемента) в слово (список) – слово моделюється списком:

елемент (м, елемент) = елемент (м, лемент) = елемент (м, емент) = =елемент (м, мент) = T;

елемент (ф, елемент) = елемент (ф, лемент) = елемент (ф, емент) = = елемент (ф, мент) = елемент (м, ент) = елемент (ф, нт) = елемент (ф, т)= = елемент (ф, Nil) = F.

Приклад 6.

Набір (*X*) – функція, що видаляє зі списку елементи, що повторюються. Тобто перетворює набір символів у множину.

Скористаємося функцією, отриманой в попередньому прикладі - визначення входження елемента в список. Напишемо низхідну рекурсію:

Набір (X): = if X = Nil (те ж саме, що Null (X)) then Nil else

if елемент (Car (X), Cdr (X)) then Набір (Cdr (X))

else Cons (Car (X), Набір (Cdr (X)).

Набір(addabsrst) = Набір(ddabsrst) = Набір (dabsrst) = Cons (d, набір (absrst))=
= Cons (d, Cons (a, набір (bsrst))) = Cons (d, Cons (a, Cons (b, Набір (srst)))) =
=Cons (d, Cons (a, Cons (b, Набір (rst)))) =
= Cons (d, Cons (a, Cons (b, Cons (r, Набір (st))))) =
= Cons (d, Cons (a, Cons (b, Cons (r, Cons (s, Набір (t)))))) =
= Cons (d, Cons (a, Cons (b, Cons (r, Cons (s, Cons (t, Набір (Nil))))))) =
= Cons (d, Cons (a, Cons (b, Cons (r, Cons (s, Cons (t, Nil)))))) =
= Cons (d, Cons (a, Cons (b, Cons (r, Cons (s, (t)))))) =
= Cons (d, Cons (a, Cons (b, Cons (r, (st))))) =
= Cons (d, Cons (a, Cons (b, (r s t)))) = Cons (d, Cons (a, (brst))) =
= Cons (d, (abrst)) = (dabrst).

Висхідна рекурсія:

Наб (X, У): = if X = Nil then У else if елемент (Car (X), Cdr (X)) then
Наб (Cdr (X), У) else Наб (Cdr (X), Cons (Car (X), У)).

НАБІР (X): = Наб (X, Nil).

Приклад 7.

Визначимо функцію вставки елемента *s* на *n*-ю позицію в список *X* (низхідна):

Вставка (s, *n*, *X*): = *if* [*Null (X) or n* = 1][15] *then cons* (s, *X*)
else cons (car (X), Вставка (s, n-1, cdr (X))).

Вважаємо, що всі перевірки на помилки вхідних даних зроблені і критичних ситуацій у нас не виникне.

Вставка (и, 3, пшк) = cons (п, Вставка (и, 2, шк)) = cons (п, cons (ш, Вставка (и, 1, к))) = = cons (п, cons (ш , cons (и, (к)))) = cons (п, cons (ш, (ик))) = cons (п, (шик)) = (пшик).

7.3.3. Визначення функцій вищих порядків та композицій

Розглянемо функцію:

Збільшити(*X*): = *if Null (X) then Nil else Cons*(*Car*(*X*) +1, *Збільшити*(*Cdr*(*X*)))

Функція додає 1 до кожного елементу числового списку.

Частка(X, *a*): =*if Null (X) then Nil else Cons*(*div*(*Car*(X), *a*), *Частка* (*Cdr*(X))).

Ця функція ділить на *a* кожен елемент списку.

Як бачимо, обидві функції абсолютно однотипні. Вони виконують відображення числового списку на деяку нову множину. Тому замість всіх таких функцій можна зробити одну:

---

[15] – вставка в кінець слова; перевірка на *n* = 1 забезпечує також вставку в самий початок слова; в умовних операторах допустимо використання логічних предикатів **or, and** і т.п..

Відображення(X, *f*):=*if Null*(*X*) *then Nil else Cons*(*f*(*Car*(*X*), *Відображення*(*Cdr*(*X*), *f*)).

Це та ж функція, задана не конкретно, а ім'ям *f*. Таким чином, можна, використовуючи аплікацію, визначити:

збільш (z): = z +1;

Збільшити (*X*): = *Відображенн* (*X* , *збільш*).

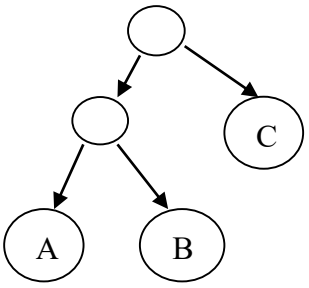
Можна зробити те ж саме, використовуючи λ-операцію:

Збільшити (*X*): = *Відображення* (*X* , λ*y*. *y* +1).

Таким чином, функція *Відображення*(*X, f*) - це функція вищого порядку. Спектр її застосування широкий і дозволяє не робити багато різних функцій.

Приклад 8. **Рекурсія по дереву**

Використовується для обходу дерева, тобто списку, елементи якого самі можуть бути списками.



Друк дерева: *Prtree* (*X*) = *If X=Nil then Stop Else if Atom* (*X*) *then print* (*X*)
Else Cons (*Prtree* (*Car* (*X*)), *Prtree* (*Cdr* (*X*))).

Підрахунок всіх термінальних вузлів:
Ntip(*X*):= *If X = Nil then* 0 *Else if Atom* (*X*) *then* 1
Else Ntip (*Car* (*X*)) + *Ntip* (*Cdr* (*X*)).

Наприклад:

Prtree ((*A B*) *C*) = *Cons* (*Prtree* (*A B*), *Prtree* (*C*)) = *Cons* (*Cons* (*Prtree* (*A*), *Prtree* (*B*)), *Prtree* (*C*)) = (*A B C*).

Ntip ((*A B*) *C*) = *Ntip* ((*A B*)) + *Ntip* ((*C)*) = *Ntip* ((*A)*) + *Ntip* ((*B*)) + *1*= = *1 + 1 + 1 = 3*

Приклад 9.

Перевірити, чи є слово паліндромом (слово змодельоване списком).

Palindrom (X): = PL (X, Обернути (X));

PL (X, Y): = if X = Nil then T else
if not (Car (X) = Car (Y)) then F else PL (Cdr (X), Cdr (Y)).

Очевидно, що функція не оптимальна, тому що обробляє ціле слово і виходить, що порівнює двічі початок-кінець і кінець-початок. Але програма вийшла дуже проста, немає сенсу її ускладнювати обчисленням довжини слова, поділом його навпіл і т.п ..

Приклад 10.

З двійковими числами можна працювати, моделюючі їх списками. Наприклад розглянемо отримання додаткового коду двійкового числа. Перший алгоритм – *інверсія зліва направо і додати 1, за розряд не виходимо*. Другий алгоритм – *розглядаємо слово з кінця, нулі залишаємо до першої 1, яку теж залишаємо; потім рухаємося справа наліво і інвертуємо цифри.* У цьому прикладі основним прийомом є композиція різного роду функцій, що робить програму дуже зручною.

Алгоритм 1:

ДодатковийКод(X): = Обернути(Додати1(Обернути (Інвертувати (X)));

Працюємо з «оберниними» списками через те, що CONS додає голову до хвоста, і ніяк інакше.

Інвертувати (X): = if X = Nil then Nil else
Cons (inv (Car (X)), Інвертувати (Cdr (X)));

Додати1(X): = if X = Nil then Nil else if Car (X) = 0 then Cons (1, Cdr (X))
else Cons (0, Додати1 (Cdr (X)).

inv (y): = if y = 0 then 1 else 0;

Алгоритм 2:

ДодатКод (X): = Обернути (Огляд (Обернути (X)));

Огляд (X): = if X = Nil then Nil else {якщо всі 0} *else if Car (X) = 0 then*
Cons (0, Огляд (Cdr (X)) else Cons (1, Інвертувати (Cdr (X))).

Дано двійкове число 0010. В додатковому коді до нього буде 1110. *Перевіряємо 0010 + 1110 = (1) 0000.*

ДодатковийКод (0010) =
=Обернути (Додати1(Обернути (Інвертувати (0010))))=
= Обернути (Додати1(Обернути (1101))) =
=Обернути (Додати1 (1011)) =
= Обернути (Cons (0, Додати1(011)) = Обернути (Cons (0, Cons (1, (11)))) =
= Обернути ((0111) = 1110.

ДодатКод (0010) =
=Обернути (Огляд (Обернути (0010))) = Обернути (Огляд (0100)) =
= Обернути (Cons (0, Огляд (100))) =
=Обернути (Cons (0, Cons (1, Інвертувати (00)))) =
=Обернути (Cons (0, Cons (1, (11)))) = Обернути (0111) = 1110.

Питання до розділу 7

1. Лямбда-числення. Функціональне програмування.
2. Визначення функцій за допомогою λ-абстрактора. Операція аплікації.
3. Польський запис арифметичного виразу: прямий і зворотний. Обхід дерева виразу.
4. Особливості функціонального програмування. Перелічити, навести приклади.
5. Висхідна рекурсія. Низхідна рекурсія. Навести приклади.
6. Функції обробки списків, атом, список.

8. НЕКЛАСИЧНІ ЛОГІКИ

8.1. Багатозначні логіки

Двозначна логіка передбачає істинність і хибність висловлювань. У багатозначних логіках число значень істинності аргументів та функцій може бути навіть у загальному випадку нескінченним. Багатозначна логіка – тип формальної логіки, характерний наявністю більш ніж двох можливих істиннісних значень (істинності та хибності). Першу систему багатозначної логіки запропонував польський математик Ян Лукасевич[16] 1920 році. В даний час існує дуже багато систем багатозначної логіки, які у свою чергу можуть бути згруповані за класами. Найважливішими з таких класів є часткові логіки та нечіткі логіки.

Розглянемо кілька прикладів багатозначних логік.

8.1.1. Тризначна система Лукасевича

Тризначна логіка була історично першою багатозначною логікою і є найпростішим розширенням двозначної логіки. Перелік істиннісних значень тризначної логіки, крім «істинно» і «хибно», включає також третє значення, яке, як правило, трактується як «невизначено», «невідомо» або «помилково». В останньому випадку логіку зазвичай називають частковою.

У тризначній логіці природно не виконується закон виключення третього. Разом з тим, важливою властивістю тризначних логік, що відображає їхню адекватність, є те, що всі вони є розширеннями класичної двозначної логіки. Тобто, в припущенні, що символи, що інтерпретуються, не приймають третього істиннісного значення, семантика формул у тризначній логіці така ж, як і в двозначній.

Як третє логічне значення висловлювання Лукасевичем було введено значення, що виражається словами «ймовірно», «нейтрально». Про кожне висловлювання у системі Лукасевича можна сказати: воно або істинно (1), або хибно (0), або нейтрально (S). Це стало можливим завдяки тому, що Лукасевич

---

[16] Ян Лукасевіч (польськ. Jan Łukasiewicz; 21 грудня 1878, Львів — 13 листопада 1956, Дублін) — польський логік, член Польської Академії Наук (1937), один із головних представників львівсько-варшавської школи.

З 1945 - професор Королівської ірландської академії у Дубліні. Працював у галузі логічних проблем індукції та причинності та логічних підстав теорії ймовірностей . Побудував першу систему багатозначної логіки, і з її допомогою — систему модальної логіки. Розробив оригінальну мову для формалізації логічних виразів (т. зв. польський запис, що послужив основою для більш відомого зворотного польського запису). За філософськими поглядами – позитивіст (напрямок у методології науки, що оголошує єдиним джерелом істинного, дійсного знання емпіричні дослідження і заперечує пізнавальну цінність філософського дослідження).

одним із перших висунув тезу про можливість побудови логічних обчислень, в яких не діє принцип несуперечності. На підставі тризначної логіки Лукасевич побудував систему модальної логіки, в якій поряд із дослідженням логічних операцій над асерторичними висловлюваннями (ствердженнями та запереченнями) досліджуються так звані модальні висловлювання (сильні та слабкі твердження та заперечення).

Основними функціями їм були взяті заперечення та імплікація, а похідними – кон'юнкція та диз'юнкція. Тавтологія у логіці Лукасевича набуває значення «1», тобто це – виділене істинне значення. Заперечення та імплікація визначаються таблицями та рівностями, наведеними нижче:

x	***заперечення***
1	0
S	S
0	1

імплікація

x\y	**1**	**S**	**0**
1	1	S	0
S	1	1	S
0	1	1	1

Через те, що в загальному випадку в багатозначних логіках число значень істинності аргументів та функцій може бути нескінченним, то прийняті позначення та обчислення через формули:

[Nx] = 1-[x] - заперечення

[Cxy]=1, якщо [x]≤[y] - імплікація
[Cxy]=1-[x]+[y], якщо [x]>[y]
Тобто,
[Cxy] = min (1,1-[x] + [y])

Визначення похідних функцій:

[Kxy] = min ([x], [y]) - кон'юнкція
[Axy]=max([x],[y]) - диз'юнкція

Вочевидь, що у цих визначеннях не є законами логіки (тавтологіями) закони двозначної логіки: виключення третього, протиріччя та його заперечення. Тому система Лукасевича не є запереченням двозначної логіки. У його логіці правило зняття подвійного заперечення, правила де Моргана та правило контрапозиції є тавтологіями. А правило приведення до абсурду – ні.

У цій логіці не є тавтологіями і ряд формул, що виражають правильні дедуктивні умовиводи традиційної логіки, формалізовані засобами логіки алгебри (наприклад, закон силогізму).

Очевидно, що всі тавтології логіки Лукасевіча є тавтологіями двозначної логіки. Проте, через те, що Лукасевич має ще одне значення істинності – S, зворотне твердження невірно.

У 1954 Лукасевич розробив чотиризначну систему логіки, а потім - нескінченнозначні (n-значні) логічні системи, в яких множина істиннісних значень лічильно-нескінченно або має потужність континууму. У якості істиннісних значень виступають раціональні числа з відрізка (0, 1). Моделями нескінченних логік Лукасевича є ним же розроблені алгебри.

8.1.2. Логіка Гейтинга

Із закону виключення третього у двозначній логіці виводяться:

1. $\neg\neg x \to x$

2. $x \to \neg\neg x$

Гейтінг[17] створив тризначну пропозиційну логіку, ґрунтуючись на твердженні, що істинним є лише $x \to \neg\neg x$. Імплікація та заперечення відрізняються від визначень цих операцій у Лукасевича.

x	***заперечення***
1	0
S	0
0	1

імплікація

x\y	**1**	**S**	**0**
1	1	S	0
S	1	1	0
0	1	1	1

Через те, що в загальному випадку в багатозначних логіках число значень істинності аргументів та функцій може бути нескінченним, то прийняті позначення та обчислення через формули:

[Cxy]=1, якщо [x]≤[y]

[Cxy]= [y], якщо [x]>[y]

Тавтологія набуває значення 1, кон'юнкція-диз'юнкція обчислюється як мінімум і максимум. У логіці Гейтинга закони несуперечності, контрапозиції, де Моргана та виключення четвертого є тавтологіями. Але закон заперечення третього та його заперечення – не є тавтологіями.[ii]

[17] **Аренд Гейтінг** (1898 – 1980) – голландський математик і логік, один з найвизначніших представників інтуїціонізму після Брауера.

8.1.3. Тризначна система Бочвара

Побудована Бочваром[18] 1938 р. тризначна логіка B_3 була застосована ним для аналізу парадоксів Б.Рассела та Г.Вейля. Третім істиннісним значенням (відмінним від істини і хибно) B_3 є «безглуздя». Аналіз логічних і семантичних парадоксів у сенсі Бочвара полягає у доказі безглуздості парадоксальних висловлювань.

Важливою методологічною ідеєю, запропонованою Бочваром у зв'язку з аналізом парадоксів, була ідея розрізнення внутрішніх та зовнішніх логічних зв'язок та побудова двох рівнів логічної мови – внутрішньої мови, в якій виражаються деякі факти, але немає засобів їх доказу, та зовнішньої мови, в якій доводяться затвердження про факти, подані формулами внутрішньої мови. Відповідно до цієї ідеї парадоксальна формула належить внутрішній мові, а твердження про її безглуздість – зовнішній.

Бочвар опублікував цикл робіт з теорії логічних парадоксів. Засобами розширеного числення предикатів K_0 він охарактеризував обмеження, які слід накласти на аксіоми згортання[19], що є «джерелом» парадоксів.

Бочвар досліджував також логічні парадокси, засновані на множині визначень предикатів, кожне з яких несуперечливо окремо. Він також ввів у розгляд оператори логічної апроксимації вивчення структур аксіом згортання (як «джерел» парадоксів).

