МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

**М. С. Нікітченко, Т. В. Панченко,
С. А. Поляков**

ТЕОРІЯ ПРОГРАМУВАННЯ В ПРИКЛАДАХ І ЗАДАЧАХ

Навчальний посібник

КИЇВСЬКИЙ УНІВЕРСИТЕТ 2015

УДК 519.8(075.8)
ББК 22.18я73
Н62

Рецензенти:
д-р фіз-мат. наук, проф. Ю. А. Бєлов,
д-р фіз-мат. наук, проф. А. Ю. Дорошенко

Рекомендовано до друку
вченою радою факультету кібернетики
(протокол № 2 від 28 жовтня 2013 року)

Ухвалено науково-методичною радою
Київського національного університету імені Тараса Шевченко
29 листопада 2013 року

Нікітченко М. С.

Н62 Теорія програмування в прикладах і задачах : навч. посіб. / М. С. Нікітченко, Т. В. Панченко, С. А. Поляков. – К. : ВПЦ "Київський університет", 2015. – 191 с.

Розглянуто формалізацію простої мови програмування, формальні мови та граматики, операційну та аксіоматичну семантику мов, рекурсію в мовах. Зазначені питання проілюстровано задачами з розв'язаннями, коментарями та поясненнями.

Для студентів спеціальності "Інформатика".

УДК 519.8(075.8)
ББК 22.18я73

ЗМІСТ

ПЕРЕДМОВА

Інформаційні технології застосовуються нині практично в усіх сферах життя людства. Вони докорінно змінили світ. Значні успіхи в телекомунікаціях, енергетиці, транспорті, медицині, освіті, космічних дослідженнях та інших галузях діяльності людства досягнуто завдяки використанню інформаційних технологій. Однак розширення арени їх застосування має також негативні наслідки. Людство стає все більш залежним від надійного та правильного функціонування комп'ютерних систем; помилки в їх роботі часом зумовлюють величезні фінансові й навіть людські втрати. Дотепер фірми-розробники програмного забезпечення не дають гарантію на програмні продукти, які вони пропонують користувачу, чого фактично немає в жодній іншій галузі. Будь-який легальний продавець товару гарантує якість і готовий (пригадаємо повернення мільйонів автомобілів на заводи виробника в разі виявлення дефектів) фінансово та юридично відповідати за поставку недоброякісного продукту.

Можна навести безліч прикладів, коли неякісне програмне забезпечення призводило до знищення космічних ракет, аварій на заводах, гідроелектростанціях тощо. Про складний стан справ говорять і самі фахівці з комп'ютерних технологій. Причини цього, безумовно, різноманітні й пов'язані з науковими, економічними, соціальними, психологічними та іншими проблемами використання інформаційних технологій. Тому однією з найважливіших є проблема створення ефективних методів розробки програмних систем. Такі методи повинні базуватися на розвиненій теорії програмування, що має на меті дослідження програм і методів їх аналізу та синтезу. Зазначимо, що тут термін "програма" тлумачиться в широкому сенсі, охоплюючи, зокрема, як програми конкретних мов програмування, так і великі програмні системи.

Даний посібник можна розглядати як вступ до теорії програмування, що є базовою нормативною дисципліною спеціальності "Інформатика".

Щоб отримати початкове уявлення про дисципліну "Теорія програмування", треба визначити її мету та завдання, об'єкт і предмет, а також розглянути її основні методи.

Метою й завданням навчальної дисципліни "Теорія програмування" є засвоєння основних концепцій, принципів і понять сучасного, зокрема композиційного, програмування. У світоглядному аспекті поняття й методи теорії програмування необхідні для обґрунтування та формалізації способів розробки правильних, ефективних програм. У прикладному аспекті апарат теорії програмування необхідний для адекватного моделювання мов специфікацій, програмування й використання побудованих моделей для створення сучасних програмних та інформаційних систем високої якості.

Об'єктом навчальної дисципліни "Теорія програмування" є програми, а її **предмет** включає вивчення основних (базових) понять теорії програмування, розгляд основних аспектів програм (семантика та синтаксис), їх формалізацію й дослідження, а також уточнення основних методів розробки програм.

Вимоги до знань і вмінь. Для засвоєння матеріалу необхідні знання основ елементарної та дискретної математики, алгебри, математичної логіки й теорії алгоритмів. При вивченні дисципліни використовуватимуться **методи** теорії множин, дискретної математики, універсальної алгебри, математичної логіки й теорії алгоритмів.

Місце у структурно-логічній схемі спеціальності. Нормативна навчальна дисципліна "Теорія програмування" є складовою циклу професійної підготовки фахівців освітньо-кваліфікаційного рівня "бакалавр". Курс теорії програмування потрібен для подальшого вивчення нормативних дисциплін "Системне програмування", "Бази даних та інформаційні системи", "Інтелектуальні системи", "Теорія обчислень", "Штучний інтелект", "Формальні методи розробки програмних систем", "Інформаційні технології", "Прикладна логіка", низки спецкурсів відповідного напряму.

Суть теорії програмування – категоріальний (поняттєвий) аналіз процесу програмування. Поняття визначаються в єдності їх двох моментів: змісту (інтенсіоналу) та обсягу (екстенсіоналу). Через це так багато часу приділяється аналізу основних понять програмування й розкриттю їх інтенсіоналів та екстенсіоналів. Отже, для теорії програмування найважливішим є не конкретна програма, а відношення між програмними поняттями (тобто не стільки предметна, екстенсіональна, скільки логічна, інтенсіональна складова). Тому вміння писати конкретні програми ще не означає розуміння теорії програмування. Тут простежується аналогія з політикою, оскільки задача політики полягає не стільки в безпосередній роботі з пересічною людиною, скільки у вибудовуванні суспільних відносин.

Структура дисципліни:

- формалізація простої мови програмування;
- основні програмні поняття;
- синтактика;
- семантика;
- методи розробки програм.

У першій частині посібника розглядаються основні програмні поняття, формалізується проста мова програмування, вводиться поняття програмної системи, вивчаються формальні мови та граматики, методи уточнення рекурсії програм.

Назва "Теорія програмування" складається з двох термінів – "теорія" та "програмування". Існують різні підходи до побудови теорій. Щоб зрозуміти особливості теорії програмування, спробуємо відповісти на кілька загальних запитань щодо властивостей теорій.

Які значення та роль теорії?

Яка теорія потрібна (за побудовою)?

Яким методом будувати теорію?

Відповідаючи на перше запитання, зазначимо, що теорію слід розглядати в нерозривному зв'язку з практикою. Дійсно, розробка теорії ґрунтується на науковому дослідженні та аналізі практики. Потім наукові теорії втілюються в практику за допомогою відповідних технологій. Таким чином, маємо цикл: практика – наукове дослідження – теорія – технологія – практика.

Відповідаючи на друге запитання, зазначимо, що існують різні підходи до побудови теорій: історичний, логічний, еволюційний тощо. Зосередимось на розгляді феноменологічного й сутнісного підходів. Феноменологічний підхід має на меті описання зовнішніх виявів певних процесів, а сутнісний – їх внутрішніх причин. Хоча ці підходи мають багато спільного, основну увагу приділимо сутнісному підходу.

Третє запитання пов'язане з вибором методу побудови теорії. Зважаючи на те, що основні методи побудови програм (систематичний, зверху-вниз, структурний тощо) орієнтовані на поступове уточнення програм, будемо використовувати метод побудови теорії від абстрактного до конкретного (від простого до складного).

Основні принципи побудови теорії програмування:

- принцип єдності теорії та практики,
- принцип побудови сутнісної теорії,
- принцип побудови теорії методом від абстрактного до конкретного.

Автори висловлюють подяку Гришко Ю. В., Криволапу А. В., колегам і студентам факультету кібернетики за їх корисні коментарі та поради.

РОЗДІЛ 1
СИНТАКСИС І СЕМАНТИКА. МОВА SIPL. КОМПОЗИЦІЙНА СЕМАНТИКА

Одним з основних програмних понять є поняття імені (знака). Питаннями функціонування знаків займається також *семіотика* – наука про знаки та знакові системи. Тому не дивно, що здобутки семіотики застосовують і для означення програмних аспектів.

Основні семіотичні аспекти (прагматика, семантика й синтаксис) були визначені та досліджені в роботах Ч. Пірса, Ч. Морріса, Р. Карнапа та інших учених.

- Прагматика – це відношення між мовою та її користувачем.
- Семантика – відношення між виразами мови та їх значеннями (денотатами), абстраговане від користувача.
- Синтаксис – відношення між мовними виразами, абстраговане від їх денотатів.

Ці три аспекти також застосовують до програм, визначаючи таким чином їх прагматичний, семантичний і синтаксичний аспекти. Будемо називати ці аспекти семіотичними аспектами програм.

Семантика є провідним аспектом, а інші аспекти можуть розглядатися як похідні. Це дозволяє сформулювати такий принцип.