Ідеї, розвинені Бочваром, активно застосовувалися у роботах з проблем філософської логіки, що розглядають природу парадоксів, у дослідженнях з логічного аналізу природної мови, а також теорії автоматизованих правдоподібних міркувань, заснованих на багатозначних логіках.

Створюючи свою систему Бочвар не тільки розділив висловлювання на такі, які мають сенс («істина» чи «хибно») і безглузді, а й виділив зовнішні та внутрішні форми (функції).

Внутрішніми функціями називаються класичні змістовні функції змінних висловлювань, а зовнішніми – некласичні. Позначивши «істину» R або 1, «хибно» - F або 3, «безглуздість» - S або 2, автор запровадив заперечення внутрішнє - ~a , зовнішнє заперечення - ¬ a, ã - внутрішнє заперечення зовнішнього твердження, ≡ - зовнішня рівнозначність, ↔ - зовнішня рівносильність. У логіці Бочвара закони тотожності, заперечення двозначної логіки є тавтологіями.

---

[18] **БОЧВАР** Дмитро Анатолійович (1903 – 1990) – логік, квантовий хімік, творець (поряд з Е.Постом та Я.Лукасевичем) нового напряму досліджень у логіці – багатозначних логік ; закінчив Московське Вище Технічне училище (1924), стажувався (з хімії) у Німеччині, де слухав лекції Д.Гільберта, які пробудили в ньому інтерес до логіки.

[19] Аксіоми вибору фактично - принцип згортання, полягає в тому, що для будь-якої властивості P вважається існуючим безліч, що складається з тих і лише тих об'єктів, які мають властивість P.

Заперечення закону тотожності якраз і дозволило вирішити парадокс Рассела про множину усіх формальних множин і довести існування такого предмета, як множина всіх нормальних множин. Це означає, що багато всіх нормальних множин не можна розглядати як фіксований об'єкт, що не змінюється в часі.

У логіці Бочвара формули, наведені нижче, є протиріччями:

$a \wedge \neg a$

$a \leftrightarrow \tilde{a}$

$a \equiv \neg a$

8.1.4. К-значна логіка Поста

1921 року Еміль Пост[20] захистив докторську дисертацію в галузі математики в Колумбійському університеті. У дисертації він виклав метод оцінки пропозиційних формул за допомогою таблиць істинності. У ній вперше отримано низку фундаментальних результатів у металогіці для класичної логіки висловлювань: несуперечність, дедуктивна повнота, розв'язність, функціональна повнота. У цій роботі вперше побудовано багатозначну логіку з більш ніж 3 істинними значеннями та з довільною кількістю виділених значень. Тут встановлено, що множина замкнутих класів у класичної логіці є зліченою.

Логіка Э.Л.Поста є узагальненням окремого випадку двозначної логіки, коли к=2. За Постом значення істинності приймають значення 1,2, ..., к. У цих термінах формула є тавтологія, коли приймає таке значення і, що $1 \le i \le S$, де $1 \le S \le n-1$. Значення 1, ..., S називаються виділеними. При цьому S може бути і більше 2. Пост ввів N^1_x – циклічне заперечення, N^2_x – симетричне заперечення. Вони визначаються такими таблицями та рівностями:

Циклічне заперечення визначається рівностями:

$[N^1_x]=[x]+1$ при $[x] \le k-1$

$[N^1_x]=1$

Симетричне заперечення за Постом визначається:

$[N^2_x] = k-[x] +1.$

---

[20] **Еміль Леон Пост** (1897, Августов, Царство Польське, Російська імперія) - 1954, Нью-Йорк, США) - американський математик і логік; один із засновників багатозначної логіки (1921); запропонував абстрактну обчислювальну машину - машину Поста. Еміль Пост входить до четвірки великих вчених, які практично одночасно усвідомили можливість уточнення загального уявлення про алгоритм.

У дитячому віці Еміль захоплювався астрономією, проте нещасний випадок перекреслив плани хлопця – у 12-річному віці він втратив ліву руку. Перед закінченням школи Еміль подав запит до кількох обсерваторій — чи не завадить його нестачі професії астронома. Отримані відповіді втримали його від дитячих амбіцій, і Еміль зайнявся математикою.

x	N^1_x	N^2_x
1	2	k
2	3	k-1
3	4	k-2
.	.	.
.	.	.
k-1	k	2
k	1	1

Очевидно, що при k=2 циклічне та симетричне заперечення збігаються з запереченням двозначної логіки та між собою. Операції кон'юнкції та диз'юнкції визначаються як мінімум і максимум значення аргументів.

Якщо значення істинності є 1, 2, 3, то з n-значної системи Посту вичленовується тризначна логіка P_3. Аналогічно для логік вищих порядків.

Як приклад розглянемо тризначну систему Посту, засновану на циклічному запереченні.

P	**1**	**2**	**3**
Циклічне заперечення	2	3	1

P\H	1 2 3	1 2 3	1 2 3	1 2 3
1	1 1 1	1 2 3	1 2 3	1 2 3
2	1 2 2	2 2 3	1 2 2	2 2 2
3	1 2 3	3 3 3	1 1 1	3 2 1
	кон'юнкція (мінімум)	*диз'юнкція (максимум)*	*імплікація* $(\neg P \vee H)$	*еквівалентність* $(P \rightarrow H)\&(H \rightarrow P)$

Вочевидь, якщо у ролі значень істинності взяти *1 – «істина» і 3 – «хибно»*, тоді таблиць Посту для тризначної логіки легко визначаються операції заперечення, кон'юнкції, диз'юнкції, імплікації, *справедливі для двозначної класичної математичної логіки.*

У системі P_3, як і в попередніх випадках, тавтологія набуває значення 1, і закон виключення третього не є нею ні для першого, ні для другого заперечення Посту. Однак закон виключення четвертого для першого (циклічного) заперечення є тавтологія.

8.1.5. Тризначна система Рейхенбаха

Апарат багатозначних логік знаходить дедалі ширше застосування різних науках. Більшість операцій цієї системи було введено вже Постом, але з метою застосування своєї системи до квантової механіки Рейхенбах[21] запроваджує нові. У Посту було запроваджено два види заперечення – циклічне та симетрічне. В системі Рейхенбаха вони називаються циклічним і діаметральним запереченнями, крім них Рейхенбах ввів повне заперечення. В системі Рейхенбаха є стандартна імплікація ⊃ і стандартна еквівалентність ≡. Вводяться й інші операції: альтернативна імплікація →, квазіімплікація э і альтернативна еквівалентність ≡. Знаком • позначена кон'юнкція, ∨ - диз'юнкція.

Таблиця для трьох видів заперечень Рейхенбаху. Позначення:

$\sim A$ – циклічне заперечення; $-A$ – діаметральне заперечення; $\bar{A}$ - повне заперечення.

Рейхенбах позначив "істину" як 1, "невизначеність" - 2, "хибність" - 3. Тавтологія набуває значення 1.

A	$\sim A$	$-A$	$\bar{A}$
1	2	3	2
2	3	2	1
3	1	1	1

Інші функції Рейхенбаху також визначаються матрицями.

Відзначимо ряд властивостей, притаманних запереченням в системі Рейхенбаха.

Для циклічного заперечення вірний закон зняття потрійного заперечення: $\sim\sim\sim A \equiv A$, тобто внаслідок потрійного заперечення A повертаємося до початкового значення A. Для циклічного заперечення закони несуперечливості та виключення третього, правила де Моргана двозначної логіки не є тавтологіями, але тавтологією є закон виключення четвертого: $A \vee \sim A \vee \sim\sim A$.

Для діаметральної заперечення зберігається правило зняття подвійного заперечення: $- - A \equiv A$. Ні самі закони несуперечливості та виключення третього, ні їх заперечення при діаметральном запереченні не є тавтологіями.

Для повного заперечення виявилися тавтологіями закон непротиріччя, закон виключеного четвертого, правила де Моргана.

---

[21] **Ханс Райхенбах** (1891, Гамбург - 1953, Лос-Анджелес) - німецько-американський філософ, представник логічного позитивізму. Приймаючи принцип верифікації як методологічний інструмент, Рейхенбах розширив його початкове трактування, замінивши поняття «істинності» на поняття «імовірності», що дозволяло розв’язати методологічну скруту, пов'язану із «законами природи»: закони природи є припущеннями, які треба оцінити з погляду граничних ймовірностей - "істина" (1) і "хибно" (0)

Рейхенбах збудував свою тризначну систему для опису явищ квантової механіки. На його думку, говорити про істинності чи хибності висловлень правомірно лише тоді, коли можливо здійснити їх перевірку. Якщо не можна ні підтвердити істинність висловлювання (верифікувати його), ні спростувати його за допомогою перевірки (фальсифікувати), то такий вислів має оцінюватися третім значенням - невизначено. До таких висловлювань відносяться висловлювання про неспостережені об'єкти у мікросвіті.

8.1.6. Нескінченна логіка

Нескінченну логіку можна ввести таким чином:

- істиннісне значення знаходиться у відрізку дійсних чисел від 0 до 1;
- заперечення визначається як: $\neg A = 1 - A$;
- кон'юнкція визначається як: $A \wedge B = \min(A, B)$;
- диз'юнкція визначається як: $A \vee B = \max(A, B)$.

До формальних систем нескінченної логіки можуть бути віднесені системи R-функцій В. Л. Рвачова (1926-2005, ректор Харківського інституту радіоелектроніки).

Може здатися, що теорія ймовірностей дуже схожа на нескінченнозначну логіку: ймовірність відповідає істинному значенню (1 = істина, 0 = хибно), ймовірність ненастання будь-якої події відповідає запереченню, ймовірність одночасного настання двох подій відповідає кон'юнкції, а ймовірність настання хоча б однієї з двох подій відповідає диз'юнкції.

Однак між багатозначними логіками і теорією ймовірностей є принципова відмінність: у логіках істиннісне значення будь-якої функції цілком визначається істиннісним значенням її аргументів, тоді як у теорії ймовірностей ймовірність складової події залежить не тільки від ймовірностей подій-компонентів, що входять до нього, але і від їх залежності один від одного (що виражається через їх умовні ймовірності).

Це виявляється, зокрема, у тому, що в теорії ймовірностей виконується еквівалент «закону виключеного третього»: ймовірність того, що деяка подія настане чи не настане, завжди дорівнює одиниці, тоді як в багатозначних логіках закон виключеного третього не виконується.

В теорії ймовірностей виконується також еквівалент «закону протиріччя»: ймовірність того, що деяка подія одночасно настане і не настане, завжди дорівнює 0, тоді як в багатозначних логіках закон протиріччя не виконується.

У той же час існує певний зв'язок між істинними значеннями вищеописаної нескінченнозначної логіки та ймовірностями теорії ймовірностей, а саме:

- якщо a - ймовірність деякої події, то ймовірність ненастання цієї події становить $1 - a$;

- якщо a і b - ймовірності деяких двох подій, то ймовірність спільного наступу цих двох подій не перевищує min(a , b);
- якщо a і b — ймовірності деяких двох подій, то ймовірність настання хоча б однієї з цих подій більша або дорівнює max(a , b).

8.2. Модальні логіки

У природній мові висловлювання можуть набувати різних значень істинності залежно від контексту. Часто ми не можемо сказати з упевненістю, що висловлювання істинне чи хибне. Ми вживаємо тоді такі слова, як «можливо», «іноді», «мабуть», «потрібно», і т.п. Такі слова називаються *модальностями.* Виникають модальності у тих галузях мислення, у яких допускаються різні аспекти в питаннях істинності. Неможливо, щоб 2+2=5, але можливо, що існує снігова людина. У класичній логіці спосіб міркування вважається правильним, якщо істинність посилок гарантує істинність висновку. Істинність речень забезпечується її відповідністю до дійсності. Речення вважається істинним, якщо воно відповідає дійсності. Однак такі важливі аспекти, як відносність знання, зміна та зростання знання, конкретність істини та низка інших ігноруються при побудові семантики класичних логік. Ці аспекти враховуються у некласичних логіках, які називаються інтенсіональними: модальними, часовими, деонтичними, епістемічними, релевантними логіках.

8.2.1. Логіка можливого

Перші дослідження в галузі модальної логіки належать Арістотелю, який поряд з асерторичними силогізмами ввів в обіг модальні силогізми, в яких хоча б одна з посилок є висловлюванням типу "А необхідно належить", "А можливо належить". При цьому необхідне Аристотель не вважав за можливе. Наступний крок у розвитку модальної логіки зробив учень Аристотеля Теофраст, який став відносити модальність до висловлювань загалом, а не до окремих понять. Крім того, він прийняв тезу: все необхідне можливо, що відкрило дорогу до визначення можливості через необхідність: "можливо А" еквівалентно "не необхідно не-А". У середні віки стався поділ модальностей на модальності de dicto (про мову), які стосуються висловлювання загалом, і модальності de re (про речі), які стосуються властивостей. Сучасні дослідження модальної логіки пов'язані багато в чому з ім'ям К.Льюїса, який побудував обчислення S1 - S6. Характерним прикладом можуть бути його обчислення S4 і S5 (в трактуванні *К.Геделя*) . Ці обчислення будуються як розширення класичної логіки висловлювань та класичної логіки предикатів. Мова логіки поповнюється модальним оператором (необхідно), що діє на речення мови. Оператор можливості ◊ вводиться як

скорочення ¬□¬. Визначення формули поповнюється пунктом: якщо А – формула, то □А – теж формула.

Можливість та *необхідність* називаються *алетичними* модальностями або модальностями *можливого*. Формальна мова, що використовує поняття "можливо" і "необхідно" як квантори, називається *логікою можливого,* або *алетичною логікою*.

У модальній логіці, тобто в області логіки, в якій вивчаються логічні оператори, які називають модальностями, лише на рівні обчислення висловлювань, де відомо лише те, що вони істинні чи хибні, застосовується відома нам система пропозиційних зв'язок. Як вище було сказано, що у *алетичної* логіці до цих зв'язків додаються модальні оператори □ – *необхідно* і ◊ – *можливо* .

Наприклад, □*F* читається: “необхідно, щоб *F*” або “*F* необхідно”. Це означає, що *F* необхідно істинно, чи істинно у всіх можливих світах.

◊*F* читається: “можливо, що *F*” або “*F* можливо”. Це означає, що *F* можливо істинно, або істинно у деякому можливому світі.

Як і в будь-якій логічній дисципліні, в модальній логіці зв'язок між символами речення (поняттями, об'єктами) виражається у вигляді формул. Окрім формули □*P* в *алетичній логіці* додаються такі істинні формули з модальними операторами:

$$\Box P = \neg\Diamond\neg P \quad \textit{(закон двоїстості)}$$

$$\Diamond P = \neg\Box\neg P$$

$$P \rightarrow \Diamond P$$

$$\Box P \rightarrow P$$

Не виводяться (тобто не є істинними, але не обов'язково – хибними):

$$P \rightarrow \Box P$$

$$\Diamond P \rightarrow P$$

$$\Diamond P$$

$$\neg\Box P$$

Якщо у формулу замість пропозиційних символів підставити якесь судження, вона стає судженням типу: «необхідно, що матерія вічна», «можливо, що на Марсі є життя».

Аксіоматику пропозиційного модального обчислення отримуємо, додаючи до аксіомних схем і правил виведення класичної логіки висловлювань модальну схему аксіом □(А→В) → (□А→□В) і правило виведення: «якщо доводиться А→В, то доводиться □А→□ В» (правило С). Це пропозиціональне модальне обчислення С2. Замінивши правило С більш сильним правилом висновку: «доказуємо А→доказуємо □А» (правило Геделя) і додавши до С2 одну з аксіомних схем □А→А, □А→□□А, □А→□ □А, отримуємо модальні пропозиції обчислення Т, S4 та S5 відповідно. З цими обчисленнями немає ніяких принципових проблем.

Цілком інша ситуація виникає, якщо ці «модальні приставки» додавати до класичної логіки предикатів, оскільки в предикатних модальних контекстах може порушуватися закон підстановки тотожних $\forall x \forall y(x = y \rightarrow (F(x) \rightarrow F(y))$. Наприклад (приклад Куайна), твердження «необхідне, що 9 більше 7» та його екзистенційне узагальнення «$\exists$ x такий, що необхідно, що x більше 7» вірно, якщо x є 9 і 9 є натуральне число, але невірно, якщо x є 9 та 9 є число планет.

Відповідно до Куайну, входження змінної x у відкриту формулу «необхідно, що x більше 7» референціально неясно, оскільки не можна гарантувати, що, будучи пов'язаною, змінна x іменує точно один об'єкт. Тому модальна логіка предикатів потребує деякої зміни принципів, на яких побудовано немодальну стандартну теорію квантифікації.

Зокрема, екзистенційне узагальнення в модальних контекстах має ґрунтуватися на наступному правилі: $\exists$ -квантифікація відкритого речення справедлива, якщо, і тільки якщо є замкнутий терм, підстановка якого на місце змінної квантифікації призводить до істинного речення. Відповідно підстановочність тотожного має місце, якщо і тільки якщо взаємозамінні терміни є синонімами.

Прийняття такого принципу теорії квантифікації веде до так званної підстановочної інтерпретації кванторів, на відміну стандартної чи об'єктної їх інтерпретації. У стандартній інтерпретації значеннями пов'язаних змінних є об'єкти універсуму, у підстановчій – терміни мови. Підстановча теорія нічого не говорить про існування чи неіснування об'єктів; вона досліджує лише певні відносини між твердженнями мови. Всі істинні теорій підстановочного типу є в загальному випадку лінгвістичними, і їх використання для опису конкретних ситуацій потребує додаткових припущень про характер універсуму (багато об'єктів, допустимих у цій ситуації). Ще один спосіб обґрунтування квантифікації в модальних контекстах заснований на припущенні, згідно з яким значеннями пов'язаних змінних у модальних контекстах є не об'єкти та не терміни, а сенси, тобто, певні методи розуміння об'єктів. При цьому одному й тому ж об'єкту можуть відповідати різні сенси.

Розроблено кілька аксіоматичних систем модальних логік – Геделя, Аккермана, Лукасевича, Каррі, Тарського, Льюїса тощо.

Видно зв'язок модальної логіки з багатозначною тому, що найпростіша система модальної логіки є системою тризначної логіки, в якій прийнято третє значення – "можливо". Більшість систем модальної логіки тісно стикаються з ймовірнісною логікою, тому що вони є лічильно-нескінченнозначними. Модальності корисні в описах фізичного світу, у процесі аналізу причинності.

8.2.2. Види модальностей

Деонтична логіка вводить модальності “дозволено”, “заборонено” та “обов'язково”, які використовуються як конструкції “дозволяється” та “треба, щоб”.

Епістемічна логіка, чи логіка знання, використовує модальності “знання” та “віри”. Часто використовуються "достовірно", "ймовірно", або "доведено", "спростовано", "підтверджено". Наприклад, епістемічний оператор: *вірить (а)А* – формула *вірить (а)А* істинна, якщо суб'єкт *а* вірить у *А.*

Автоепістемічні логіки формалізують інтроспективні та ідеально розумні міркування. Вони припускають існування ідеально розумного суб'єкта, який веде міркування про власні знання. Модальні логіки віри та знання підходять для формалізації такого роду міркувань.

Часова логіка вводить модальності "іноді" і "завжди" та їх заперечення "часто" і "ніколи" разом з квантифікаторами "у минулому" та "в майбутньому".

Часові оператори:

G – завжди (у майбутньому). GA є істинним, якщо *А* залишається завжди істинним.

H – завжди (у минулому). HA є істинним, якщо *А* завжди було істинним.

F – іноді (у майбутньому). FA – істинно, якщо *А* іноді буде дійсним.

P – іноді (у минулому). PA є істинним, якщо *А* іноді було істинним.

U – до тих пір, поки. U(A, B) істинно, якщо *А* істинно, починаючи з поточного моменту, доки B не стане істинним у майбутньому.

Всі ці модальності запроваджуються як оператори. Є схожість між визначеннями кожної пари модальних операторів – для кожного модального оператора існує двоїстий: *необхідно – можливо, завжди – іноді, знаю – вірю* тощо. Модальності різних видів вивчаються часто одними й тими самими формальними методами, тобто, при побудові формальних систем ми відволікаємося від їхньої конкретної інтерпретації. Тому, зважаючи на формальну подібність, вони вивчаються разом, і всі вони називаються *модальними логіками.* Логіка з кванторами *необхідно* і, *можливо,* історично з'явилася першою, вона найкраще досліджена. Цю логіку називають *логікою можливого,* проте часто саме її мають на увазі під *модальною логікою.*

Приклади використання модальних операторів для формалізації висловлювань:

◊*Виграти (Джон, лотерея, телевізор) – можливо, що Джон виграє у лотерею телевізор.*

¬(◊(H ∀*y* ∃*z* (*Джон, y, z*))) – *неможливо, щоб Джон у майбутньому завжди в усе вигравав.*

Вірить (Джон) УДАЧА → вірить (Джон) (вірить (Джон) УДАЧА) - якщо Джон вірить в удачу, то він вірить, що він вірить в удачу (аксіома позитивної інтроспекції).

¬Вірить (Джон) УДАЧА → вірить (Джон) (¬вірить (Джон) УДАЧА) – якщо Джон не вірить у удачу, то він вірить, що він не вірить у удачу (аксіома негативної інтроспекції).

8.2.3. Зміст модальних операторів

У зв'язку з «модальною приставкою» виникають деякі змістовні труднощі. Обчислення з правилом Ґеделя та аксіомною схемою □A→A називаються нормальними, тобто відповідними змістовним стандартам логічної необхідності: всяка теорема логічно необхідна (логічно істинна) і всяке необхідне істинне твердження істинно. Всі інші обчислення не вважаються нормальними, і для них окремо мають бути вказані значення, в яких вони використовують оператори необхідності та можливості. Ось деякі можливі змісти цих операторів, відмінні від зазначеного вище «алетичного» сенсу □ і ◊:

1) □ означає доказовість, а ◊ – несуперечність (інтуїціоністські модальності, що не виключають, втім, існування спеціальної логіки доказовості);

2) □ означає обов'язковість у сенсі необхідності дотримання норм, а ◊ – дозвіл або відсутність заборони (деонтичні модальності);

3) □ означає прийнятність емпіричної гіпотези, а ◊ – її невід'ємність (індуктивні модальності);

4) □ означає «скрізь» або «завжди», а ◊ – «де-не-де» або «іноді» (просторово-часові модальності);

5) □ означає «знаю, що», а ◊ – «не знаю не» (епістемічні модальності). Істотно, що цей список потенційно необмежений (тобто він обмежений лише нашою винахідливістю, а чи не істотою справи).

Синтаксичні характеристики операторів □ та ◊ у всіх цих випадках мають бути різними. Наприклад, для деонтичних модальностей не проходить аксіомна схема □A→A, оскільки норми можуть бути порушені. Замість неї повинна використовуватися аксіомная схема □ A → ¬ □ ¬ A (обов'язкова норма допустима).

Для всіх цих та багатьох інших модальних обчислень гостро постала проблема їхньої формальної інтерпретації: побудова адекватної їм формальної семантики, в якій:

1) кожна формула обчислення є або істинною, або хибною;

2) кожна доведена формула істинна (несуперечність числення);

3) кожна істинна формула доведена (повнота обчислення);

4) встановлено тісний зв'язок із змістовною семантикою.

Перший крок було зроблено Р.Карнапом . Використовуючи ідеї Лейбніца, він будує семантику на основі множини описів стану (положень справ, що характеризуються засобами мови, або «можливих світів»). Вислів «А можливо» семантично характеризується ним як «А істинно хоча б в одному описі стану (можливому світі)» та висловлювання «А необхідне» як «А істинно у всіх описах стану (можливих світах)».

Наступний крок пов'язаний з ім'ям *С.Кріпке.* Він відмовився від обов'язкового уявлення можливого світу як опису стану, що залежить від структури логічного мови. Таке уявлення зберігається лише в канонічних моделях (максимально несуперечливих множинах), тоді як у загальному випадку можливий світ – це просто елемент довільної непустої множини (можливих світів). При цьому допускається можливість існування ізольованих елементів такої множини (елементів, не пов'язаних з жодними іншими елементами множини).

8.2.4. Строга імплікація

Модальні оператори тісно пов'язані з розумінням імплікації «якщо… то…». Класичне визначення імплікації, яка є хибною лише за істинності посилки та хибності висновку, призводить до істинності таких безглуздих з погляду здорового глузду тверджень, як, наприклад: «якщо двічі дві – п'ять, то Київ – столиця України». Така імплікація одержала назву *матеріальної.*

Імплікація, яка вважається істинною тільки в тому випадку, коли неможливо і не було можливим, щоб посилка була істинною, а висновок – хибним називають *нематеріальною.* Визначення такої імплікації включає модальність та часову залежність.

Парадокси матеріальної імплікації полягають у невідповідності сенсу операції $A \rightarrow B$, вираженої формально, і звичного, який у життєвому побуті вкладається в операцію логічного висновку однієї думки з іншої, пов'язаних змістовно спілками «якщо…, то…»:

1. з брехні може випливати будь-що;
2. істина може випливати з чого завгодно.

Ці парадокси практично не ведуть до негативних наслідків пізнання. Йдеться не про парадокси логіки, а про невідповідність їхньої інтерпретації.

Відома низка спроб усунення цієї проблеми.

К.І.Льюїс (1912 р.) побудував низку логічних систем для уточнення поняття нематеріальної імплікації. Вони містили модальні оператори. Нематеріальну імплікацію він назвав *строгою імплікацією.* Остаточні результати його роботи були пізніше сформульовані та узагальнені як 5 канонічних систем S1 – S5.

Льюіс (Lewis) Кларенс Ірвінг (1883-1964) - американський філософ, викладав у Гарварді, прихильник "концептуального прагматизму". Логічні дослідження Льюїса присвячені розробці різних систем модальної логіки, до яких він прийшов, відкидаючи класичну імплікацію. У статті "Імплікація та алгебра логіки" (Implication and algebra of logic. - "Mind", 1912, v. 21). Льюїс піддав критиці парадоксальну, на його думку, відсутність смислового зв'язку між A і B у істинній матеріальній імплікації $A \to B$. Найбільш повно логічні системи Льюїса розвинені в написаній спільно з Г. Лангфордом монографії «Символічна логіка» (Symbolic logic), де викладено систему, в якій задається оператор можливості ◊. $\Diamond p$ означає: "p можливо", або "можливо, що p істинно". Показується, що $\Diamond p$ еквівалентно твердженню «неправильно, що p тягне власне заперечення». Через заперечення, кон'юнкцію та можливість визначається строга імплікація $p \to q = \neg \Diamond(p \& \neg g)$: з p строго слідує q, що означає «неправильно, що можливо, щоб p було істинно, а q хибно». Якщо q виводиться з p, то твердження "p істинно, а q хибно" означає приховане протиріччя.

Записується строга імплікація у вигляді наступної формули:

$A \prec B = \neg \Diamond (A \& \neg B) = \neg \Diamond \neg (A \to B) = \Box (A \to B)$.

Схема аксіом, запропонована Льюїсом, складається з 9 аксіом. У цій системі не виведені закони: $A \prec (B \prec A)$ закон монотонності, $\neg A \prec (A \prec B)$ закон протиріччя.

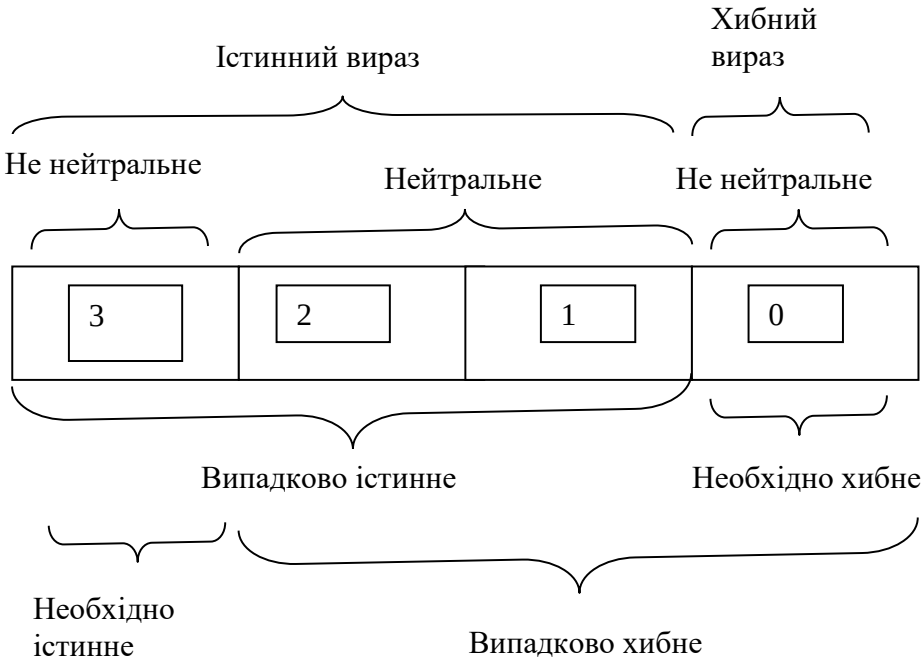
Зв'язка $\prec$ відображає змістовний зв'язок антецедента (А) і консеквента (В), але строга імплікація породжує свої парадокси, аналогічно традиційним, лише пов'язані з поняттями необхідності та можливості. При інтерпретації строгої імплікації як логічного слідування виходить, що з неможливого висловлювання випливає будь-яке, і необхідне висловлювання випливає з будь-якого. Таким чином, строга імплікація стає марною, оскільки, якщо А неможливо, то не можна використовувати $A \prec B$ як підставу для доказу В. Якщо В необхідно, то не потрібно ніяких посилок для його застосування і $A \prec B$ стає зайвим. Крім того, модальні поняття «необхідність» і «можливість» прийняті як первинно ясні, хоча такими не є.

Відома система обчислення висловлювань Аккермана для строгої імплікації складається із 15 схем аксіом. Строгу імплікацію Аккерман визначає, як «В є частиною змісту А». Йому вдалося позбутися парадоксів системи Льюїса. Хоча також не виводяться закони монотонності та протиріччя, наприклад. Але й ця система не позбавлена недоліків: багато корисних формул і законів теж не виводяться.

Введення поняття строгої імплікації має основну мету: так чи інакше наблизитися до того відношення видимості, яким люди інтуїтивно керуються у змістовних (неформальних) міркуваннях.

8.2.5. Чотиризначна семантика модальної логіки

Множину істинних подій можна поділити на *випадково істинні* і *необхідно істинні.* Аналогічно хибні події можна розділити на *випадково хибні* і *необхідно хибні.* Необхідні істинні події істинні у всіх можливих світах, а випадково істинні пов'язані з подіями, що підтвердилися у існуючому світі, але не у всіх можливих світах. Аналогічно для хибних.



Можна порівняти:

Необхідно істинне – 3,
Випадково істинне – 2,
Випадково хибне – 1,
Необхідно хибне – 0.

Тоді модальна логіка буде занурена у чотиризначну логіку.

Припустимо, що семантика можливих світів зводиться до семантики з двома можливостями: існуючий світ А і можливий В. Тоді вважатимемо, що подія

- необхідно хибна, якщо вона хибна в А і В, - (0).
- випадково хибна, якщо вона хибне в А і істинно у В, – (1).
- випадково істинна, якщо вона істинна в А і хибно у В - (2).
- необхідно істинна, якщо вона істинна і А, і В, – (3).

Така інтерпретація призводить до чотиризначної логіки (логіка Льюїса) на решітках:

X	¬X
0	3
1	2
2	1
3	0

□X	◊X
0	0
0	3
0	3
3	3

X&Y	0	1	2	3
0	*0*	*0*	*0*	*0*
1	*0*	*1*	*0*	*1*
2	*0*	*0*	*2*	*2*
3	*0*	*1*	*2*	*3*

8.2.6. Релевантна логіка

Релевантна логіка – напрям у символічній логіці, який виник і розвивався як альтернатива класичній символічній логіці. У назві «релевантна» (терміни «релевантний», «релевантність» є калькою з англійської, і їх можна перекласти як «доречний», «що стосується справи», «доречність») знайшло відображення та обставина, що в ній виключаються властиві класичній логіці принципи, які з погляду інтуїції та, головне, реальної практики міркувань трактуються як недоречні, що не відповідають цій практиці, парадоксальні. Релевантна логіка відрізняється від класичної логіки у двох основних пунктах. По-перше, в об'єктну мову обчислень вводиться імплікація, що інтенсійно розуміється, істиннісне значення якої на відміну від екстенсіональної матеріальної імплікації не детермінується істиннісними значеннями зв'язуваних висловлювань. Технічно для імплікацій, що фігурують в релевантній логіці, інтенсивність означає, що принципів, аналогічних відомим парадоксам матеріальної імплікації $A \rightarrow (B \rightarrow A)$ і $A \rightarrow (\neg A \rightarrow B)$, в релевантній логіці немає. По-друге, в релевантній логіці, щоб прийняти метатвердження про відношення логічного слідування між A і B (символічно: $A \vDash B$), недостатньо того факту, що B тотожно істинно, або A тотожно хибно (суперечливо). Відповідно в релевантних обчисленнях немає і теорем виду $A \rightarrow B$, де B є теорема, а A - довільна формула, або A є запереченням теореми, а B - довільної формулою.