Принцип підпорядкованості. Семантика є провідним внутрішнім аспектом програм, а синтаксичний і денотаційний аспекти підпорядковані семантиці [7]. Усі аспекти варто спочатку вивчати незалежно один від одного, а потім – у їх єдності.

Унаслідок принципу підпорядкованості семантичний аспект є важливішим за синтаксичний. Отже, будемо використовувати семантико-синтаксичний підхід до дослідження й формалізації програмних понять. Сформулюємо це у вигляді принципу.

Принцип семантико-синтаксичної орієнтації. Основним підходом до дослідження й формалізації програмних понять є семантико-синтаксичний, а синтактико-семантичний підхід будемо розглядати як допоміжний.

Аналізуючи семантичний і синтаксичний аспекти, можна стверджувати, що для семантичного аспекту найважливішим є поняття композиції, тобто оператора над функціями, оскільки саме воно визначає властивості програм. Зазначимо також важливість відношень іменування (номінативність), а запропонований підхід назвемо *композиційно-номінативним.*

Більш розвинену систему аспектів програм, які називаються сутнісними аспектами, подано в посібнику з теорії програмування.[1]

1.1. Синтаксис мови SIPL

Для побудови дерева синтаксичного виведення, композиційного семантичного терму й доведення коректності програми будемо використовувати мову SIPL. Неформально мова SIPL має змінні цілого типу, над якими будуються арифметичні вирази та умови. Основними операторами є присвоювання, послідовне виконання, розгалуження, цикл.

Мова SIPL може розглядатися як надзвичайно спрощена традиційна мова програмування, наприклад Паскаль. У мові SIPL відсутні типи, оператори введення-виведення й багато інших конструкцій традиційних мов програмування. Разом з тим ця мова досить потужна для програмування різних арифметичних функцій, до того ж нею можуть бути запрограмовані всі обчислювальні функції над цілими числами.

Для опису синтаксису мов зазвичай використовують форми Бекуса – Наура (БНФ). Програми або їх частини виводять із метазмінних (нетерміналів), які записують у кутових дужках. Метазмінні задають синтаксичні класи. У процесі виведення метазмінні замінюються на праві частини правил, що задають ці метазмінні. Праві частини для однієї метазмінної розділяють знаком альтернативи "|". Процес породження припиняється, якщо всі метазмінні замінено на термінальні символи.

Синтаксис мови SIPL можна задати за допомогою такої БНФ (табл. 1.1):

[1] **Нікітченко М. С.** Теорія програмування / М. С. Нікітченко. – Ніжин, 2010. – Ч. 1.

Таблиця 1.1

БНФ мови SIPL

Ліва частина правила – метазмінна (дефінієндум)	Права частина правила (дефінієнс)	Ім'я правила
<програма> ::=	begin <оператор> end	*NP*1
<оператор> ::=	<змінна>:=<вираз> | <оператор> ; <оператор>| if <умова> then <оператор> else <оператор> | while <умова> do <оператор> | begin <оператор> end | skip	*NS*1 *NS*2 *NS*3 *NS*4 *NS*5 *NS*6
<вираз> ::=	<число> | <змінна> | <вираз> + <вираз> | <вираз> – <вираз> | <вираз> * <вираз> | <вираз> ÷ <вираз> | (<вираз>)	*NA*1– *NA*7
<умова> ::=	true | false | <вираз> < <вираз> | <вираз> ≤ <вираз> | <вираз> = <вираз> | <вираз> ≠ <вираз> | <вираз> > <вираз> | <вираз> ≥ <вираз> | <умова> ∨ <умова> | <умова> ∧ <умова> | ¬ <умова> | (<умова>)	*NB*1– – *NB*12
<змінна> ::=	*M* | *N* | . . .	*NV*1–…
<число> ::=	. . . –1 | 0 | 1 | 2 | 3 | . . .	*NN*1–…

Наведена БНФ задає мову SIPL як набір речень (слів), які виводяться з метазмінної <програма>. Щоб переконатись у синтаксичній правильності програми, треба побудувати її виведення, яке можна подати у вигляді дерева.

1.2. Композиційна семантика мови SIPL

Семантика задає значення (суть, зміст) програми, яке полягає в перетворенні вхідних даних на вихідні. У математиці такі перетворення називають функціями. Тому до семантичних понять відносять поняття даних, функцій і методів їх конструювання.

Такі методи називаються *композиціями*, а відповідна семантика – композиційною. Будемо вживати термін *композиційна семантика*, тому що саме композиції визначають її властивості. Композиційна семантика є певною конкретизацією функціональної семантики, оскільки базується на тлумаченні програм як функцій.

Дані

Базові типи даних – множини цілих чисел, булевих значень і змінних (імен):

- $Int = \{ \ldots, -2, -1, 0, 1, 2, \ldots \}$
- $Bool = \{true, false\}$
- $Var = \{X, Y, \ldots\}$

Похідний тип – множини станів змінних (наборів іменованих значень, наборів змінних з їхніми значеннями):

$State = Var \to Int$

Приклади станів змінних:

$[M \mapsto 8, N \mapsto 16]$,

$[M \mapsto 3, X \mapsto 4, Y \mapsto 2, N \mapsto 16]$

Операції на базових типах даних. Щоб відрізнити операції від символів, якими вони позначаються, будемо записувати їх спеціальними позначеннями.

Операції на множині Int. Символам $+$, $-$, $*$, $\div$ відповідають операції *add*, *sub*, *mult*, *div* (додавання, віднімання, множення, ділення). Це бінарні операції типу $Int^2 \to Int$.

Операції на множині Bool. Символам $\vee$, $\wedge$, $\neg$ відповідають операції *or*, *and*, *neg*. Це бінарні операції типу $Bool^2 \to Bool$ (диз'юнкція та кон'юнкція) та унарна операція типу $Bool \to Bool$ (заперечення).

У мові також є *операції змішаного типу*. Символам операцій порівняння $<$, $\leq$, $=$, $\neq$, $\geq$, $>$ відповідають операції *less*, *leq*, *eq*, *neq*, *geq*, *gr*. Це бінарні операції типу $Int^2 \to Bool$.

Ще однією основою мови SIPL є множина станів змінних. На ній задана бінарна операція *накладання* ∇ (іноді вживають термін *накладка*). Ця операція за двома станами будує новий стан

змінних, до якого входять усі іменовані значення з другого стану й ті значення з першого, імена яких не входять до другого стану. Операція подібна до операції копіювання каталогів зі спеціальним випадком однакових імен файлів у двох каталогах. У цьому випадку файл із першого каталогу замінюється на файл з тим самим іменем із другого каталогу. Наприклад:

$$[M\mapsto 8,\ N\mapsto 16]\ \nabla\ [M\mapsto 3,\ X\mapsto 4,\ Y\mapsto 2] =$$

$$= [M\mapsto 3,\ N\mapsto 16,\ X\mapsto 4,\ Y\mapsto 2]$$

Введемо відношення розширення $\subseteq$ станів новими іменованими значеннями (іменними парами). Наприклад:

$$[M\mapsto 8,\ N\mapsto 16] \subseteq [M\mapsto 8,\ X\mapsto 4,\ Y\mapsto 2,\ N\mapsto 16]$$

Для створення та оперування станами змінних треба визначити дві функції: *іменування* $\Rightarrow x$: $Int \to State$ та *розіменування* $x\Rightarrow$: $State \to Int$, які мають параметр $x \in Var$:

- $\Rightarrow x(n) = [x\mapsto n]$,
- $x\Rightarrow(st) = st(x)$

Тут і далі вважаємо, що $n \in Int$, $st \in State$. Перша функція іменує іменем x число n, створюючи стан $[x\mapsto n]$, друга бере значення імені x у стані st. Сама формула ґрунтується на тому факті, що стани змінних можуть тлумачитись як функції вигляду $Var \to Int$. Функція розіменування є частковою. Вона не визначена, якщо x не має значення в стані st. Наприклад:

- $\Rightarrow M(5) = [M\mapsto 5]$,
- $Y\Rightarrow([M\mapsto 3,\ X\mapsto 4,\ Y\mapsto 2]) = 2$,
- $Y\Rightarrow([M\mapsto 3,\ X\mapsto 4])$ не визначене.

Крім того, введемо:

- *функцію-константу арифметичного типу* $\overline{n}$: $State \to Int$ таку, що $\overline{n}(st) = n$,

- функції-константи булевого типу $\overline{true}$, $\overline{false}$: $State \to Bool$ такі, що $\overline{true}(st) = true$, $\overline{false}(st) = false$,
- *тотожну функцію id*: $State \to State$ таку, що $id(st) = st$.

Отримали багатоосновну *алгебру даних мови SIPL*:

$A\ \ Int\ \ Bool\ \ State = <Int, Bool, State; add, sub, mult, div, or, and, neg, less, leq, eq, neq, geq, gr, \Rightarrow x, x\Rightarrow, \overline{n}, id, \nabla, \overline{true}, \overline{false}>$

Функції

Аналіз алгебри даних показує, що в мові SIPL можна виділити два типи функцій: 1) *n*-арні функції на базових типах даних; 2) функції над станами змінних. Другий тип функцій будемо називати *номінативними функціями*. Назва пояснюється тим, що вони задані на наборах іменованих даних (латин. *nomen* – ім'я).

Зазначимо класи функцій, які будуть задіяні при визначенні семантики мови SIPL:

1. *n*-арні операції над базовими типами:
- $FNA = Int^n \to Int$ – *n*-арні арифметичні функції (операції);
- $FNB = Bool^n \to Bool$ – *n*-арні булеві функції (операції);
- $FNAB = Int^n \to Bool$ – *n*-арні функції (операції) порівняння.

2. Функції над станами змінних:
- $FA = State \to Int$ – номінативні арифметичні функції;
- $FB = State \to Bool$ – номінативні предикати;
- $FS = State \to State$ – біномінативні функції –перетворювачі (трансформатори) станів.

Для операцій мови SIPL зазвичай $n = 2$, а для булевої операції заперечення $n = 1$.

Композиції

Композиції формалізують методи побудови програм. Аналіз мови SIPL дає підстави вживати композиції різних типів, а саме:

- пов'язані з номінативними функціями та предикатами,
- пов'язані з біномінативними функціями.

Перший клас композицій використовується для побудови семантики арифметичних виразів та умов, другий – операторів.

Цей клас композицій складається з композицій *суперпозиції* в n-арні функції, які задані на різних основах (класах функцій):

- суперпозиція номінативних арифметичних функцій у n-арну арифметичну функцію має тип S^n: $FNA\times FA^n\to FA$;
- суперпозиція номінативних арифметичних функцій у n-арну функцію порівняння має тип S^n: $FNAB\times FA^n\to FB$;
- суперпозиція номінативних предикатів у n-арну булеву функцію має тип S^n: $FNB\times FB^n\to FB$.

Зауваження: суперпозиції різного типу позначаємо одним знаком.

Суперпозиція задається формулою (тут f – n-арна функція, $g_1,\dots,g_n$ – номінативні функції відповідного типу):
$(S^n(f, g_1,\dots,g_n))(st) = f(g_1(st),\dots,g_n(st))$.

Другий клас складається з таких композицій:

- *Присвоювання* AS^x: $FA \to FS$ (x – параметр). Задається формулою $AS^x(fa)(st) = st\,\nabla[x\mapsto fa(st)]$.
- *Послідовне виконання* $\bullet$: $FS^2\to FS$. Задається формулою $(fs_1\bullet fs_2)(st) = fs_2(fs_1(st))$.
- *Умовний оператор* (розгалуження): $IF{:}FB\times FS^2\to FS$. Задається формулою

$$IF\left(fb, fs_1, fs_2\right)\left(st\right) = \begin{cases} fs_1(st), \text{ якщо } fb((st) = true, \\ fs_2(st), \text{ якщо } fb(st) = false. \end{cases}$$

- *Цикл* (*ітерація з передумовою*): $WH{:}FB\times FS\to FS$. Задається рекурентно (індуктивно) таким чином:

$WH(fb,fs)(st) = st_n$,
де $st_0 = st$, $st_1 = fs(st_0)$, $st_2 = fs(st_1),\dots, st_n = fs(st_{n\text{-}1})$, причому
$fb(st_0) = true, fb(st_1) = true,\dots, fb(st_{n\text{-}1}) = true, fb(st_n) = false$.

Важливо зазначити, що для циклу наведена послідовність визначається однозначно. Позначимо число n (кількість ітерацій циклу) як $NumItWH((fb, fs), st)$. Однозначність означення n дозволяє розглядати $NumItWH((fb, fs), st)$ як тернарне відображення, що залежить від fb, fs, st. Якщо цикл не завершується, то $NumItWH((fb, fs), st)$ вважається невизначеним. Це відображення буде використовуватися в індуктивних доведеннях властивостей циклу.

Програмні алгебри

Побудовані композиції дозволяють стверджувати, що отримано *алгебру функцій* (*програмну алгебру*):

$A\_Prog=<FNA, FNB, FNAB, FA, FB, FS; S^n, AS^x, \bullet, IF, WH, x\Rightarrow, id>$.

Наведена алгебра містить багато "зайвих" функцій, які не можуть бути породжені в мові SIPL. Аналіз мови дозволяє стверджувати, що функції, які задаються мовою SIPL, породжуються в алгебрі *A_Prog* з таких *базових функцій*:

- $add, sub, mult, div \in FNA$,
- $or, and, neg \in FNB$,
- $less, leq, eq, neq, geq, gr \in FNAB$,
- $\overline{n} \in FA$ ($n \in Int$),
- $\overline{true}$, $\overline{false} \in FB$.

Підалгебру алгебри *A_Prog*, породжену наведеними базовими функціями, назвемо *функціональною алгеброю мови SIPL* і позначимо A_SIPL.

Зауважимо, що всі функції алгебри *A_SIPL* є *однозначними* (*детермінованими*). Це випливає з того, що всі базові функції є однозначними, а композиції зберігають таку властивість (довести цю властивість самостійно).

Формули для обчислення композицій та функцій алгебри *A_Prog* подамо у табл. 1.2 (тут f – n-арна функція, fa, $g_1,\ldots, g_n$ – номінативні арифметичні функції, fb – номінативний предикат, fs, fs_1, fs_2 – біномінативні функції, st – стан, n – число).

Таблиця 1.2

Формули для обчислення композицій та функцій алгебри

Композиція (функція)	Формула обчислення	Ім'я формули
Суперпозиція	$(S^n(f, g_1,\ldots, g_n))(st) = f(g_1(st),\ldots, g_n(st))$	*AF_S*
Присвоювання	$AS^x(fa)(st) = st\ \nabla[x \mapsto fa(st)]$	*AF_AS*
Послідовне виконання	$fs_1 \bullet fs_2(st) = fs_2(fs_1(st))$	*AF_SEQ*

Закінчення табл. 1.2

Композиція (функція)	Формула обчислення	Ім'я формули
Умовний оператор	$if\,(fb, fs_1, fs_2)(st) = \begin{cases} fs_1(st), \text{ якщо} \\ \quad fb(st) = true \\ fs_2(st), \text{ якщо} \\ \quad fb(st) = false \end{cases}$	*AF_IF*
Цикл	$WH(fb, fs)(st) = st_n$, де $st_0 = st$, $st_1 = fs(st_0)$, $st_2 = fs(st_1)$, …, $st_n = fs(st_{n-1})$, причому $fb(st_0) = true$, $fb(st_1) = true$,…, $fb(st_{n-1}) = true$, $fb(st_n) = false$	*AF_WH*
Функція розіменування	$x \Rightarrow (st) = st(x)$	*AF_DNM*
Тотожна функція	$id(st) = st$	*AF_ID*

Побудована алгебра *A_SIPL* дозволяє тепер формалізувати семантику програм мови SIPL, задаючи їх функціональними виразами (семантичними термами) алгебри *A_SIPL*.

1.3. Побудова семантичного терму програми

Програма мови SIPL може бути перетворена на семантичний терм (терм програмної алгебри), який задає семантику цієї програми (семантичну функцію програми), таким чином:

- *sem_P*: $Prog \to FS$;
- *sem_S*: $Stm \to FS$;
- *sem_A*: $Aexp \to FA$;
- *sem_B*: $Bexp \to FB$.

Ці перетворення задаються рекурсивно, за структурою програми (табл. 1.3). Тому побудова семантичного терму залежить від вибору структури синтаксичного запису програми. Однак треба зважати на неоднозначність обраної нами граматики, що може зумовити різну семантику програм. Для досягнення однозначності треба користуватись пріоритетами операцій і типом їх асоціативності.