Символічна релевантна логіка ближче до тієї логіки, що використовується у нормальних міркуваннях. Водночас у ній, порівняно із класичною, виникають серйозні семантичні проблеми. Наприклад, відкидаючи твердження $A, \neg A \vDash B$ про виведення довільного B з протиріччя A , $\neg A$, необхідно обґрунтувати відкидання твердження про логічне слідування $\neg A , A \vDash B$ у семантичному сенсі, а також мати семантику, в якій формули виду $\neg A \& A \rightarrow B$, $\neg (A \rightarrow A) \rightarrow B$, $A \rightarrow (B \rightarrow B)$ і т.п., антецеденти яких суперечливі, або консеквента загальнозначимі, не були б семантично щирими. Технічні рішення, як було зазначено, знайти вдалося. Однак із змістовної точки зору запропоновані семантики виглядають дуже штучними. У семантику можливих світів довелося вводити «неможливі

можливі світи» та тернарне (замість звичайного бінарного) ставлення досяжності одних світів з інших.

8.2.7. Світи Кріпке

Для формального висловлювання своїх ідей **Кріпке**[22] вводить відношення досяжності – деяке бінарне відношення R, що визначається на множині можливих світів. Нехай *a* та *b* – можливі світи. Тоді, якщо має місце *a* R *b*, то ці світи пов'язані: зі світу *a* можна досягти світу *b*. В іншому випадку це виявляється неможливим. Формальні семантики для різних обчислень розрізняються тепер лише властивостями відношення R. Так, щоб отримати адекватну семантику для S4, достатньо припустити, що відношення R є рефлексивним і транзитивним. Якщо додатково припустити симетричність цього відношення, отримаємо адекватну семантику для S5. У цьому останньому випадку кожен можливий світ можна досягти з кожного, і потреба у спеціальному відношенні досяжності відпадає. Запропонована Карнапом (Рудольф Карнап (нім. Rudolf Carnap, 18.05.1891- 14.09.1970) — німецький, а після 1935 року американський філософ-позитивіст, член Віденського гуртка) формальна модальна семантика відповідає цьому окремому випадку і годиться, отже, тільки для S5.

Далі, для кожної предикатной моделі кожен світ w з множини можливих світів W, у якому визначено бінарне відношенення досяжності R, характеризується порожньою множиною D_w індивідів, що у цьому світі. Існує також виділений елемент w*, званий дійсним світом. Для різних w множини D_w можуть бути різними.

З цієї точки зору зрозуміло, чому принцип підстановки тотожного та екзистенційне узагальнення вимагають обмежень у модальних контекстах. Якщо індивідуальні константи чи змінні перебувають у сфері дії модального оператора, всі вони можуть означати той самий індивід у дійсному (виділеному) світі, але різні індивіди в інших можливих світах (а в якихось світах нічого не позначати). Тому, щоб зазначені принципи були застосовними в модальних контекстах, кожен індивідуум, що входить у цей контекст, повинен позначати один і той же об'єкт у всіх світах, пов'язаних з даним світом відношенням R.

Швидке зростання числа модальних обчислень у 70-80-ті роки поставив питання про створення більш загальної та багатшої за своїми можливостями формальної семантики, ніж семантика Кріпке. Один із шляхів створення такої семантики пов'язаний з ім'ям З. Стахняка. Його основна ідея є елегантною і

---

[22] Кріпке (Kripke) Сол Аарон (р.р.1940, Бей Шор, шт. Нью-Йорк) - американський філософ і логік. Закінчив Гарвардський університет. Викладав у Гарвардському та Рокфеллерівському університетах (1963–76), з 1977 – професор Прінстонського університету.

простою, хоча її реалізація технічно може бути дуже складною. Семантика Кріпке є теоретико-множинною. Кожен «можливий світ» є просто позбавлений внутрішньої структури елемент деякої множини, якому (елементу) у предикатних інтерпретаціях приписано ще одну множину – множину індивідів, допустимих у цьому світі. Вся її образотворча сила визначається тому лише властивостями відношення R. Якщо вдасться наділити і самі елементи внутрішньою структурою, то образотворча міць формальної семантики різко зросте.

Для реалізації цієї ідеї Стахняк використав поєднання алгебраїчних та теоретико-множинних підходів. На вихідній множині, що розглядається як алгебраїчний об'єкт, можна побудувати вторинну множину алгебраїчних структур (наприклад, ультрафільтрів). На новій множині процес можна повторити, отримуючи множину елементів з багатшою структурою, а потім побудувати на ньому відношення досяжності R. Тим самим ми отримуємо семантику можливих світів, в якій на відміну від семантики Кріпке елементи базисної множини можуть бути наділені якою завгодно складною внутрішньою структурою.

Питання до розділу 8

1. Багатозначні логіки.
2. К-значна логіка Посту. Симетричне і циклічне заперечення.
3. Тризначна система Гейтинга і Лукасевича - загальне і відмінності.
4. Модальні логіки.
5. Логіка можливого. Оператори необхідності і можливості.
6. Строга імплікація.

9. ДОСТОВІРНІ ТА ПРАВДОПОДІБНІ ВИСНОВКИ

9.1. Достовірні та недостовірні висновки

Достовірний висновок

Будемо називати логічний висновок *достовірним,* якщо він задовольняє наступній умові: з множини посилок, що мають виділені істинні значення, виведені висновки, також мають виділені істинні значення.

Для двозначної логіки це означає, що з істинних посилок не виводяться хибні висновки, інакше кажучи, з істинних посилок виводяться тільки істинні висновки.

Елементарний акт дедукції – правило виведення *modus ponens* (MP) – є *достовірним правилом виведення*. Якщо правило "з *A* слідує *B*" покладається істинним (тобто формула $A \to B$ істинна), і істинна посилка *A,* то істинний і наслідок *B*:

$$\frac{\mathbf{A},\ \mathbf{A \to B}}{\mathbf{B}}$$

Висновок в аксіоматичних формальних теоріях – обчисленні висловлювань та обчисленні предикатів 1-го порядку, де використовуються лише достовірні правила, – є достовірним висновком.

Недостовірний висновок

Правило висновку називають *недостовірним*, якщо з виділення всіх посилок виділеність слідства випливає не завжди.

Логічний висновок називається *недостовірним* , якщо при побудові цього висновку було застосовано принаймні одне недостовірне правило виведення.

Наприклад, *при недостовірному висновку* про посилку, ґрунтуючись на істинності висновку, міркують за наступною схемою *абдукції*:

$$\frac{\mathrm{B},\ \mathrm{A} \to \mathrm{B}}{\mathrm{A}}$$

У цьому висновку, згідно з визначенням імплікації: $T \to T = T$, $F \to T = T$ – при істинності правила $A \to B$ та істинності висновку *B*, посилка *A* може бути як істинною, так і хибною.

Відмінність між правилами достовірного та недостовірного висновку

Дедуктивний логічний висновок є достовірним. Для доказу достовірності виведення $\Gamma \models B$ досить показати, що немає таких умов (інтерпретацій), у яких формула *B,* виведена з істинних посилок *Г*, може бути хибною.

На цьому засновані метод редукції (від протилежного) та метод спростування (резолюцій).

Якщо існує множина умов *M*, при яких формула *B* може стати хибною, то така множина умов *M* називається множиною *фальсифікаторів* формули *B*.

Для достовірного виведення множина фальсифікаторів порожня, для недостовірного виведення – не порожня.

Наприклад, у двозначній логіці висловлювань, якщо формула *B* недоведена, можна ефективно породити її контроцінки, тобто знайти такі значення змінних, що входять до неї, при яких вона набуває хибних значень. Контроцінку називають *фальсифікацією* формули *B*. Якщо формула *B* доведена, вона може бути фальсифікована.

Правило недостовірного висновку називається *правдоподібним*, якщо для відомої (до теперішнього часу) множини можливих фальсифікаторів формули *B* жоден з них не робить формулу *B* хибною.

Іншими словами, висновок є правдоподібним, якщо множину фальсифікаторів не знайдено, але потенційно вона може бути не пустою. Тут суттєвий той факт, що знання про світ (про проблемну галузь) неповно, тобто множина можливих фальсифікаторів відкрита.

Приклад недостовірного висновку дає *перечислювальна індукція* (індукція через простий перелік).

Нехай є певна умова, задана формулою логіки предикатів $\varphi(x)$, і нехай в результаті спостережень встановлена істинність висловлювань $\varphi(a_1)$, $\varphi(a_2)$, ..., $\varphi(a_n)$. І нехай невідомий такий випадок, що умова $\varphi(x)$ стає неправдивою, тобто невідомо таке $b \neq a_i$ для всіх a_i, щоб $\varphi(b)$ ставало хибним. Тоді висновок

$$\varphi(a_1), \varphi(a_2),\ldots, \varphi(a_n) \models \forall x \varphi(x)$$

є недостовірним висновком, що не має, однак, якогось відомого фальсифікатора, тому що стверджувати, що множина фальсифікаторів порожня, ми не маємо підстав. Такий висновок є *правдоподібним*.

9.2. Види міркувань

Згідно з Чарльзом Пірсом існує три види елементарних міркувань: ***дедукція, індукція та абдукція***. Взагалі ще є редукція і традукція, але це на основі трохи іншого принципу класифікації.

Редукція (лат. reductio - зведення, приведення назад) - логічний прийом перетворення будь-яких даних до більш зручного з будь-якої точки зору виду; зведення складного до більш простого, доступного для аналізу чи рішення.

Традукція (лат. traductio - переміщення) - вид опосередкованого висновку, в якому посилки і висновок є міркуваннями однакового ступеня спільності. Традуктивним висновком є аналогія. За характером посилок та висновку традукція може бути трьох типів:

- *висновок від одиничного до одиничного,*
- *висновок від приватного до приватного,*
- *висновок від загального до загального.*

Дедукція (лат. deductio - виведення, а також дедуктивний висновок, силогізм) - спосіб мислення, наслідком якого є логічний висновок, істинність якого гарантується істинністю посилок. Також може визначатися логіко-методологічна процедура, за допомогою якої здійснюється перехід від загального до приватного у процесі міркування.

Початком (посилками) дедукції є аксіоми чи навіть гіпотези, які мають характер загальних тверджень («загальне»), а кінцем — наслідки з посилок, теореми («приватне»). Якщо посилки дедукції є істинними, то істинними є і її наслідки. Дедукція – основний засіб логічного доказу; протилежно до індукції.

Індукція (лат. inductio - наведення, від. inducere - тягнути за собою, встановити) - процес логічного висновку на основі переходу від приватного становища до загального. Індуктивний висновок пов'язує часні передумови з висновками не строго через закони логіки, а скоріше через деякі фактичні, психологічні чи математичні уявлення.

Абдукція (від лат. ab - "с, від" і лат. ducere - "водити") - пізнавальна процедура висування гіпотез. Абдукція є видом редуктивного висновку з тією особливістю, що з посилки, яка є умовним висловом, і висновку випливає друга посилка. Наприклад:

- *перша посилка: люди - смертні;*
- *висновок: Сократ – смертний.*

Ми можемо припустити, за допомогою абдукції, *що друга посилка: Сократ - людина.*

В історії логіки ідея абдукції у формі апагогії (доведення до абсурду) перегукується з Аристотелем. В даний час абдукція вперше розглянута основоположником прагматизму і семіотики Ч. С. Пірсом, який систематично використовує термін з 1901 року.

Абдукція має широке поле наукового та прикладного використання, у тому числі, в системах штучного інтелекту – абдуктивні міркування найчастіше використовуються для відкриття емпіричних законів, які встановлюють необхідні регулярні зв'язки між спостережуваними властивостями та відношеннями явищ.

Як відбуваються дедуктивні та індуктивні міркування? У чому видно їх відмінності?

Як зазначалося, дедукція це – спосіб міркування від загальних положень до окремих висновків. Дедуктивне міркування лише конкретизує наше знання. У дедуктивному висновку міститься лише інформація, що є у прийнятих посилках. Дедукція дозволяє з знання отримувати нові істини за допомогою чистого міркування.

Дедукція дає стовідсоткову гарантію правильного висновку (при достовірних посилках):

Усі метали пластичні (велика достовірна посилка чи основний аргумент).

Вісмут - метал (достовірна посилка).

Отже, вісмут пластичний (правильний висновок).

Індукція[23] - спосіб міркування від часних положень до загальних висновків. В індуктивному висновку може бути інформація, відсутня в прийнятих посилках. Достовірність посилок не значить достовірність індуктивного висновку. Посилки надають висновку більшу чи меншу правдоподібність.

Індукція дає не достовірне, а ймовірнісні знання, що потребує перевірки. З часних положень не випливає логічно загальний висновок.

Варіант індукції - висновок за аналогією (з урахуванням подібності двох об'єктів за одними параметрами робиться висновок про їх схожість також і за іншими параметрами).

Планети Марс та Земля багато в чому схожі. На Землі є життя. Оскільки Марс схожий на Землю, на Марсі також є життя.

Цей висновок є, звісно, лише ймовірним. Будь-який індуктивний висновок потребує перевірки.

Абдукція – пізнавальна процедура прийняття гіпотез. Вперше явно виділено Ч.С.Пірсом, який розглядав абдукцію (абдуктивний висновок) поряд з індукцією та дедукцією. Ч.С.Пірс вважав, що, відбираючи серед неосяжної множини гіпотез найбільш суттєві, дослідники реалізують «абдукційний інстинкт», без якого неможливий був б розвиток науки. Згідно з Пірсом, методологія науки повинна розумітися як взаємодія 1) абдукції, що здійснює прийняття пояснювальних правдоподібних гіпотез; 2) індукції, що реалізує емпіричне тестування висунутих гіпотез; 3) дедукції, з якої з прийнятих гіпотез виводяться слідування. Таким чином, Ч.С.Пірс створив ідейний ескіз теорії міркувань, що згодом отримав розвиток у дослідженнях із штучного інтелекту, в яких абдуктивний висновок представлений як вид автоматизованого правдоподібного міркування. Ідея абдукції, згідно з Ч.С.Пірсом, може бути сформульована наступним чином:

- *D – множина фактів,*
- *H – множина висунутих гіпотез,*
- *H пояснює D.*

Отже, гіпотези з H правдоподібні.

Для послідовного здійснення та посилення цієї ідеї потрібно формалізувати як процедуру висування гіпотез з H, так і відношення «H пояснює D». Крім того, необхідно було конструктивно встановити процедуру

[23] Див.«Парадокс воронів», відомий також як парадокс Гемпеля або ворони Гемпеля - парадокс підтвердження, сформульований німецьким математиком Карлом Густавом Гемпелем у 1940-х роках, для ілюстрації того, що індуктивна логіка іноді входить у протиріччя з інтуїцією.

оцінки правдоподібності гіпотез, породжених за допомогою індукції на основі фактів з D.

У ряді робіт з автоматизованих правдоподібних міркувань було встановлено, що формалізація абдукції як конструктивної аргументації можлива через взаємодію останньої з індукцією та аналогією, причому аргументи породжуються за допомогою індукції, передбачення за допомогою аналогії, а прийняття гіпотез здійснюється за допомогою абдукції. Для цього уточнення абдукції (у сенсі Ч.С.Пірса) використовуються багатозначні логіки.

У роботах із штучного інтелекту поширена наступна формалізація абдукції засобами двозначної логіки предикатів першого порядку:

Нехай D - множина фактів, що спостерігаються, T - деяка задана теорія, H - множина гіпотез. Тоді множина висловлювань E називається абдуктивним поясненням D, якщо тільки для нього виконуються такі умови:

(1) E міститься в H,

(2) з об'єднання T та E виводиться D,

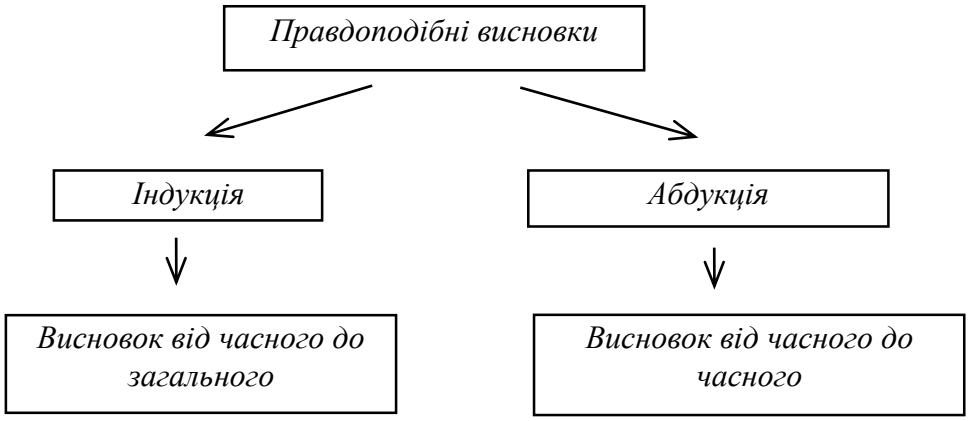
(3) T і E – несуперечливі.