Таблиця 1.3

Правила перетворення

Правило заміни	Назва правила
sem_P: *Prog* → *FS* задається правилами:	
sem_P(begin *S* end)=*sem_S*(*S*)	*NS_Prog*
sem_S: *Stm* → *FS* задається правилами:	
$sem\_S(x:=a)=AS^{x}(sem\_A(a))$	*NS_Stm_As*
$sem\_S(S_1;S_2)= sem\_S(S_1) \bullet sem\_S(S_2)$	*NS_Stm_Seq*
$sem\_S(\text{if } b \text{ then } S_1 \text{ else } S_2)=IF(sem\_B(b), sem\_S(S_1), sem\_S(S_2))$	*NS_Stm_If*
$sem\_S(\text{while } b \text{ do } S)=WH(sem\_B(b), sem\_S(S))$	*NS_Stm_Wh*
$sem\_S(\text{begin } S \text{ end})=(sem\_S(S))$	*NS_Stm_Op*
$sem\_S(skip)=id$	*NS_Stm_skip*
sem_A: *Aexp* → *FA* задається правилами:	
$sem\_A(n)=\overline{n}$	*NS_A_Num*
$sem\_A(x)=x\Rightarrow$	*NS_A_Var*
$sem\_A(a_1+a_2)=S^{2}(add, sem\_A(a_1), sem\_A(a_2))$	*NS_A_Add*
$sem\_A(a_1-a_2)=S^{2}(sub, sem\_A(a_1), sem\_A(a_2))$	*NS_A_Sub*
$sem\_A(a_1*a_2)=S^{2}(mult, sem\_A(a_1), sem\_A(a_2))$	*NS_A_Mult*
$sem\_A(a_1\div a_2)=S^{2}(div, sem\_A(a_1), sem\_A(a_2))$	*NS_A_Div*
$sem\_A((a))=sem\_A(a)$	*NS_A_Par*

Закінчення табл. 1.3

Правило заміни	Назва правила
sem_B: *Bexp* $\rightarrow$ *FB* задається правилами:	
$sem\_B(true)=\overline{true}$	*NS_B_true*
$sem\_B(false)=\overline{false}$	*NS_B_false*
$sem\_B(a_1<a_2)=S^2(less, sem\_A(a_1), sem\_A(a_2))$	*NS_B_less*
$sem\_B(a_1\leq a_2)=S^2(leq, sem\_A(a_1), sem\_A(a_2))$	*NS_B_leq*
$sem\_B(a_1=a_2)=S^2(eq, sem\_A(a_1), sem\_A(a_2))$	*NS_B_eq*
$sem\_B(a_1\neq a_2)=S^2(neq, sem\_A(a_1), sem\_A(a_2))$	*NS_B_neq*
$sem\_B(a_1\geq a_2)=S^2(geq, sem\_A(a_1), sem\_A(a_2))$	*NS_B_geq*
$sem\_B(a_1>a_2)=S^2(gr, sem\_A(a_1), sem\_A(a_2))$	*NS_B_gr*
$sem\_B(b_1\vee b_2)=S^2(or, sem\_B(b_1), sem\_B(b_2))$	*NS_B_or*
$sem\_B(b_1\wedge b_2)=S^2(and, sem\_B(b_1), sem\_B(b_2))$	*NS_B_and*
$sem\_B(\neg b)=S^1(neg, sem\_B(b))$	*NS_B_neg*
$sem\_B((b))=sem\_B(b)$	*NS_B_Par*

Наведені правила слід розглядати як загальні, які в логіці називають *схемами правил*. Щоб із *загального* правила (метаправила) отримати *конкретне* (об'єктне правило), слід замість синтаксичних метасимволів, таких як *a*, *b*, *S*, *P*, *n*, *x*, підставити конкретні синтаксичні елементи (записи), наприклад замість *a* підставити *N–M*, замість *b* – *M>N* тощо. Далі ліва частина конкретного правила заміняється на його праву частину і т. д.

1.4. Доведення коректності програм

Доведення коректності програм можна виконувати різними способами, зокрема (див. розд. 5) за допомогою аксіоматичної семантики. Головна задача – показати часткову та/або тотальну коректність програми, тобто те, що вона на довільних вхідних даних видає правильні, заздалегідь очікувані, результати (такі, що відповідають початковій формальній специфікації програми). Одним з підходів до доведення часткової коректності програм у композиційній семантиці є обчислення значення семантичного терму програми над деяким узагальненим даним *d*, що має

вигляд пар ім'я-значення без конкретизації фіксованих значень змінних, тобто *sem_P*(*P*) (*d*), і порівняння отриманого даного (точніше – окремих його компонент – значень змінних) з очікуваним згідно з формальною специфікацією. Тотальна коректність доводиться шляхом доведення завершуваності програми. Останнє найчастіше зводиться до доведення завершуваності циклів, що входять до програми.

1.5. Розв'язання типових задач

Завдання 1.1. За допомогою алгоритму Евкліда знайти найбільший спільний дільник двох чисел GCD(*M*, *N*). Виконати такі етапи:

1) побудувати програму для розв'язання задачі мовою SIPL;
2) побудувати дерево синтаксичного виведення програми;
3) побудувати композиційний семантичний терм програми;
4) обчислити значення семантичного терму над вхідним даним [*M*↦15, *N*↦9].

Розв'язання.

Алгоритм Евкліда для пошуку найбільшого спільного дільника двох чисел є таким:

1) Вхід: цілі числа *m* та *n*.
2) Якщо $m = n$, то GCD $(m, n) = m = n$.
3) Якщо $m > n$, то покласти $m = m - n$, інакше покласти $n = n - m$. Далі див. п. 2.

Суть алгоритму полягає в такому. Нехай *gcd* (m, n) – функція, що повертає найбільший спільний дільник чисел *m* та *n*. Якщо $m = n$, то *gcd* $(m, n) = m = n$. Якщо $m > n$, то $gcd(m, n) = gcd(m - n, n)$; якщо $n > m$, то $gcd(m, n) = gcd(m, n - m)$. Доведемо це.

Нехай $gcd(m, n) = p$, тоді *m* ділиться на *p* та *n* ділиться на *p*, відповідно $m - n$ та $n - m$ також діляться на *p*. Отже, $gdc(m - n, n)$ та $gdc(m, n - m)$ діляться на $gcd(m, n)$. Нехай $q = gcd(m - n, n)$. Тоді маємо, що існують такі цілі t_1 і t_2, що $m - n = t_1 q$, $n = t_2 q$, тоді $m = m - n + n = t_1 q + t_2 q = (t_1 + t_2)q$, $gcd(m, n)$ ділиться на *q*. Сформулюємо ще одну властивість: з того, що *p* ділиться на *q* та *q* ділиться на *p*, отримуємо, що $p = q$.

Алгоритм Евкліда обчислення найбільшого спільного дільника мовою SIPL:

```
GCD =
begin
 while M ≠ N do
 if M > N then
      M := M – N
 else
      N := N – M
end
```

Побудуємо дерево синтаксичного виведення наведеної програми згідно з правилами БНФ з табл. 1.1, які задають мову SIPL.

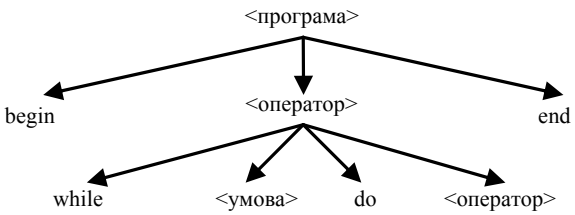
За правилом *NP*1 БНФ мови SIPL бачимо, що нетермінал <програма> подається у вигляді <програма> ::= begin <оператор> end.

Отже, на першому кроці дерево виведення матиме вигляд:

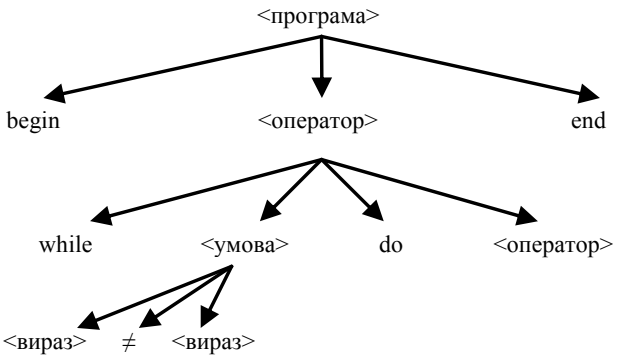
На другому кроці розпишемо нетермінал <оператор>. Виходячи з коду нашої програми, наш нетермінал <оператор> має такий вигляд:

<оператор> ::= while <умова> do <оператор> (правило *NS*4)

Отже, на другому кроці дерево виведення буде мати такий вигляд:

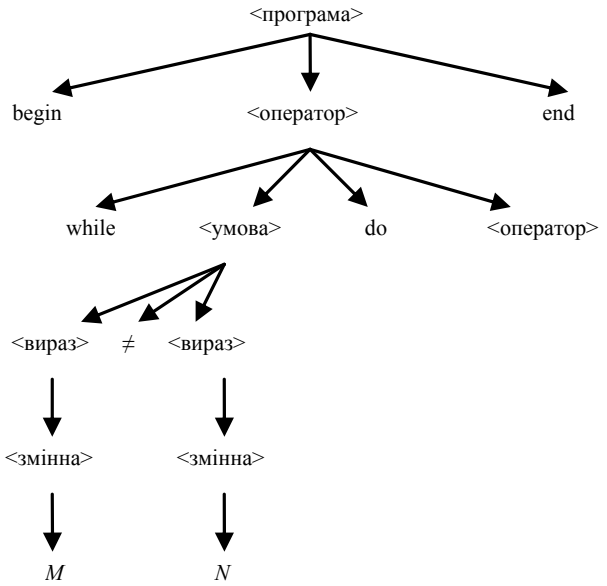


Далі розпишемо нетермінал <умова>. У нашому випадку буде: <умова> ::= <вираз> ≠ <вираз> (правило *NB*4)



Тут кожен <вираз> подається у вигляді <вираз>::=<змінна> (правило *NA*2), де в першому випадку<змінна>::= *M*, а в другому <змінна>::=*N*.