Завдання отримання абдуктивного пояснення зводиться до знаходження E. Вказана вище формалізація абдуктивного пояснення не є достатньо глибокою імітацією ідеї Ч.С.Пірса про абдукцію як пізнавальну процедуру, властиву творчій активності людини. Є цікаві зв'язки між абдукцією та немонотонними міркуваннями, а також формалізаціями, діагностичних процедур, представлених у системах штучного інтелекту.

9.3. Класифікація недостовірних висновків

До *приближених висновків* відносять такі, у яких оцінюється **ступінь істинності** висновку. Прикладами приближених міркувань є висновки у ймовірнісній логіці (байесовський висновок, наприклад) або нечіткі висновки.

Було наведено класифікацію на основі існування певної множини фальсифікаторів. Розглянемо класифікацію правдоподібних висновків на підставі спільності посилок та висновків.



Приклад абдуктивного висновку:

a* подібно до *b*, *φ(a)

φ(b)

Очевидно, що існування множини фальсифікаторів цілком можливе, і якщо невідомий спосіб їхнього породження, цей висновок є правдоподібним. Очевидно також, що якби не було посилки «*a* подібно до *b»,* то правдоподібність висновку була б меншою.

Можна уточнити цю схему абдукції:

a* подібно до *b*, *φ(a)

φ(*b*) більш правдоподібно

9.4. Особливості правдоподібних висновків

Можна навести *особливості правдоподібних висновків (за В.К. Фінном):*

По-перше, правдоподібний висновок не є достовірним.

По-друге, хоча правдоподібний висновок не є достовірним, *він не повинен бути тривіально недостовірним,* як, наприклад: $\psi, \varphi \rightarrow \psi \models \varphi$, – бо він призначений виражати деяку зручну евристику.

Дві наведені вище схеми міркування відрізняються від тривіального недостовірного висновку *залежністю висновку від контексту вживання посилок*. Таким чином, **третьою характеристикою правдоподібних висновків** є їхня *контекстна залежність від умов вживання посилок*.

Четвертою характеристикою правдоподібних висновків є *неповнота* інформації, яка змушує використовувати посилки із істинистим значенням "не визначено".

Неповна інформація означає *нездійсненність закону виключення третього*, як засобу класифікації знань на істинні та хибні. Тоді для формалізації неповноти інформації можна використовувати *ймовірнісну* або *нечітку* логіку, *багатозначні* або *модальні* логіки.

Якщо уточнення неповноти інформації перекласти на логічну мову, **то п'ятою характеристикою** правдоподібних висновків буде *недвозначність логіки*, яка використовується для формалізації правил правдоподібного висновку.

Неповнота інформації, за умов якої відбувається процес міркування, що використовує правдоподібні висновки, природно пов'язана з *відкритістю* системи знань про предметну область, щодо якої проводиться міркування, що включає правдоподібні висновки.

Таким чином, **шостою характерною рисою правдоподібних висновків** є *відкритість множини тверджень*, що представляють систему знань про предметну область, які використовуються як посилки виведених правдоподібних висновків.

Сьомою характерною рисою правдоподібних висновків є *немонотонність*.

Виведення формули φ називається *монотонним*, якщо має місце таке:

якщо Г |= φ, то Г, ψ|= φ.

В іншому випадку висновок називається *немонотонним* (тобто Г |= φ, але при додаванні ψ φ стає невиводимою).

Очевидним є природний ***зв'язок*** *немонотонності* виведення в умовах неповноти інформації, що входить у міркування, та *відкритості* систем. Зокрема, немонотонним висновком є висновок щодо схеми індукції через просте перерахування:

$$\frac{\varphi(a_1),\ldots,\varphi(a_n)}{\forall x\ \varphi(x)}$$

Додавання нового факту $a_{n+1} \neq a_i$ може спричинити фальсифікацію індуктивного узагальнення $\forall x\ \varphi(x)$. Тоді a_{n+1} є фальсифікатором твердження $\forall x\ \varphi(x)$.

Восьмою характерною особливістю правдоподібних висновків є використання *подібності* об'єктів, що розглядаються, або ситуацій для виведення правдоподібних висновків, що містять *нове знання* про предметну область. Це означає, що виявлена схожість є креативним джерелом правдоподібних висновків. Ця властивість правдоподібних міркувань є основою для індуктивних методів Д. С. Мілля.

9.5. Методи індуктивних міркувань Д.С. Мілля

Будь-який силогізм (дедуктивне міркування), на думку Мілля[24], містить в собі petitio principii[25]; будь-який силогістичний висновок йде насправді від часного до часного, а не від загального до часного. Розглядаючи індукцію, Мілль, по-перше, задається питанням про заснування чи право на індуктивний висновок і бачить це право в ідеї одноманітного порядку явищ, і, по-друге, зводить всі способи висновку в індукції до чотирьох основних:

- метод подібності - згоди (якщо два або більше випадки досліджуваного явища сходяться в одній тільки обставині, то ця обставина і є причиною або частиною причини досліджуваного явища,

- метод розходження - відмінності (якщо випадок, у якому зустрічається досліджуване явище, і випадок, у якому воно не зустрічається, абсолютно подібні у всіх подробицях, за винятком досліджуваної, то обставина, що зустрічається в першому випадку і відсутня у другому, і є причина або частина причини досліджуваного явища);

- метод залишків (якщо в досліджуваному явищі частина обставин може бути пояснена певними причинами, то частина явища, що залишилася, пояснюється з попередніх фактів, що залишилися) і

- метод супутніх змін (якщо за зміною одного явища помічається зміна іншого, ми можемо укласти причинний зв'язок поміж них).

Ці методи при найближчому розгляді виявляються дедуктивними методами; наприклад, метод залишків є саме, як визначення шляхом виключення. Формалізуємо ДСМ-методи.

1. Метод подібності

a, b, c	$\Rightarrow$	d
a, e, f	$\Rightarrow$	d
…	…	
a, l, m	$\Rightarrow$	d
a	$\Rightarrow$	d

[24] **Джон Стю́арт Мілль** (1806 — 1873) — британський філософ, політичний економіст, теоретик протофемінізму. Основу вчення Мілля про науковий метод складає його теорія індукції та суто пізнавальне питання: як ми можемо обґрунтувати своє знання, згідно з яким властивості, що притаманні обмеженій кількості певних явищ, притаманні і всім явищам подібного роду?

[25] Логічна помилка - «**передбачення основи**» (лат. *petitio principii*). Вона полягає в тому, що у якості аргументів використовуються недоведені, як правило, довільно взяті положення. Насправді ж доброякісність таких аргументів лише передбачається, але не встановлюється з безсумнівністю. Зазвичай подібні хибні аргументи супроводжуються фразами: «як абсолютно всім відомо…», «зрозуміло, що…», «кожному відомо, що…», щоб розсіяти можливі сумніви у простого слухача.

Наслідок *d* виникає щоразу, як спостерігається факт *a,* тому можна припустити, що саме *a* є причиною *d,* хоча насправді це може бути не так.

2. Метод розходження.

a, b, c	⇒	d
…	…	…
a, l, m	⇒	d
	ні	
b, c	⇒	d
…	…	…
	ні	
l, m	⇒	d
a	⇒	d

Цей спосіб посилює спосіб подібності. Поряд з позитивними прикладами, включаються негативні, що підтверджують, що інші факти, що спостерігалися при появі *d,* не призводять до появи *d,* якщо відсутній факт *a.*

3. Метод залишків.

a, b, c	⇒	d, e, f
a	⇒	d
h	⇒	e
c	⇒	f

4. Метод супутніх змін.

Нехай є дві множини *A* і *B* , $a_i \in A$, $b_i \in B$, $c \notin A$, $c \notin B$.

		b
a_1, c	⇒	1
		b
a_2, c	⇒	2
…	…	…
		b
a_n, c	⇒	n
A	⇒	B

За допомогою цього робиться звичний крок індукції: від підтверджуючих прикладів ми переходимо до узагальненого твердження, що стосується об'єктів всього класу.

ДСМ-міркування є синтезом пізнавальних процедур: індукції, аналогії та абдукції. В кінці сімдесятих років В. К. Фіном[26] був запропонований ДСМ-метод автоматичного породження гіпотез, який формалізує схему правдоподібного і достовірного висновку, так зване ДСМ-міркування. Назву методу становлять ініціали саме Джона Стюарта Мілля, чиї «методи розсудливого натураліста» частково формалізовані в методі.

Питання до розділу 9

1. Достовірні міркування.
2. Які міркування називаються недостовірними?
3. Що таке фальсифікатор міркування?
4. Яке міркування називається тривіально недостовірним?
5. Класифікація недостовірних висновків на основі існування множини фальсифікаторів.
6. Класифікація правдоподібних висновків на підставі спільності посилок і висновку.
7. Індукція і абдукція. Нетривіальна абдукція.
8. Особливості правдоподібних висновків.
9. Методи індуктивних міркувань Д.С. Мілля.

[26] **Віктор Костянтинович Фінн** (нар. 1933) - радянський і російський філософ. Розробив концепцію представлення інтелектуальної діяльності за допомогою квазіаксіоматичних (відкритих) теорій, що реалізують правдоподібні міркування у вирішувачах задач типу «правдоподібний висновок + достовірний висновок». У вирішувачах цього типу використовується ДСМ-метод автоматичного породження гіпотез, який формалізує та розширює індуктивні методи Д. С. Мілля. В.К.Фінн сформулював новий клас багатозначних логік, що є формалізаціями процедур аргументації; показав, що ДСМ-метод автоматичного породження гіпотез є варіантом синтезу пізнавальних процедур: індукції, аналогії, абдукції та дедукції; встановив, що ДСМ-метод автоматичного породження гіпотез є каузальною аргументацією та конструктивною абдукцією, що уточнює ідею абдуктивного виведення у сенсі Ч. С. Пірса.

10. НЕЧІТКІ МНОЖИНИ

Нелегко складалася доля нового наукового підходу. Неоднозначно оцінювалася «нечітка логіка» в різних колах і на різних рівнях. Так, ще в 1989 році Національний науковий фонд США ставив питання про виключення її з університетських підручників. А всього через рік, COCOM (Комітет з контролю над експортом) вніс технології, розроблені на її основі, в список критично важливих оборонних технологій, і заборонив їх експорт.

У чому ж суть нового підходу? Багато хто вважає «нечітку логіку» (fuzzy logic) третьою хвилею інтелектуального програмування. Інші - авантюрою і аферою. В основі нової науки лежить теорія нечітких множин, викладена в 1965-1973 роках в серії робіт відомого американського математика Лотфі Заде (Lotfi Zadeh). До речі його сім'я виїхала свого часу з СРСР - з Азербайджану.

Вже тоді були популярні експерименти з «мажоритарними просторами». У них навмисно усувалося поняття міри: замість нього вводився ряд якісних факторів (типу квантора «більшості»). Це був прообраз перших «нечітких тверджень». У той час існуючі експертні системи викликали ряд нарікань і залишали бажати кращого. Тому і з'явилося соціальне замовлення на дослідження подібного роду.

Зростаючий «штучний інтелект», який легко справлявся із завданнями управління складними технічними комплексами, пасував перед найпростішими поняттями з повсякденного життя. Для створення дійсно інтелектуальних систем необхідним був новий математичний апарат, що переводить невиразні і неоднозначні життєві твердження в мову чітких і формальних математичних формул. Тільки так ставало уможливити адекватну взаємодію машини і людини.

В теорії нечітких множин функція приналежності елемента («належить» або «не належить») являє собою не жорстку умову, а плавну сигмоїду (часто спрощується ламаною лінією). Вона пробігає всі значення від нуля до одиниці.

Поняття «нечіткої множини» кожному знайоме на побутовому рівні. Будь-яка людина його інтуїтивно розуміє. Ми стикаємося з цим поняттям повсякденно. Більшість використовуваних нами понять за своєю природою нечіткі і розмиті. Який момент можна вважати початком життя людини? Яка грань відрізняє худу людину від товстої? Наскільки добрий прибуток відрізняється від середнього? Коли ми говоримо «пригальмуй» або «додай ходу», то не вкладаємо в ці слова абсолютного конкретного сенсу. Їх не можна висловити двійковій логікою.

Конкретні числові рамки або неприпустимо огрубляють предметну область, або надмірно ускладняють рішення задачі. А fuzzy logic пропонує елегантне і просте рішення для подібних ситуацій. Спочатку описується необхідне якісне поняття («великий», «хороший», «сильний», «молодий», «розумний», «популярний») деякою функцією розподілу. Вона за своєю

природою подібна ймовірнісним функціям. І потім використовується як точне визначення, не піклуючись про його нечіткості. Теорія fuzzy logic дозволяє виконувати над такими величинами весь спектр необхідних логічних операцій: об'єднання, перетин, заперечення та ін.

Послідовник і учень Заде, живий класик «нечіткої логіки» Барт Коско (Bart Kosko) в своїй книзі «Нечітке мислення» («Fuzzy Thinking») вважав, що два тисячоліття тому людство крупно помилилося. У фундамент науки потрібно було закласти не суху двійкову логіку Аристотеля, а «нечітку логіку» і поетику ранніх східних філософій. І з тих самих пір класична «чорно-біла» бінарна наука, яка затиснута законом «виключення третього», все далі і далі віддаляється від нашого реального багатозначного світу. Адже в ньому немає нічого абсолютного, а все найцікавіше укладено, як раз, між «так» і «ні». У своїй знаменитій теоремі FAT («Fuzzy Approximation Theorem») Коско довів, що будь-яка математична система може бути апроксимована системою, заснованої на «нечіткій логіці».

Але, апарат теорії нечітких множин, продемонструвавши ряд багатообіцяючих можливостей застосування: від систем управління літальними апаратами до прогнозування підсумків виборів, виявився, разом з тим, надмірно складний у втіленні для свого часу. І на багато років fuzzy logic зайняла своє місце в ряду інших спеціальних наукових дисциплін - десь посередині між експертними системами і нейронними мережами.

Друге народження fuzzy logic відноситься якраз до часу проведення експерименту фінансовою корпорацією «Yamaichi Securities». Відразу кілька груп дослідників (США і Японії) всерйоз зайнялися створенням електронних систем різного застосування, що використовують нечіткі керуючі алгоритми. Теоретичні основи для цих спроб були закладені в ранніх працях Коско (якому в ту пору було 24 роки) та інших вчених.

Велику роль, ймовірно, зіграли два отриманих видатних наукових результати: доказ FAT-теореми, що дала «нечіткій логіці» право на життя і комбінація «нечіткої логіки» з нейронними мережами Кохонена. Вона вказала шлях до вирішення проблеми нової теорії: автоматизованого формування системи «нечітких правил» у вмісті вхідних даних.

10.1. Визначення нечіткої множини

Розглянемо кінцеву множину $E = \{ x_1, x_2, ..., x_n \}$ і $L = \{0,1\}$. Тоді $L^E = 2^E$ є множина-степінь, тобто множина всіх підмножин множини E, включаючи $\varnothing$.

Визначення 10.1. Нехай E - універсальна множина і L - решітка. Нехай $\alpha \in L$. Нечітка підмножина $A \subset E$, або, що еквівалентно, $A \in L^E$ - це така підмножина, що кожному елементу $x \in E$ можна

поставити у відповідність елемент $\alpha \in L$. Цей елемент позначають $\mu_A(x)$ і називають *функцією приналежності.*

У разі, якщо решітка L є замкнений інтервал [0, 1] на множині дійсних чисел, що представляє собою ланцюг, то зведення його в степінь довільної множини E дає множину нечітких підмножин в сенсі Заде. Тоді функція приналежності $\mu_A(x)$ приймає значення з інтервалу [0, 1] і визначає степінь, з якою елемент x належить нечіткій множині A. Зокрема, якщо $\mu_A(x) = 0$, то елемент x не належить нечіткій множині A, якщо $\mu_A(x) = 1$, то елемент x належить нечіткій множині A зі степенем 1, якщо $\mu_A(x) = 0.6$, то елемент x належить нечіткій множині A зі степенем 0.6 і т.п.