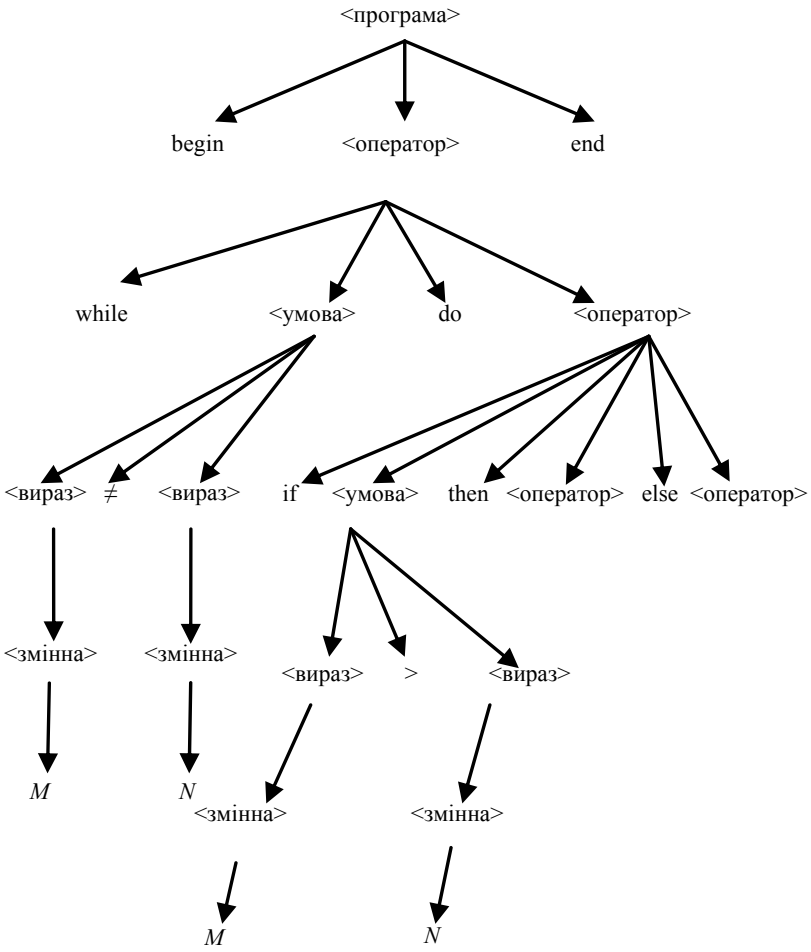
Отже, дерево синтаксичного виведення на такому кроці має вигляд:



Тепер розпишемо другу частину оператора *while*.

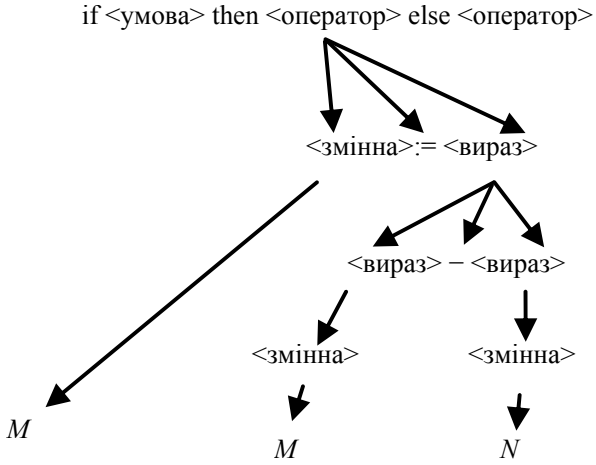
Виходячи з коду програми, відповідний нетермінал має вигляд: <оператор> ::= if <умова> then <оператор> else <оператор> (правило *NS*3), де умова оператора if: <умова> ::= <вираз> > <вираз> (правило *NS*7), де кожен з виразів представлений змінною *M* та *N*, відповідно.

Отже, дерево виведення на такому кроці матиме вигляд:

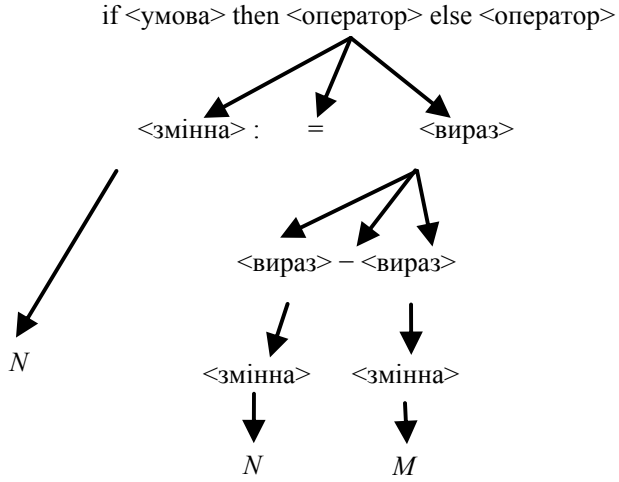


Далі залишилось розписати два оператори. Перший виконується тоді, коли умова оператора if істинна, другий – коли хибна.

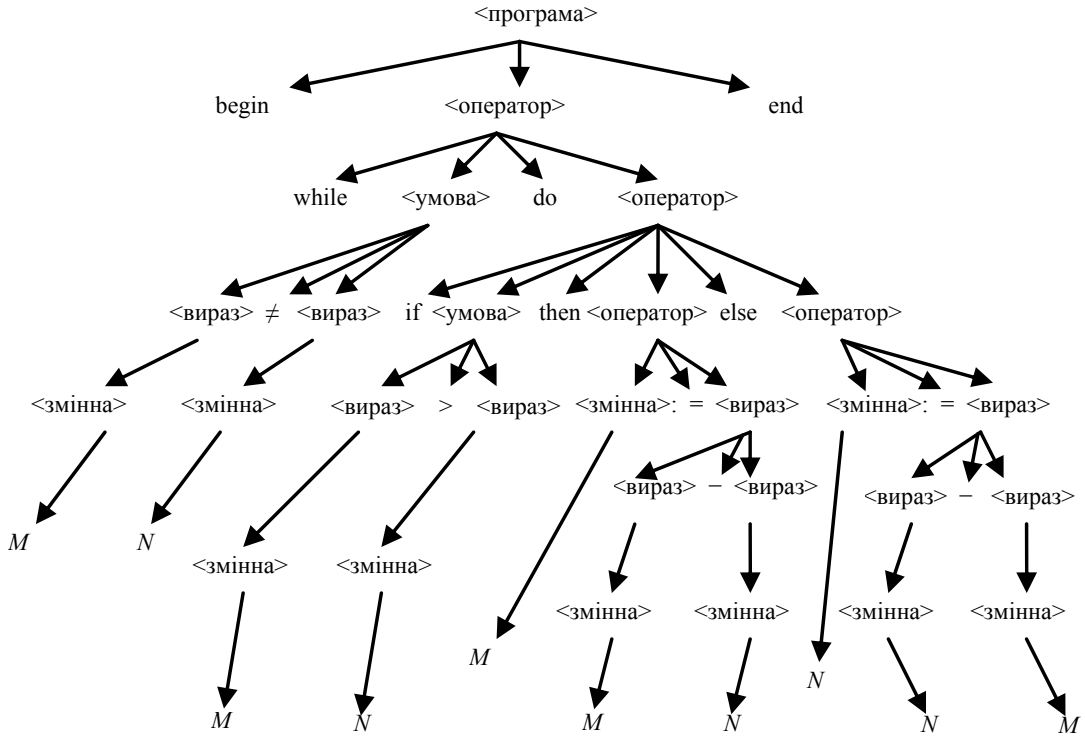
Неважко переконатись, що в першому випадку на дереві виведення <оператор> буде мати такий вигляд:



У другому випадку:



Отже, остаточно дерево синтаксичного виведення для нашої програми буде мати такий вигляд:



Для перевірки коректності побудови дерева треба обійти його зліва направо, записуючи всі термінали (ті вирази, що записані без кутових дужок). У результаті має бути отриманий вихідний текст програми мовою SIPL.