Таким чином, *нечітка множина в сенсі Заде* є відображення $[0, 1]^E : E \to [0, 1]$.

Приклад 10.1. Нехай множина E є множиною чисел, що позначають *вік* людини, наприклад, в межах від 0 до 100. Розглянемо такі поняття, як *молодий* і *старий.* Очевидно, важко встановити якусь точну межу, де закінчується вік *молодості* і настає вік *старості.* Якщо ми введемо ще поняття *середній вік,* це нам не допоможе. Ми знову розійдемося в думках про рамки вікових категорій. Ми можемо вказати ці кордони приблизно, і, опитавши, наприклад, кілька людей, встановити, з якою *степенню* можна віднести той чи інший вік до категорії *молодий* або *старий.* Результати цих опитувань можна буде зобразити у вигляді графіка (рис. 10.1). Значення функції приналежності з інтервалу [0, 1] показують, з яким степенем той чи інший вік належить віку *молодих* або *старих.* Наприклад, вік *40 років* зі степенем 0.48 віднесений до поняття *молодий* і зі степенем 0.36 - до поняття *старий.*

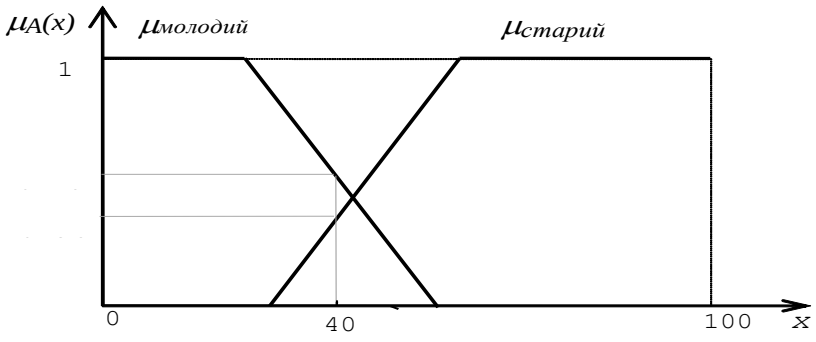


Рис.10.1. Функції приналежності нечітких множин.

Приклад 10.2. Нехай $E = \{a, b, c\}$. $A = \{a/0.2, b/0.6, c/0.8\}$, $B = \{a/0.1, b/0.4, c/0.9\}$ є дві нечітких множини. Елемент a належить множині A зі степенем 0.2, і множині B зі степенем 0.1, і т.д.

Наведене вище визначення 10.1 є узагальненням поняття нечіткої множини, введеного Заде. Розглянемо узагальнення властивостей нечітких множин та операцій на них, якщо L - решітка.

10.2. Операції над нечіткими множинами

Визначення 10.2. Якщо $\leq$ - відношення порядку на решітці L , E - деяка множина, $A \subseteq E$, $B \subseteq E$, то говорять, що A міститься в B, або A включено в B ($A \subseteq B$), якщо $\forall\, x_i \in E : \mu_A(x_i) \leq \mu_B(x_i)$.

Таким чином, дві нечіткі множини можна порівняти, якщо порівняти відповідні значення функцій приналежності і між двома нечіткими підмножинами існує відношення домінування. Наприклад, якщо A={a /0.2, b /0.6, c /0.8}, B={a /0.3, b /0.8, c /0.9}, то $A \subset B$.

Визначення 10.3. Дві нечітких множниA і B рівні тоді і тільки тоді, коли $\forall\, x_i \in E : \mu_A(x_i) = \mu_B(x_i)$.

Поняття доповнення в теорії нечітких множин Заде і в теорії решіток - різні.

Визначення 10.4. Доповнення нечіткої підмножини $A \in [0, 1]^E$ є нечітка підмножина B зі значеннями функції приналежності, такими що $\forall\, x_i \in E : \mu_B(x_i) = 1 - \mu_A(x_i)$.

Наприклад, якщо A = { a /0.2, b /0.6, c /0.8}, то доповнення A - нечітка множина B = { a /0.8, b /0.4, c /0.2}.

Визначення 10.5. Операція перетину на решітці L індукує операцію перетину нечітких множин як $\forall\, x_i \in$ E: $\mu_{A \cap B}(x_i) = \mu_A(x_i) \wedge \mu_B(x_i)$. Для нечітких множин в сенсі Заде це визначення збігається з $\mu_{A \cap B}(x_i) = min\ \{\mu_A(x_i), \mu_B(x_i)\}$.

Наприклад, якщо A = {a /0.2, b /0.6, c /0.8}, B = {a /0.1, b /0.4, c /0.9}, то $A \cap B$ = { a /0.1, b /0.4, c /0.8}.

Визначення 10.6. Операція об'єднання на решітці L індукує операцію об'єднання нечітких множин як $\forall\, x_i \in E : \mu_{A \cup B}(x_i) = \mu_A(x_i) \vee \mu_B(x_i)$. Для нечітких множин в сенсі Заде це визначення збігається з $\mu_{A \cup B}(x_i) = max\ \{\mu_A(x_i), \mu_B(x_i)\}$.

Наприклад, якщо A = {a /0.2, b /0.3, c /0.8}, B = {a /0.1, b /0.4, c /0.9}, то $A \cup B$ = { a /0.2, b /0.4, c /0.9}.

Визначення 10.7. *Алгебраїчний добуток* нечітких множин A і B (позначається AB) визначається як арифметичний добуток їх функцій приналежності: $\mu_{AB}(x) = \mu_A(x)\, \mu_B(x)$.

Визначення 10.8. *Алгебраїчна сума* нечітких множин A і B (позначається $A + B$) визначається як

$$\mu_{A+B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\, \mu_B(x),$$

де + і – є операції арифметичного додавання і віднімання відповідно.

10.3. Нечіткі відношення

При описі суджень не менш важливе місце займають нечіткі відношення на множинах. Нечітким бінарним відношенням R_A на множині X називається нечітка підмножина прямого добутку X × X, що характеризується функцією приналежності $\mu_R: X \times X \to [0,1]$.

Значення μ_R для конкретної пари $(x_i, x_j) \in X \times X$ характеризує суб'єктивну міру або степінь виконання відношення $x_i \, R_A \, x_j$.

Якщо множина X звичайна і невелика, нечітке відношення R_A зручно задавати в матричному вигляді. Матриця $M(R_A)$ являє собою квадратичну матрицю, рядки і стовпці якої позначені елементами $x \in X$, а в осередках записані значення $r_{ij} = \mu_R (x_i, x_j)$.

У загальному випадку нечітке *n–арне* відношення на множині *X* виражено декартовим добутком $x_1 \times x_2 \times \ldots \times x_n$, що визначається R_A з функціями приналежності $\mu_R (x_1, x_2, \ldots, x$.

Носієм нечіткого відношення R_A на множинах *X* і *Y* називається підмножини декартового добутку $X \times Y$ виду:

$$supp = \{(x, y) : (x, y) \in X \times Y, \mu_R(x, y) > 0\},$$

тобто носій нечіткого відношення можна розглядати як звичайне відношення, що зв'язує всі пари $(x, y) \in X \times Y$, для яких степінь виконання нечіткого відношення *R* не дорівнює нулю.

Більш корисним є використання α-перетинів нечіткого відношення, визначення яких аналогічно визначенню для множин α- рівня:

α- *перетином нечіткого відношення R* на $X \times Y$ називається звичайне відношення, що зв'язує всі пари $(x, y) \in X \times Y$, для яких степінь виконання нечіткого відношення *R* НЕ менше α: $R = \{(x, y) : (x, y) \in X \times Y, \mu_R(x, y) \geq \alpha\}$.

Нечітке відношення *R* на $X \times X$ називається *рефлексивним,* якщо для будь-якого $x \in X$ виконується рівність $\mu_R(x, x) = 1$. У разі кінцевої множини *X* всі елементи головної діагоналі матриці *R* дорівнюють *1*. Прикладом рефлексивного нечіткого відношення може бути відношення «приблизно дорівнюють».

Нечітке відношення *R* на $X \times X$ називається *антирефлексивним,* якщо для будб-якого $x \in X$ виконується рівність $\mu_R(x, x) = 0$. У разі кінцевої множини *X* всі елементи головної діагоналі матриці *R* дорівнюють *0*. Прикладом антирефлексивного нечіткого відношення може бути відношення «значно більше».

Нечітке відношення R на $X \times Y$ називається *симетричним,* якщо для будь-якої пари $(x, y) \in X \times Y$ виконується рівність $\mu_R(x, y) = \mu_R(y, x)$. Матриця симетричного нечіткого відношення, заданого на кінцевій множині, симетрічна.

Нечітке відношення R на $X \quad Y$ називається *асиметричним,* якщо висловлювання $\mu_R(x, y) > 0 \Rightarrow \mu_R(y, x) = 0$ справедливо для будь-якої пари $(x, y) \in X \times Y$. Прикладом асиметричного нечіткого відношення може служити відношення «набагато більше».

Нечітке відношення R на $X \times X$ називається *транзитивним*, якщо для будь-якої трійки $(x, z), (y, z), (x, z) \in X \times X, x, y, z \in X$ виконується умова $\mu_R(x, z) \geq \max[\min(\mu_R(x, y), \mu_R(y, z))]$.

Нечіткі відношення R і R^{-1} на $X \times Y$ називаються *зворотними,* якщо для будь-якої пари $(x, y) \in X \times Y$ виконується рівність $\mu_R(x, y) = \mu_{R^{-1}}(y, x)$

Прикладом зворотних нечітких відношень може служити пара «набагато більше» - «набагато меньше».

Нечіткою множиною типу n називається нечітка множина, у якої значеннями функції приналежності є НМ типу *n-1*. Так, нечітка множина типу 1 є $\alpha: X \to [0, 1]$, а нечітка множина типу 2 є $\mu: X \times [0,1] \to [0, 1]$ і т.д.

10.4. Нечіткі та лінгвістичні змінні

Поняття змінної є основоположним математичним поняттям. Нечіткі і лінгвістичні змінні використовуються при природно-мовному описі різних об'єктів і явищ, при формалізації процесів прийняття рішень в важкоформалізуємих ситуаціях, в описі складних явищ.

Здатність людини оцінювати інформацію найбільш яскраво проявляється в використанні природних мов. Кожне слово x можна розглядати як стислий опис нечіткої підмножини $A(x)$ повної множини області міркувань X. Таким чином, $A(x)$ є значення x. У цьому сенсі всю мову можна розглядати як систему, відповідно до якої нечітким підмножинам множини X приписуються елементарні або составні символи, тобто слова, групи слів і речення.

Нечіткою змінною н а з и в а є т ь с я

$$< \alpha, X, C_\alpha > ,$$

де α - найменування нечіткої змінної; $X = \{x\}$ - область її визначення (базова множина); C_α - $< \mu_A(x) / x > \}$ - нечітка множина на X, що описує обмежені, але можливі значення нечіткої змінної α.

У більш загальному випадку значеннями таких змінних можуть бути слова або речення природної або формальної мови, і тоді відповідні змінні називають *лінгвістичними*. Так, наприклад, нечітка змінна *висота* могла б набувати таких значень: *високий, низький, доволі високий, дуже високий*

більш-менш високий, не високий, не дуже високий тощо. Ці значення є речення, що утворені поняттям *високий та* запереченням *ні*, спілками *і*, *але*, а також словами типу *дуже*, *досить*, *цілком*, *більш-менш*.

Лінгвістична змінна є змінною вищого порядку, ніж нечітка змінна, в тому сенсі, що значеннями лінгвістичної змінної є нечіткі змінні. Лінгвістичні змінні призначені в основному для аналізу складних або погано визначених явищ. Використання словесних описів типу тих, якими оперує людина, робить можливим аналіз систем настільки складних, що вони недосяжні звичайному математичному аналізу.

Лінгвістичною змінною н а з и в а є т ь с я

$$<\beta, T_\beta, X, G, M>,$$

де β - найменування лінгвістичної змінної;

T_β- множина значень (терм-множина), що представляють собою найменування нечітких змінних, областю визначення (базовою множиною) кожної з яких є *X;*

G - синтаксичне правило, породжує найменування $\alpha \in T_\beta$ вербальних значень лінгвістичної змінної β;

M - семантичне правило, яке ставить у відповідність кожній нечіткій змінній $\alpha \in T_\beta$, нечітку множину C_α, тобто сенс (семантику) нечіткої змінної α.

Процедура утворення нових термів *G* здійснюється за допомогою логічних зв'язок: *ні*, *і* та *або* і модифікаторів типу «*дуже»*, *«злегка»* т.п.. Процедуру завдання нечітких змінних *M*, наприклад, "мала швідкість», «середня швидкість», «висока швидкість», а так само відповідних нечітких множин для термів з $G(T)$ виконується відповідно до правил трансляції нечітких зв'язок і модифікаторів. Правило *M* може бути виконане по одній з типових операцій над нечіткими множинами.

10.5. Методи побудови функції приналежності

Існує кілька точок зору на змістовну інтерпретацію функцій приналежності, при яких переважаючою оцінкою степеня приналежності є фіксування конкретних значень в інтервалі *[0,1]*. Таке фіксування носить суб'єктивний характер і проводиться в результаті експертного опитування.

Для методів, заснованих на експертних оцінках, важливим є характер вимірювань: первинний, тобто безпосередній, або похідний, тобто опосередкований, а також тип шкали, якою користується експерт і яка потім визначає вид операцій з обробки експертної інформації. Інший особливістю можна назвати те, що оцінювані об'єкти володіють двома типами властивостей:

1) властивості, які можна безпосередньо виміряти, наприклад: довжину, висоту, зріст, вагу, обсяг і т.п.,

2) властивості, які не мають кількісної міри, а є якісними, наприклад: краса, зручність, корисність і т.п.

У другому випадку, як правило, вдаються до попарного порівняння об'єктів, що володіють якісними властивостями і визначають місце (ранг) по відношенню до розглядаємого поняття *A*.

Існуючі методи побудови функції приналежності можна розділити на групи: *прямі і непрямі*.

Прямі методи полягають в тому, що експерт безпосередньо задає степінь приналежності кожного $x \in X$ поняттю *A* чи визначає вид функції $\mu_A(x)$. При цьому значення функції узгоджуються з уподобаннями експерта по відношенню x_i на множині *X* наступним чином:

- для будь-яких x_1, x_2 $\in X$ $\mu_A(x_1) < \mu_A(x_2)$ тоді і тільки тоді, коли x_2 переважніше x_1 тобто в більшій мірі (з більшою ймовірністю, з більшою можливістю, з більшою необхідністю) характеризується поняттям *A;*

- для будь-яких x_1, x_2 $\in X$ $\mu_A(x_1) = \mu_A(x_1)$ тоді і тільки тоді, коли $x1$ і $x2$ байдужі щодо поняття *A*.

Як правило, прямі методи використовуються для опису об'єктів, властивості яких можна безпосередньо виміряти. У цьому випадку здійснюють безпосереднє завдання значень степені приналежності. До прямих методів можна віднести методи, засновані на ймовірнісному трактуванні функції приналежності $\mu_A(x) = p(x/A)$, тобто ймовірність того, що об'єкт $x \in X$ буде віднесений до нечіткої множини *A*.

У непрямих методах значення функції приналежності вибираються виходячі з заздалегідь сформульованих умов. Інформація, що надходить від експертів, є лише вихідною для реалізації встановленого в даному методі алгоритму обробки цієї інформації з метою отримання $\mu_A(x)$. Опосередковані методи є єдино можливими в тих випадках, коли властивості, через які визначається поняття *A*, не мають кількісної міри, наприклад, "краса". У таких випадках використовують рангові вимірювання при попарному порівнянні об'єктів. Непрямі методи більш трудомісткі, ніж прямі, але їхня перевага - в стійкості по відношенню до спотворень у відповіді, а отже, в більшій об'єктивності отримуваної функції приналежності.

З метою підвищення об'єктивності функції приналежності для її побудови враховують думку групи експертів.

Практичне використання теорії нечітких множин передбачає наявність функцій приналежності, якими описуються лінгвістичні терми, наприклад, «низький», «середній», «високий» і т.п. Задача побудови функцій приналежності ставиться наступним чином: дані дві множини: множина термів $L=\{l_1, \ldots, l_m\}$ і універсальна множина $U=\{u_1, \ldots, u_n\}$. Нечітка множина

$\tilde{l}_j$, якою описується лінгвістичний терм l_j, $j=\overline{1,m}$, на універсальній множині U представляється у виді: $\tilde{l}_j=\left(\frac{\mu_{l_j}(u_1)}{u_1},\frac{\mu_{l_j}(u_2)}{u_2},\ldots,\frac{\mu_{l_j}(u_n)}{u_n}\right)$. Необхідно визначити степені приналежності елементів множини U до елементів з множини L, тобто знайти $\mu_{l_j}(u_i)$ для всіх $j=\overline{1,m}$ і $i=\overline{1,n}$.

Далі розглядаються два методи побудови функцій приналежності. Перший метод заснований на статистичній обробці думки групи експертів. Другий метод базується на парних порівняннях, які виконуються одним експертом.

10.5.1. Побудова функцій приналежності на основі експертної інформації. Метод статистичної обробки експертної інформації

Кожен експерт заповнює опитувальник, в якому вказує свою думку про наявність в елементів властивостей нечіткої множини. Опитувальник має наступний вид:

	u_1	u_2		u_n
$\tilde{l}_1$				
$\tilde{l}_2$				
$\tilde{l}_m$				

Введемо наступні позначення: К – кількість експертів; $b_{j,i}^k$ - думка k-го експерта про наявність в елемента u_i властивостей нечіткої множини $\tilde{l}_j$, $k=\overline{1,K}$, $i=\overline{1,n}$, $j=\overline{1,m}$. Будемо вважати, що експертні оцінки бінарні, тобто: $b_{j,i}^k\in\{0,1\}$, де 1(0) вказує на наявність(відсутність) в елемента u_i властивостей нечіткої множини $\tilde{l}_j$. За результатами опитування експертів, степені приналежності нечіткій множині $\tilde{l}_j$ ($j=\overline{1,m}$) розраховуються наступним чином:

$$\mu_{l_j}(u_i)=\frac{1}{K}\sum_{k=\overline{1,K}} b_{j,i}^k,\ i=\overline{1,n} \qquad (10.1)$$

Приклад 10.3. Побудувати функції приналежності термів «низький», «середній», «високий», які використовуються для лінгвістичної оцінки змінної «ріст чоловіка». Результати опитування п’яти експертів наведені в табл. 10.1.

Таблиця 10.1 – Результати опитування експертів

k	терми	[160, 165)	[165, 170)	[170, 175)	[175, 180)	[180, 185)	[185, 190)	[190, 195)	[195, 200)
Експерт 1	низький	1	1	1	0	0	0	0	0
	середній	0	0	1	1	1	0	0	0
	високий	0	0	0	0	0	1	1	1
Експерт 2	низький	1	1	1	0	0	0	0	0
	середній	0	0	1	1	0	0	0	0
	високий	0	0	0	0	1	1	1	1
Експерт 3	низький	1	0	0	0	0	0	0	0
	середній	0	1	1	1	1	1	0	0
	високий	0	0	0	0	0	1	1	1
Експерт 4	низький	1	1	1	0	0	0	0	0
	середній	0	0	0	1	1	1	0	0
	високий	0	0	0	0	0	0	1	1
Експерт 5	низький	1	1	0	0	0	0	0	0
	середній	0	1	1	1	0	0	0	0
	високий	0	0	0	1	1	1	1	1

Результати обробки експертних міркувань представлені в таблиці 10.2. Для кожного терма маємо два рядки - це кількість голосів, відданих експертами за приналежність нечіткій множині відповідного елемента універсальної множини, та - степені приналежності, розраховані за формулою (10.1). Графіки функцій приналежності наведені на рис. 10.2.

Таблица 10.2 - Результати обробки думок експертів

терми	розрахунки	[160, 165)	[165, 170)	[170, 175)	[175, 180)	[180, 185)	[185, 190)	[190, 195)	[195, 200)
низький	*голоси*	5	4	3	0	0	0	0	0
	степінь	1	0.86	0.6	0	0	0	0	0
середній	*голоси*	0	2	4	5	3	2	0	0
	степінь	0	0.4	0.8	1	0.6	0.4	0	0
високий	*голоси*	0	0	0	1	2	4	5	5
	степінь	0	0	0	0.2	0.4	0.8	1	1

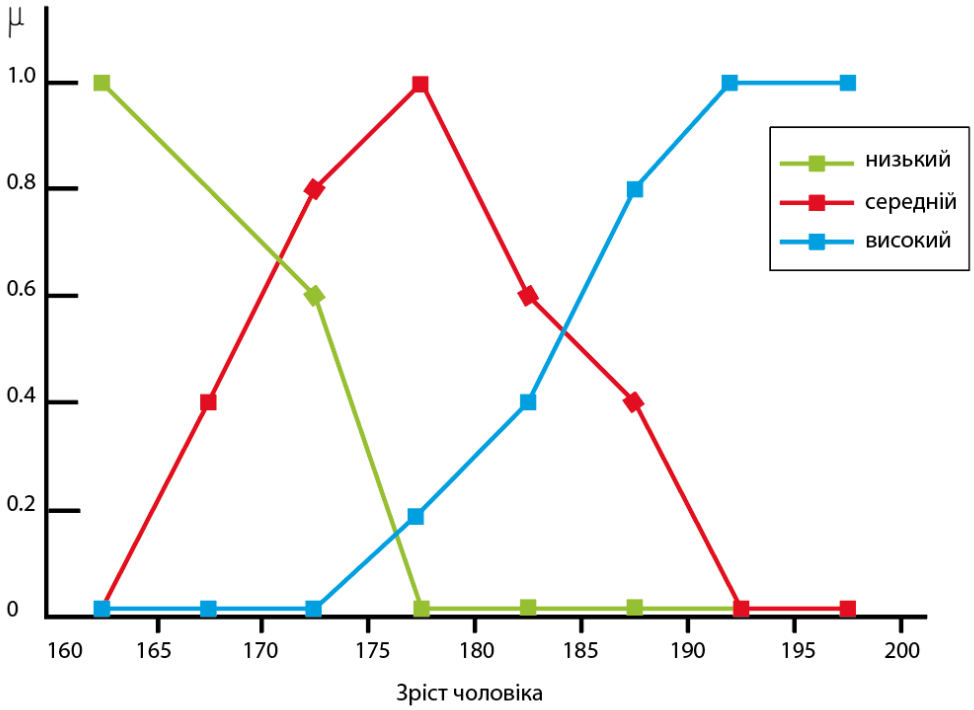


Рисунок 10.2 - Функції приналежності нечітких множин з прикладу 10.3

10.5.2. Побудова функцій приналежності на основі парних порівнянь

Вихідною інформацією для побудови функцій приналежності є експертні парні порівняння. Для кожної пари елементів універсальної множини експерт оцінює перевагу одного елемента над іншим по відношенню до властивості нечіткої множини. Парні порівняння зручно представляти наступній матрицею:

$$A = \begin{array}{c} \\ u_1 \\ u_2 \\ \dots \\ u_n \end{array} \begin{array}{c} \begin{array}{cccc} u_1 & u_2 & \dots & u_n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \end{array}$$

де a_{ij} рівень переваги елемента u_i над u_j (i,j=$\overline{1,n}$), який визначається за дев'ятибальною шкалою Сааті:

1 - якщо відсутня перевага елемента u_i над елементом u_j ;

3 - якщо є слабка перевага u_i над u_j;

5 - якщо є суттєва перевага u_i над u_j;

7 - якщо є явна перевага u_i над u_j;

9 - якщо є абсолютна перевага u_i над u_j;

2,4,6,8 - проміжні порівняльні оцінки.

Матриця парних порівнянь є діагональної ($a_{ij} = 1$,i=$\overline{1,n}$) і обернено симетричною ($a_{ij} = \frac{1}{a_{ij}}$, ij =$\overline{1,n}$).

Степені приналежності приймаються рівними відповідним координатам власного вектора W= (w_1, w_2 ,.. w_n) матриці парних порівнянь:

$\mu(u_i)= w_i$ i=$\overline{1,n}$ (10.2)

Власний вектор знаходиться з наступної системи рівнянь:

$$\begin{cases} A * W = \lambda max * W \\ w_1 + w_2 + \cdots + w_n = 1 \end{cases} \quad (10.3)$$

де λmax- максимальне власне значення матриці А.

Приклад 10.4. Побудувати функцію приналежності нечіткої множини "високий чоловік" на універсальній множині {170, 175, 180, 185, 190, 195}.

Парне порівняння задамо наступною матрицею:

A=	170	175	180	185	190	195
170	1	1/2	1/4	1/6	1/8	1/9
175	2	1	1/3	1/5	1/7	1/8
180	4	3	1	1/4	1/4	1/5
185	6	5	4	1	1/3	1/3
190	8	7	4	3	1	1
195	9	8	5	3	1	1

Власні значення цієї матриці парних порівнянь наступні:

6.2494;
0.0318 + 1.2230i;
0.318 - 1.2230i;
-0.1567 + 0.2392i;
-0.1567 - 0.2392i;
0.0004.

Виходить, що λ_{max}=6.2494. Степені приналежності, знайдені за формулами (10.3) і (10.2), наведені в табл.10.3. Нечітка множина вийшла субнормальною. Для нормалізації поділимо всі степені приналежності на максимальне значення, тобто на 0.3494. Графіки функцій приналежності субнормальної і нормальної нечіткої множини "високий чоловік" наведені на рис. 10.3.

Таблиця 10.3. Функції приналежності нечіткої множини «високий чоловік» до прикладу 10.4.

	170	**175**	**180**	**185**	**190**	**195**
(субнормальна нечітка множина)	0.0284	0.0399	0.0816	0.1754	0.3254	0.3494
(нормальна нечітка множина)	0.0813	0.1141	0.2335	0.5021	0.9314	1.0000

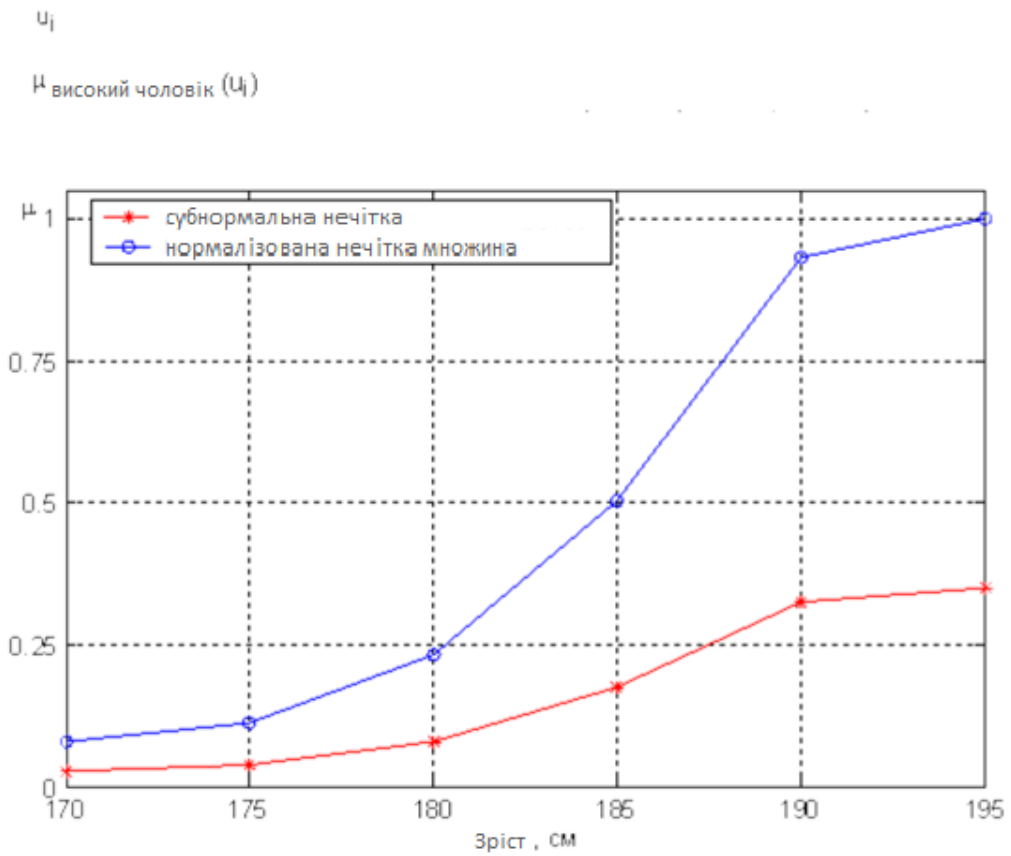


Рисунок 10.3. Функції приналежності нечіткої множини "високий чоловік" до прикладу 10.4.

Відхилення λ_{max} від n може слугувати мірою неузгодженості парних порівнянь експеременту. У прикладі 10.4 λ_{max}=6.2494, а n=6. Отже, міра неузгодженості дорівнює 0.2494. При узгоджених парних порівняннях процедура побудови функцій приналежності значно спрощується.

З узгодженими думками експерта матриця парних порівнянь має наступні властивості:

- вона діагональна, тобто $a_{ii}=1$, i=1..n;
- вона обернено симетрична, тобто елементи, які симетричні відносно головної диагоналі, пов'язані залежностю $a_{ij}=1/a_{ji}$, i,j=1..n;
- вона транзитивна, тобто $a_{ik}a_{kj}=a_{ij}$, i,j,k=1..n.

Наявність цих властивостей дозволяє визначити всі елементи матриці парних порівнянь, якщо відомі (n-1) недіагональних елементів. Наприклад, якщо відомий k-тий рядок, тобто елементи a_{kj}, i,j=1..n, то довільний елемент a_{ij} визначається так:

(10.4)

$$a_{ij} = \frac{a_{kj}}{a_{ki}} \qquad i,j,k = \overline{1,n}.$$

Після визначення всіх елементів матриці парних порівнянь степені приналежності нечіткої множини обчислюються за формулою:

$$\mu(u_i) = \frac{1}{a_{1i} + a_{2i} + \ldots + a_{ni}} \qquad (10.5)$$

Формула (10.5), у порівнянні з формулами (10.2) та (10.3) не потребує виконання складних обчислювальних процедур, пов’язаних із пошуком власного вектору матриці А.

Приклад 10.5. Побудувати функцію приналежності нечіткої множини "високий чоловік" на універсальній множині {170, 175, 180, 185, 190, 195}, якщо відомі такі експертні парні порівняння:

- *абсолютна перевага 195 над 170;*
- *явна перевага 195 над 175;*
- *значна перевага 195 над 180;*
- *слабка перевага 195 над 185;*
- *відсутність переваги 195 над 190.*

Приведеним експертним висловлюванням відповідає така матриця парних порівнянь:

$$A = \begin{array}{c} \\ 170 \\ 175 \\ 180 \\ 185 \\ 190 \\ 195 \end{array} \begin{array}{c} \begin{array}{cccccc} 170 & 175 & 180 & 185 & 190 & 195 \end{array} \\ \begin{bmatrix} 1 & 7/9 & 5/9 & 1/3 & 1/9 & 1/9 \\ 9/7 & 1 & 5/7 & 3/7 & 1/7 & 1/7 \\ 9/5 & 7/5 & 1 & 3/5 & 1/5 & 1/5 \\ 3 & 7/3 & 5/3 & 1 & 1/3 & 1/3 \\ 9 & 7 & 5 & 3 & 1 & 1 \\ \mathbf{9} & \mathbf{7} & \mathbf{5} & \mathbf{3} & \mathbf{1} & 1 \end{bmatrix} \end{array}$$

Напівжирним шрифтом виокремлені елементи, відповідні парним порівнянням з умов прикладу. Решта елементів знайдені за формулами (10.4). Використовуючи формули (10.5) знаходимо степені приналежності (табл. 10.4). Для нормалізації нечіткої множини поділимо всі степені приналежності на максимальне значення, тобто на 0.3588. Графіки функцій приналежності субнормальної та нормальної нечіткої множини "високий чоловік" наведені на рис. 10.4.