Композиційний семантичний терм визначимо згідно зі структурою програми за деревом виведення. Використаємо правила його побудови в мові SIPL, що задані в табл. 1.3.

sem_P(GCD)

Застосовуємо правило *NS_Prog*:

sem_P(begin
while $M \neq N$ do
if $M > N$ then $M := M - N$ else $N := N - M$
end)

Застосовуємо правило *NS_Stm_Wh*:

sem_S(while $M \neq N$ do
if $M > N$ then $M := M - N$
else $N := N - M$
) =
$= WH$(*sem_B*($M \neq N$)
sem_S(if $M > N$
then $M := M - N$
else $N := N - M$)
)

Для умови циклу застосовуємо правило *NS_B_neq*, а для тіла *NS_Stm_If*:

$WH(S^2$(*neq*, *sem_A*(*M*), *sem_A*(*N*)),
IF(sem_B($M > N$),
sem_S($M := M - N$),
sem_S($N := N - M$))
)

Для кожної зі змінних M та N в умові циклу застосовуємо правило *NS_A_Var*, для умови умовного оператора – правило *NS_B_gr*, а для обох його операторів – правило *NS_Stm_As*:

$WH(S^2(neq, M\Rightarrow, N\Rightarrow)),$
$IF(S^2(gr, sem\_A(M), sem\_A(N)),$
$AS^M(sem\_A(M - N)),$
$AS^N(sem\_A(N - M)))$
)

Для кожної зі змінних M та N в умові умовного оператора застосовуємо правило *NS_A_Var*, для інших двох його операторів – правило *NS_A_Sub*:

$WH(S^2(neq, M\Rightarrow, N\Rightarrow),$
$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, sem\_A(M), sem\_A(N))),$
$AS^N(S^2(sub, sem\_A(N), sem\_A(M))))$
)

Для кожної зі змінних M та N двічі застосовуємо правило *NS_A_Var*:

$WH(S^2(neq, M\Rightarrow, N\Rightarrow),$
$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow)))$
)

Процес обчислення значення задається формулами табл. 1.2. Щоб їх застосовувати, треба конкретизувати загальні правила. Завдання полягає в отриманні значення аплікації – застосуванні функції, що задається нашим термом, до конкретного даного.

Обчислимо $sem\_P(\mathrm{GCD})([M\mapsto 15, N\mapsto 9]) =$

$= WH(\ S^2(neq, M\Rightarrow, N\Rightarrow),$
$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))))$
$([M\mapsto 15, N\mapsto 9])$

Правила для обчислення циклу *AF_WH* зумовлюють необхідність поступового обчислення станів st_0, st_1, … і виконання тіла циклу (оператора *fs*) до тих пір, поки умова *fb* є істинною. Обчислимо значення умови циклу: $S^2(neq, M\Rightarrow, N\Rightarrow)([M\mapsto 15, N\mapsto 9]) = neq(15, 9) = true$, тому виконуємо тіло циклу з композицією початкового циклу:

$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))$
$) \bullet WH(\ldots, \ldots)([M\mapsto 15, N\mapsto 9]) =$

$= WH(\ldots, \ldots)(IF(S^2(gr, M\Rightarrow, N\Rightarrow), AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)), AS^N(S^2(sub, N\Rightarrow, M\Rightarrow)))([M\mapsto 15, N\mapsto 9]))$

Обчислимо значення умови оператора *IF*:

$S^2(gr, M\Rightarrow, N\Rightarrow)([M\mapsto 15, N\mapsto 9]) = gr(15,9) = true,$

тому виконуємо першу з двох альтернатив оператора *IF*:

$WH(\ldots, \ldots)(AS^M(S^2(sub, M\Rightarrow, N\Rightarrow))([M\mapsto 15, N\mapsto 9]))$

Обчислимо його, розписуючи присвоювання, віднімання та розіменування: $AS^M(S^2(sub, M\Rightarrow, N\Rightarrow))([M\mapsto 15, N\mapsto 9]) =$

$= [M\mapsto 15, N\mapsto 9] \nabla [M\mapsto S^2(sub, M\Rightarrow, N\Rightarrow)([M\mapsto 15, N\mapsto 9])] =$

$= [M\mapsto 15, N\mapsto 9] \nabla [M\mapsto sub(15,9)] =$

$= [M\mapsto 15, N\mapsto 9] \nabla [M\mapsto 6] =$

$= [M\mapsto 6, N\mapsto 9]$

Отже, ми виконали першу ітерацію циклу. Далі, за аналогією, обчислюємо значення на новому стані $[M\mapsto 6, N\mapsto 9]$, тобто

$WH(S^2(neq, M\Rightarrow, N\Rightarrow),$
$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))))$
$([M\mapsto 6, N\mapsto 9])$

Оскільки $S^2(neq, M\Rightarrow, N\Rightarrow)\ ([M\mapsto 6, N\mapsto 9]) = neq(6,9) = true$, то

$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))$
$) \bullet WH(\ldots, \ldots)([M\mapsto 6, N\mapsto 9]) =$
$= WH(\ldots, \ldots)\ ($
$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))$
$)([M\mapsto 6, N\mapsto 9]))$

Обчислюємо умову оператора розгалуження:

$S^2(gr, M\Rightarrow, N\Rightarrow)([M\mapsto 6, N\mapsto 9]) = gr(6,9) = false$

Оскільки вона дорівнює *false*, то цикл обчислюємо на другому операнді оператора розгалуження:

$WH(\ldots, \ldots)(AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))([M\mapsto 6, N\mapsto 9]))$

Обчислимо цей операнд:

$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))([M\mapsto 6, N\mapsto 9]) =$
$= [M\mapsto 6, N\mapsto 9]\ \nabla\ [N\mapsto S^2(sub, N\Rightarrow, M\Rightarrow)([M\mapsto 6, N\mapsto 9])] =$
$= [M\mapsto 6, N\mapsto 9]\ \nabla\ [N\mapsto sub(9,6)] =$
$= [M\mapsto 6, N\mapsto 9]\ \nabla\ [N\mapsto 3] =$
$= [M\mapsto 6, N\mapsto 3]$

Виконаємо цикл на отриманому стані:

$WH(S^2(neq, M\Rightarrow, N\Rightarrow),$
$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))))$
$([M\mapsto 6, N\mapsto 3])$

Знову розраховуємо умову циклу:

$S^2(neq, M\Rightarrow, N\Rightarrow)([M\mapsto 6, N\mapsto 3]) = neq(6,3) = true$

Вона істинна, тому виконуємо таку ітерацію циклу:

$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))$
$) \bullet WH(\ldots, \ldots)([M\mapsto 6, N\mapsto 3]) =$
$= WH(\ldots, \ldots)(IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))$
$)([M\mapsto 6, N\mapsto 3]))$

Обчислюємо умову оператора розгалуження:

$S^2(gr, M\Rightarrow, N\Rightarrow)([M\mapsto 6, N\mapsto 3]) = gr(6,3) = true$

Вона істинна, тому обчислюємо перший операнд:

$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow))([M\mapsto 6, N\mapsto 3]) =$
$= [M\mapsto 6, N\mapsto 3] \nabla [M\mapsto S^2(sub, M\Rightarrow, N\Rightarrow)([M\mapsto 6, N\mapsto 3])] =$
$= [M\mapsto 6, N\mapsto 3] \nabla [M\mapsto sub(6,3)] =$
$= [M\mapsto 6, N\mapsto 3] \nabla [M\mapsto 3] =$
$= [M\mapsto 3, N\mapsto 3]$

Знову обчислюємо цикл на отриманому стані:

$WH(S^2(neq, M\Rightarrow, N\Rightarrow), IF(S^2(gr, M\Rightarrow, N\Rightarrow), AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$
$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))))([M\mapsto 3, N\mapsto 3])$

Значення умови циклу становить

$S^2(neq, M\Rightarrow, N\Rightarrow)([M\mapsto 3, N\mapsto 3]) = neq(3,3) = false,$

тому зупиняємо обчислення з поточним результатом, який дорівнює $[M\mapsto 3, N\mapsto 3]$.

Отже, $sem\_P(\text{GCD})([M\mapsto 15, N\mapsto 9]) = [M\mapsto 3, N\mapsto 3]$.

Для зручності та скорочення можна перепозначати оператори і стани, що виникають у процесі обчислення, наприклад:

$F_0 := WH(S^2(neq, M\Rightarrow, N\Rightarrow),$
$IF(S^2(gr, M\Rightarrow, N\Rightarrow),$
$AS^M(S^2(sub, M\Rightarrow, N\Rightarrow)),$

$AS^N(S^2(sub, N\Rightarrow, M\Rightarrow))))$

або $d_0 = [M\mapsto 15, N\mapsto 9]$. Тоді умову можна переписати таким чином: знайти F0(d0) і продовжувати обчислення. Оскільки треба ретельно слідкувати за даними та їх змінами, то краще записувати їх в окремій таблиці.

Завдання 1.2. Розробити програму, яка за допомогою операції додавання одиниці додає два числа A та B (A, $B \geq 0$) і записує результат у змінну R.

1) Побудувати програму для розв'язання задачі мовою SIPL.
2) Побудувати дерево синтаксичного виведення програми.
3) Побудувати композиційний семантичний терм програми.
4) Довести коректність програми.

Розв'язання.

1) Програма для розв'язання задачі:

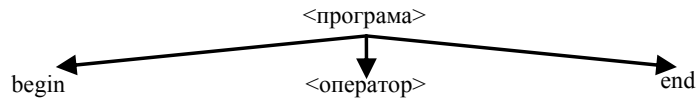
```
begin
    R := A;
    I := 0;
    while I < B do
    begin
            R := R + 1;
            I := I + 1
    end
end
```

2) Дерево синтаксичного виведення програми.