Таблиця 10.4. Функції приналежності нечеткої множини "високий чоловік" до прикладу 10.5

	170	**175**	**180**	**185**	**190**	**195**
μ високий чоловік (u_i) (субнормальна нечітка множина)	0.0399	0.0513	0.0718	0.1196	0.3588	0.3588
μ високий чоловік (u_i) (нормальна нечітка множина)	0.1111	0.1429	0.2000	0.3333	1.0000	1.0000

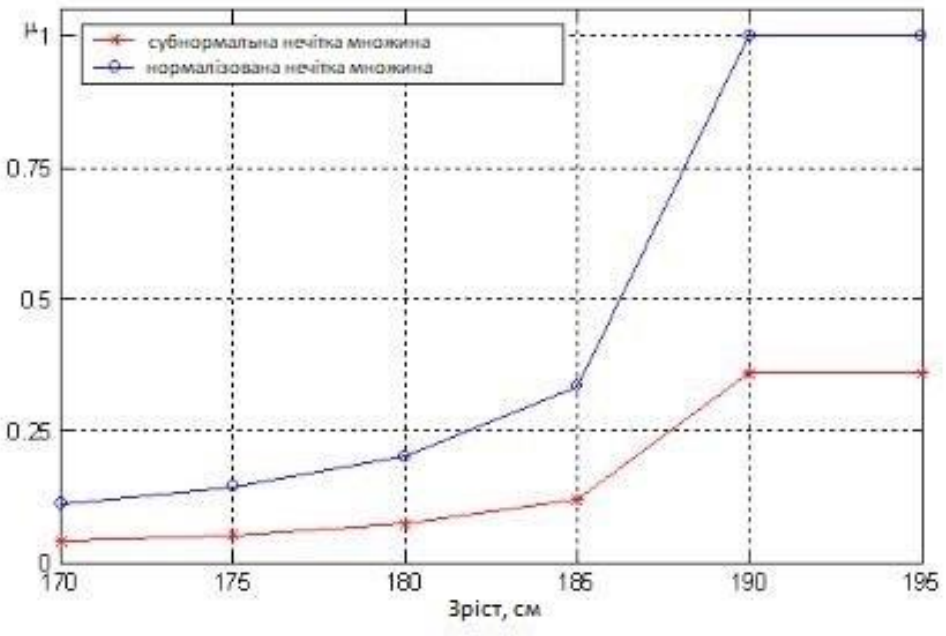


Рисунок 10.4. Функції приналежності нечеткої множини "високий чоловік" до прикладу 10.5

10.6. Нечітке логічне виведення

Одним з можливих способів прийняття рішень при невизначеності інформації є застосування нечіткого логічного висновку.

Нечітким логічним висновком називається отримання висновку у вигляді нечіткої множини, відповідного поточним значеннях входів, з використанням нечіткої бази знань і нечітких операцій: $\boldsymbol{Bt = At \circ (A \longrightarrow B)}$

Основу нечіткого логічного виведення становить композиційне правило Заде та нечітка імплікація.

Використовуючи нечітку імплікацію $\boldsymbol{A \longrightarrow B}$ отримаємо відношення на $\boldsymbol{A \times B}$: $\mu R_{A\times B}= \{\mu_{<a1,b1>}; \mu_{<a1,b2>}; ..., \mu_{<an,bm>}\}$, де кожне значення отримано як min за правилом Мандані (існують і інші схеми: Цукамото, Ларсена): $\mu_{<ai;bj>}=\min(\mu_a, \mu_{bj})$.

Композиційне правило виведення Заде формулюється так: якщо відомо нечітке відношення між вхідною (x) і вихідною (y) змінними, то при нечіткому значенні вхідної змінної x = Ã, нечітке значення вихідної змінної визначається наступним чином: y = Ã ∘ R, де - максиміна композиція[iii].

Формально це можна записати у такій формі:

якщо x є A, то y є B, інакше y є C

x є A '

---

y є D

Запропоновано декілька правил, які переводять нечітке умовне висловлювання «якщо x є A, то y є B, інакше y є C» в нечітке відношення U • V.

Нехай x, y - імена об'єктів; A, A ', B, C, D - нечіткі поняття, представлені нечіткими множинами, визначеними на U, U, V, V, V відповідно.

Тут A, B, C - нечіткі множини в U, V, V, задані у вигляді

A = ∫ μ A (u) / u; B = ∫ μB (v) / v; C = ∫ μC (v) / v.

Тоді маємо:

А. Максимине правило Rm ':

Rm '= (A × B) ∪ (~ A × C) = ∫ **(μA (u) ∧ μB (v)) ∨ ((1 - μA (u)) ∧ μC (v))** /(u,v).

Б. Арифметичне правило Ra ' (Лукасевича) :

Ra '= (~ A × V + U × B) ∩ (A × V + U × C) =

= ∫ **1 ∧ (1 - μA (u) + μB (v)) ∧ (μA (u) + μC (v))** / (u, v).

В. Розмите бінарне правило:

Rb '= (~ A × V ∪ U × B) ∩ (A × V ∪ U × C) =

= ∫ **((1 - μA (u)) ∨ μB (v)) ∧ ((μA (u) ∨ μC (v))** / (u, v).

Г. Правило Танака-Мідзумото (Геделева логічна система):

Rgg '= (A × V ⇒ U × B) ∩ (~ A × V ⇒ U × C) =

= ∫ **(μA (u) → μB (v)) ∧ ((1 - μ A (u))→ μC (v))** / (u, v)

де $\mu_A(u) \to \mu_B(v) = \begin{cases} 1, \text{ при } \mu_A \le \mu_B \\ \mu_B, \text{ при } \mu_A > \mu_B \end{cases}$

Таким чином, повертаючись до початкової постановці завдання наслідок D можна отримати наступним чином:

Dm = A '• Rm ',

Da = A '• Ra ',

Db = A '• Rb ',

Dgg = A '• Rgg '.

Питання до розділу 10

1. Способи формалізації нечіткостей.
2. Нечітка множина: визначення, зміст, функція приналежності, відміна її від ймовірності.

3. Основні поняття теорії нечітких множин: α-рівень, носії множини, ядро, найближча чітка множина.

4. Нечіткі множини як узагальнення канторовської теорії множин.

5. Операції над нечіткими множинами (перетин, об'єднання, доповнення, включення, порожня множина і універсум, розтягнення, концентрація).

6. Які тотожності алгебри множин виконуються для нечітких підмножин, а які ні?

7. Відстань Хемінга і Евкліда. Відносна відстань.

8. Нечіткі відношення. Навести приклад. Властивості нечітких відношень (рефлексивність, симетричність, транзитивність).

9. Max-min композиція нечітких відношень.

10. Відношення схожості і відмінності - якими властивостями володіють, як співвідносяться між собою?

12. Лінгвістична змінна. Терм-множина. Який об'єкт в лінгвістичної змінної моделюється нечіткими множинами?

16. Побудова функції приналежності методом статистичного опитування експертів.

17. Побудова функції приналежності методом узгодженого попарного порівняння Сааті.

18. Нечіткі правила виведення. Нечітка імплікація.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Базова література

1. Таран Т. А. Основы дискретной математики. Учебное пособие. К.: Просвіта. – 1998. – с. 148.
2. Темнікова О.Л. Математична логіка. Практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 113 «Прикладна математика», освітньої програми «Наука про дані та математичне моделювання»; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1,37 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 76 с.
3. Нікольський Ю.В., Пасічник В.В., Щербина Ю.М. Системи штучного інтелекту : Навчальний посібник. За ред. В.В.Пасічника. – 2-ге вид., випр. та доп. - Львів : Магнолія -2006, 2013
4. Новиков Ф.А. Дискретная математика для программистов. 2ое издание. Учебник для ВУЗов. СПб.: Питер. – 2006. – с.368.
5. Мендельсон Э. Введение в математическую логику. –- М.: Наука, 1976.

Допоміжна література

1. Bird R. Introduction to Functional Programming using Haskell. 2-nd edition, Prentice Hall Press, 1998.
2. Fokker J. Functional programming. Dept. of CS. Utrecht University, 1995.
3. Henson M. Elements of functional languages. Dept. of CS. University of Sassex, 1990..
4. Thompson S. Haskell: The Craft of Functional Programming. 2-nd edition, Addison-Wesley, 1999..
5. Бардачов Ю.М., Соколова Н.А., Ходаков В.Є. Дискретна математика. – К.: Вища школа. 2002. 287с.
6. Бердж В. Методи рекурсивного програмування. М .: Машинобудування, 1983.
7. Гильберт Д., Бернайс П. Основания математики. Логические исчисления и формализация арифметики. –- М.: Наука. 1979.
8. Гиндикин С. Г. Алгебра логики в задачах. – М.: Наука. 1972.
9. Глушков В.М. Введение в кибернетику.–Киев: Изд-во АНУССР. 1964.
10. Джонс С., Лестер Д. Реалізація функціональних мов. М .: Мир, 1991.
11. Ершов Ю.А., Палютин Е.А. Математическая логика.–М.: Наука, 1979.
12. Клини С. К. Введение в метаматематику. М.: Мир. 1957.
13. Клини С. К. Математическая логика. - М.: Мир, 1973.
14.Кофман А. Введение в теорию нечетких множеств. – М.: Радио и связь. 1982.
15. Кузнецов О.П. Дискретная математика для инженера. Учебное пособие. 6-е изд., стер. — СПб.: Издательство «Лань», 2009. — 400 с.

16. Кэррол Л. История с узелками. – М.: Мир. 1973.
17. Лавров И. А., Максимова Л. Л. Задачи по теории множеств, математической логике и теории алгоритмов. – М. : Наука, 1975.
18. Мальцев А.И. Алгебраические системи. - М.: Наука. 1975.
19. Нагель Э., Ньюмен Д. Теорема Геделя. –-М.: Знание. 1970.
20. Новиков П.С. Элементы математической логики.–-М.: Физматгиз, 1959.
21. Робертс Ф. С. Дискретные математические модели с приложениями к социальным, биологическим и экологическим задачам. – М.: Наука, 1986.
22. Столл Р. Множество, логика, аксиоматические теории. – М.: Просвещение, 1968.
23. Таран Т.А., Мыценко Н.А., Темникова Е.Л. Сборник задач по дискретной математике. 2ое издание. К.: Инрес. – 2005. – с. 64.
24. Філд А., Харрісон П. Функціональне програмування. М .: Мир, 1993.
25. Ханнес Хювенен Е., Сеппенен І. Світ Lisp'a. У 2-х томах. М .: Світ, 1990..
26. Хендерсон П. Функціональне програмування. Застосування та реалізація. М .: Мир, 1983.
27. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. – М.: Наука. 1983.
28. Черч А. Введение в математическую логику - М.: ИЛ. т.1. 1961.
29. Яблонский С.В. Введение в дискретную математику.–- М.: Наука. 1979.

[i] ***ДОВІДКОВИЙ МАТЕРІАЛ***

Мови функціонального програмування

Наведено короткий опис деяких мов функціонального програмування (далеко не всіх):

1. **Ліспи** (List processor). Вважається першим функціональним мовою програмування. Нетипізовані. Містить масу імперативних властивостей, проте в загальному заохочує саме функціональний стиль програмування. При обчисленнях використовує виклик-по-значенням. Існує об'єктно-орієнтована діалект мови - CLOS.
2. **ISWIM** (If you See What I Mean). Функціональний мова-прототип. Розроблено Пітером Ландін в 60-х роках XX століття для демонстрації того, яким може бути мова функціонального програмування. Разом з мовою П. Ландін розробив і спеціальну віртуальну машину для виконання програм на ISWIM'е. Ця віртуальна машина, заснована на виклик-по-значенням, отримала назву SECD-машини. На синтаксисі мови ISWIM базується синтаксис багатьох функціональних мов. На синтаксис ISWIM схожий синтаксис ML, особливо Caml.
3. **Scheme** . Діалект Lisp'a, призначений для наукових досліджень в області computer science. При розробці Scheme був зроблений упор на елегантність і простоту мови. Завдяки цьому мова вийшла набагато менше, ніж Common Lisp.
4. **ML** (Meta Language). Сімейство строгих мов з розвиненою полиморфной системою типів і параметрізуемих модулями. ML викладається у багатьох західних університетах (в деяких навіть як перша мова програмування).
5. **Standard ML** . Один з перших типізованих мов функціонального програмування. Містить деякі імперативні властивості, такі як посилання на змінні значення і тому не є чистим. При обчисленнях використовує виклик-по-значенням. Дуже цікава реалізація модульности. Потужна поліморфна система типів. Останній стандарт мови - Standard ML-97, для якого існує формальні математичні визначення синтаксису, а також статичної та динамічної семантик мови.
6. **Caml Light** і **Objective Caml** . Як і Standard ML належить до сімейства ML. Objective Caml відрізняється від Caml Light в основному підтримкою класичного об'єктно-орієнтованого програмування. Також як і Standard ML строгий, але має деяку вбудовану підтримку відкладених обчислень.

7. **Miranda** . Розроблено Девідом Тернером, в якості стандартного функціонального мови, який використав відкладені обчислення. Має строгу полиморфную систему типів. Як і ML викладається у багатьох університетах. Зробив великий вплив на розробників мови Haskell.

8. **Haskell** . Один з найпоширеніших нестрогих мов. Має дуже розвинену систему типізації. Дещо гірше розроблена система модулів. Останній стандарт мови - Haskell-98.

9. **Gofer** (GOod For Equational Reasoning). Спрощений діалект Haskell'a. Призначений для навчання функціональному програмуванню.

10. **Clean** . Спеціально призначений для паралельного і розподіленого програмування. За синтаксису нагадує Haskell. Чистий. Використовує відкладені обчислення. З компілятором поставляється набір бібліотек (I / O libraries), що дозволяють програмувати графічний користувальницький інтерфейс під Win32 або MacOS.

[ii] **Інтуїціонізм** - сукупність філософських і математичних поглядів, що розглядають математичні судження з позицій інтуїтивної переконливості. Розрізняються два трактування інтуїціонізму: інтуїтивна переконливість, яка пов'язана з питанням існування об'єктів, і наочна розумова переконливість.

В інтуїціоністській математиці відкидається підхід класичної теорії множин (зокрема, не приймаються аксіома вибору та аксіома регулярності) та ряд міркувань класичної логіки.

В інтуїціоністській математиці судження вважається істинним, тільки якщо його можна довести деяким «уявним експериментом». Тобто істинність твердження «Існує об'єкт *x*, для якого вірно судження *A(x)*» доводиться побудовою такого об'єкта, а істинність твердження «*A* або *B*» доводиться або доказом істинності твердження *A*, або доказом істинності твердження *B*. Звідси, зокрема, випливає, що твердження «*A* чи не *A*» може бути не істинним, а закон виключеного третього є неприйнятним. Істинним математичним судженням є ряд виконаних побудов ефективного характеру з використанням інтуїціоністської логіки. Ефективність не обов'язково пов'язана з наявністю алгоритму та може залежати від фізичних та історичних факторів, фактичного вирішення проблем.

Основними об'єктами дослідження інтуїціоністської математики є конструктивні об'єкти: натуральні і раціональні числа, кінцеві множини конструктивних об'єктів зі списком елементів, послідовності, що вільно стають (послідовності вибору, кожен член яких може бути ефективно доступний), інтуїціоністські види (властивості, якими можуть обладати об’єкти дослідження). Послідовності, що вільно стають, розрізняють в залежності від ступеня інформації, відомої досліднику. Якщо закон

формування послідовності відомий повністю, її називають заданою законом, якщо відомий лише початковий відрізок — беззаконною.

У трактуванні теорії множин не робиться різниця між абстрактними об'єктами і об'єктами, існування яких можна підтвердити побудовою. У класичній математиці на нескінченні множини екстраполювали властивості та закони кінцевих сукупностей. При цьому немає способу ефективної побудови об'єктів, що знаходить своє відображення в так званих «теоремах чистого існування».

Критика теорії множин призвела до виникнення двох течій: інтуїціонізму Брауера та формалізму Гільберта. У 1904 році Брауер розкритикував низку концепцій класичної математики. Його увагу привернув статус існування: чи можна потенційно побудувати такі об'єкти дослідження, як незмірна множина дійсних чисел, як ніде не диференційована функція? Чи можна вважати, що в навколишньому світі існують нескінченні множини об'єктів?

Інтуїціоністська математика в трактуванні Брауера — це переконливість уявних побудов, не пов'язана з питанням існування об'єктів. Інше трактування - це «наочна розумова переконливість найпростіших конструктивних процесів реальної дійсності». Брауер заперечував проти формалізації інтуїціонізму.

Аренд Гейтінг сформулював інтуїціоністське обчислення предикатів та інтуїціоністське арифметичне обчислення, Альфредом Тарським було відкрито топологічну інтерпретацію, а Андрієм Миколайовичем Колмогоровим — інтерпретацію у вигляді обчислення задач.

[iii] Оперція max-min – композиція над відношеннями $R_2 \circ R_1$ ($R_1 \subset X \times Y$ та $R_2 \subset Y \times Z$; результат $R_2 \circ R_1 = R_2(R_1) \subset X \times Z$):

$\mathbf{\mu_{R2 \cdot R1}(x, z) = \max_y [\min(\mu_{R1}(x, y), \mu_{R2}(y, z))]}$, де $x \in X$, $y \in Y$, $z \in Z$.