Першим кроком розпишемо нетермінал <програма>

<програма> ::= begin <оператор> end.

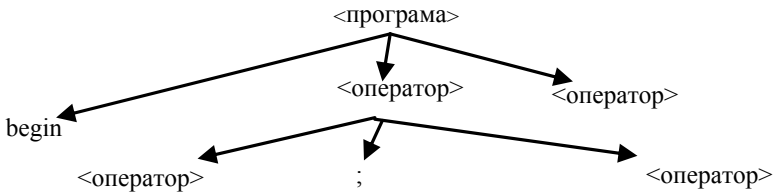
Дерево синтаксичного виведення на першому кроці має такий вигляд:



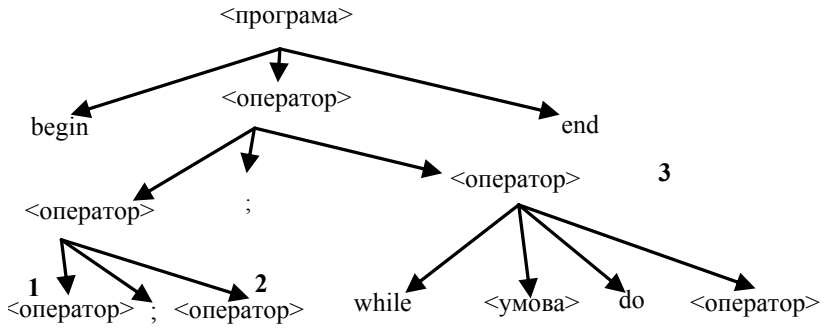
Далі необхідно розписати метазмінну <оператор>. Зображуємо цей нетермінал:

<оператор> ::= <змінна>:=<вираз> | <оператор>; <оператор>|
if <умова> then <оператор> else <оператор> |
while <умова> do <оператор> |
begin <оператор> end | skip

Виходячи з коду програми, на другому кроці дерево виведення матиме вигляд:



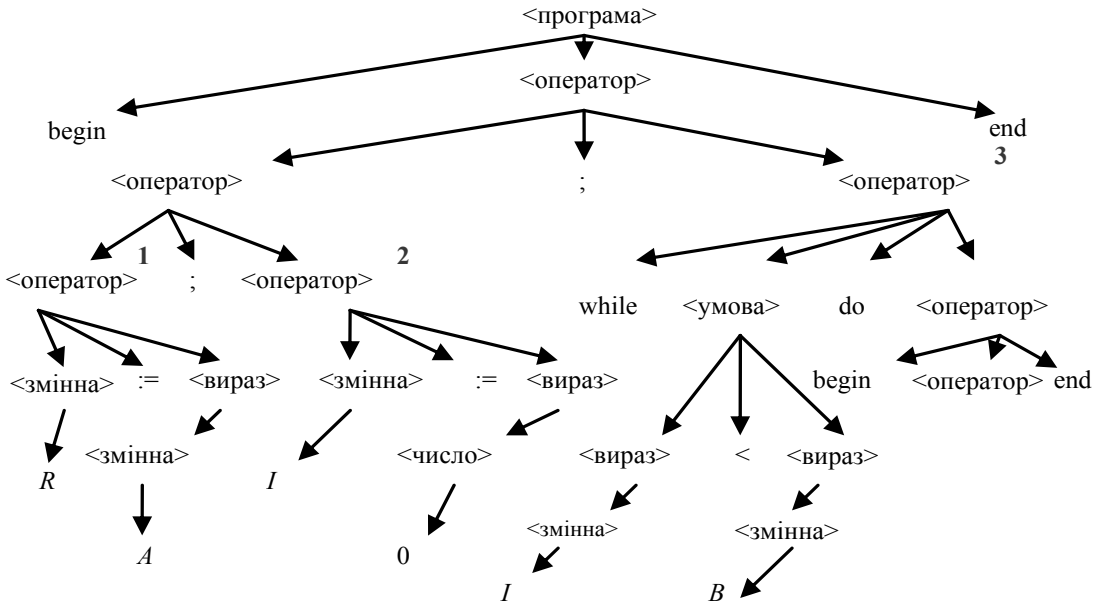
Розписуючи за правилом, наведеним вище, на третьому кроці отримаємо таке дерево виведення:



Позначення операторів числами 1, 2, 3 будуть використані далі при доведенні коректності програм.

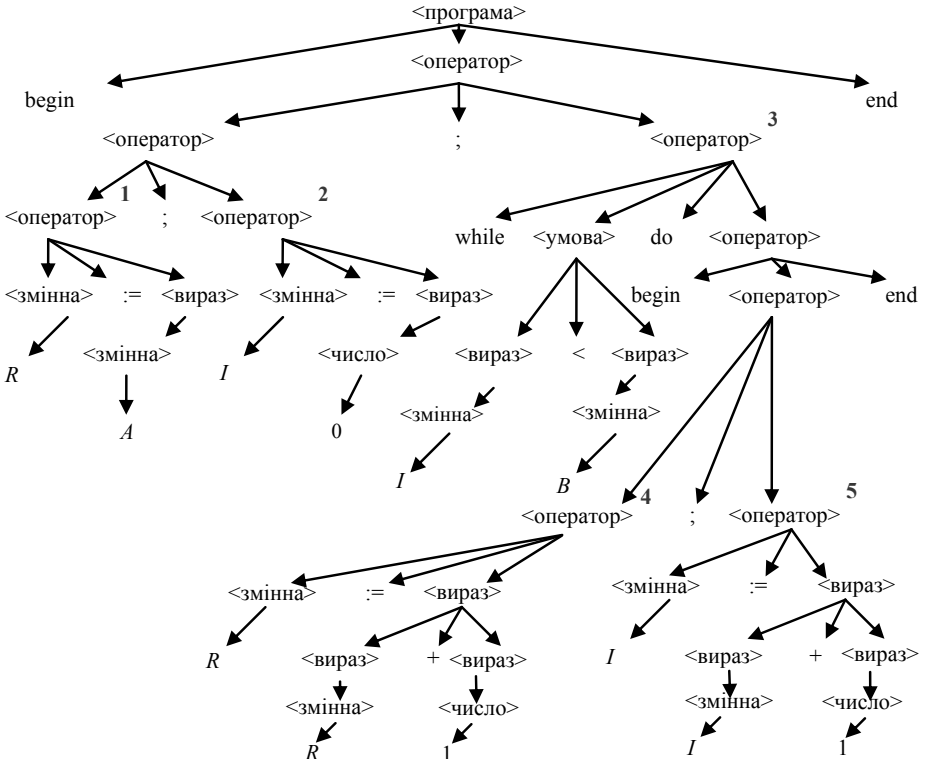
Далі треба розписати метазмінні <умова>, <змінна> та <вираз>. Зображуємо ці нетермінали згідно із синтаксисом мови SIPL:

<вираз> ::= <число> | <змінна> | <вираз> + <вираз> |
<вираз> – <вираз> |
<вираз> * <вираз> | <вираз> ÷ <вираз> | (<вираз>)



<умова> ::= true | false | <вираз> < <вираз> | <вираз> ≤ <вираз> |
<вираз> = <вираз> | <вираз> ≠ <вираз> | <вираз> > <вираз> |
<вираз> ≥ <вираз> | <умова> ∨ <умова> | <умова> ∧ <умова> |
¬ <умова> | (<умова>)

Продовжуючи цей процес, отримаємо остаточний вигляд дерева:



3) Семантичний композиційний терм отримаємо без деталізації всіх кроків перетворень:

$$
\begin{aligned}
&sem\_P(Program) = \\
&\ sem\_S(R := A) \bullet \\
&\ sem\_S(I := 0) \bullet \\
&\ sem\_S(\text{while } I < B \text{ do} \\
&\ \text{begin} \\
&\qquad R := R + 1; \\
&\qquad I := I + 1 \\
&\qquad\qquad \text{end} \\
&) = \\
&\ = AS^{R}(A\Rightarrow) \bullet \\
&\ AS^{I}(\overline{0}) \bullet \\
&\ WH(S^{2}(less, sem\_A(I), \\
&\qquad sem\_A(B)), \\
&\qquad sem\_S(R := R + 1; I := I + 1)) = \\
&= AS^{R}(A\Rightarrow) \bullet \\
&\ AS^{I}(\overline{0}) \bullet \\
&\ WH(S^{2}(less, I\Rightarrow, B\Rightarrow), \\
&\qquad AS^{R}(S^{2}(add, R\Rightarrow, \overline{1})) \bullet \\
&\qquad AS^{I}(S^{2}(add, I\Rightarrow, \overline{1})) \\
&\ )
\end{aligned}
$$

4) Доведення коректності програми.

Доведення *часткової* коректності.

Композиційні терми для зазначених у дереві виведення операторів (з 1 по 5) будемо позначати відповідно f_1, f_2, f_3, f_4, та f_5, тобто

$$
\begin{aligned}
&f_1 = AS^{R}(A\Rightarrow) \\
&f_2 = AS^{I}(\overline{0}) \\
&f_3 = WH(S^{2}(less, I\Rightarrow, B\Rightarrow), \\
&\qquad\qquad AS^{R}(S^{2}(add, R\Rightarrow, \overline{1})) \bullet \\
&\qquad\qquad AS^{I}(S^{2}(add, I\Rightarrow, \overline{1})) \\
&\qquad) \\
&f_4 = AS^{R}(S^{2}(add, R\Rightarrow, \overline{1})) \\
&f_5 = AS^{I}(S^{2}(add, I\Rightarrow, \overline{1}))
\end{aligned}
$$

Обчислимо терм на стані $d = [A \mapsto a, B \mapsto b]$:

$sem\_P(Program)(d) =$
$= (f_1 \bullet f_2 \bullet f_3)(d) =$
$= f_3(f_2\ (f_1(d))) =$
$= f_3(f_2\ (d \nabla\ [R \mapsto A(d)])) =$
$= f_3(f_2\ (d \nabla\ [R \mapsto a])) =$
$= f_3(f_2\ (d_1)),$
де $d_1 = [A \mapsto a, B \mapsto b, R \mapsto a] =$
$= f_3(d_1 \nabla\ [I \mapsto 0]) =$
$= f_3(d_2),$
де $d_2 = [A \mapsto a, B \mapsto b, R \mapsto a, I \mapsto 0]$.

Далі доведемо, що якщо значення $f_3(d_2)$ визначене, тобто цикл завершується, то результатом будуть дані $[A \mapsto a,\ B \mapsto b,\ R \mapsto a+b,\ I \mapsto b]$.

Позначимо $dw = d_2$ та $g = f_3$. Висунемо гіпотезу.

Гіпотеза: при виконанні n кроків тіла циклу
$g(dw) = [A \mapsto a, B \mapsto b, R \mapsto a + n, I \to n]$.

Доведення виконуватимемо індукцією за кількістю кроків циклу n.

База індукції. Нехай $n = 0$, тобто цикл не виконується жодного разу. Це означає, що умова циклу на стані dw є хибною:
$(S^2(less, I\Rightarrow, B\Rightarrow))(dw) = false$, тому $g(dw) = dw$.

Розпишемо обчислення умови детальніше:
$(S^2(less, I\Rightarrow, B\Rightarrow))(dw) =$
$= less(I\Rightarrow(dw), B\Rightarrow(dw)) =$
$= less(0, b) = 0 < b = false.$
Отже, $b \le 0$. Однак за умовою задачі $b \ge 0$. Таким чином, маємо $b = 0$. Звідси $g([A \mapsto a, B \mapsto 0, R \mapsto a, I \mapsto 0]) = [A \mapsto a, B \mapsto 0, R \mapsto a, I \mapsto 0]$. Як бачимо, програма працює коректно для цього випадку.

Крок індукції. Нехай гіпотеза справедлива для t кроків циклу, тобто дані після t-го кроку мають вигляд $[A\mapsto a,\ B\mapsto b,\ R\mapsto a+t,\ I\mapsto t\ ]$. Покажемо, як зміняться дані після виконання $t+1$ кроку циклу $g([A\mapsto a,\ B\mapsto b,\ R\mapsto a+t,\ I\mapsto t])$.

Позначимо $d_t=[A\mapsto a,\ B\mapsto b,\ R\mapsto a+t,\ I\mapsto t]$.

Оскільки для виконання наступного кроку умова циклу має бути істинною, то

$$g(d_t)=AS^R(S^2(add,\ R\Rightarrow,\ \overline{1}\ ))\bullet AS^I(S^2(add,\ I\Rightarrow,\ \overline{1}\ ))(d_t)=$$
$$=AS^I(S^2(add,\ I\Rightarrow,\ \overline{1}\ ))(d_t\nabla AS^R(S^2(add,\ R\Rightarrow,\ \overline{1}\ ))(d_t))=$$
$$=AS^I(S^2(add,\ I\Rightarrow,\ \overline{1}\ ))(\ d_t\nabla[\mathrm{R}\mapsto a+t+1])=$$
$$=AS^I(S^2(add,\ I\Rightarrow,\ \overline{1}\ ))([A\mapsto a,\ B\mapsto b,\ R\mapsto a+t+1,\ I\mapsto t]).$$

Позначимо $s_t=[A\mapsto a,\ B\mapsto b,\ R\mapsto a+t+1,\ I\mapsto t]$.

Тоді

$$AS^I(S^2(add,\ I\Rightarrow,\ \overline{1}\ ))(s_t)=$$
$$=s_t\nabla[I\mapsto t+1]=$$
$$=[A\mapsto a,\ B\mapsto b,\ R\mapsto a+t+1,\ I\mapsto t+1].$$

Отже, гіпотезу доведено.

Тепер неважко перевірити, що після виконання b-го кроку циклу умова стане хибною і цикл завершиться. Дійсно, згідно з доведеною гіпотезою після b-го кроку дані будуть дорівнювати $d_b=[A\mapsto a,\ B\mapsto b,\ R\mapsto a+b,\ I\mapsto b]$. Обчислимо на них умову: $(S^2(less,\ I\Rightarrow,\ B\Rightarrow))(\ d_b)=less(b,\ b)=b<b=false.$ Також безпосередньо перевіримо, що коли кількість ітерацій строго менша b, то умова циклу істинна, тобто цикл завершується на b-му кроці та значенням R буде $a+b$.

Таким чином, ми показали часткову коректність програми при правильних вхідних даних (тобто $a>0$ та $b>0$).

Обґрунтування повної коректності.

Покажемо, що при правильних вхідних даних програма завершується (зупиняється). Фактично треба показати завершуваність циклу g, оскільки інші оператори не впливають на завершення програми.

З часткової коректності випливає, що правильні вхідні дані перед виконанням циклу мають вигляд $[A \mapsto a, B \mapsto 0, R \mapsto a, I \mapsto 0]$, де $b>0$ та $a>0$. Після кожного виконання циклу значення I збільшується на 1. Це означає, що після b кроків умова $I < B$ стане хибною і цикл завершиться.

Отже, унаслідок скінченності значення B, якщо програма входить у цикл, то він завжди завершується. За інших варіантів правильних вхідних даних програма не входить у цикл, а послідовно виконує прості оператори, тому вона завжди зупиняється внаслідок скінченності кількості операторів.

Завдання 1.3. За допомогою операцій віднімання й додавання розробити програму, яка обчислює результат множення цілих чисел A та B. Виконати:

1) побудувати програму для розв'язання задачі мовою SIPL;
2) побудувати композиційний семантичний терм програми;
3) довести коректність програми.

Розв'язання.

1) Програма для розв'язання задачі:

```
begin
        if B < 0 then
        begin
                A := 0 – A;
                B := 0 – B
        end;
        R := 0;
        while B > 0 do
        begin
                R := R + A;
                B := B – 1
        end
end
```

Зазначимо, що в мові SIPL відсутня операція унарного мінуса, тобто вираз вигляду –X синтаксично невірний. Тому така операція записується через бінарній мінус: $0 – X$.

2) Семантичний композиційний терм:

$sem\_P(Program) =$

$= sem\_P($

begin

if $B < 0$ then begin $A := 0 - A; B := 0 - B$ end;

$R := 0;$

while $B > 0$ do begin $R := R + A; B := B - 1$ end

end

$) =$

$= sem\_S($

if $B < 0$ then begin $A := 0 - A; B := 0 - B$ end;

$R := 0;$

while $B > 0$ do begin $R := R + A;\ B := B - 1$ end

$) =$

$= sem\_S(\text{if } B < 0 \text{ then begin } A := 0 - A;\ B := 0 - B \quad \text{end}) \bullet$

$\bullet\ sem\_S(R := 0; \text{ while } B > 0 \text{ do begin } R := R + A;\ B := B - 1 \text{ end}) =$

$= IF(sem\_B(B < 0), sem\_S(\text{begin } A := 0 - A;\ B := 0 - B \text{ end}), id)) \bullet$

$\bullet\ sem\_S(R := 0; \text{ while } B > 0 \text{ do begin } R := R + A;\ B := B - 1 \text{end}) =$

$= IF(sem\_B(B < 0), sem\_S(\text{begin } A := 0 - A;\ B := 0 - B \text{ end}), id)) \bullet$

$\bullet\ AS^R(\overline{0}) \bullet$

$\bullet\ WH(sem\_B(B > 0), sem\_S(\text{begin } R := R + A; B := B - 1 \text{ end})) =$

$= IF(S2(less, sem\_A(B), sem\_A(0)), sem\_S(A := 0 - A;\ B := 0 - B), id) \bullet$

$\bullet\ AS^R(\overline{0}) \bullet$

$\bullet\ WH(S2(gr, sem\_A(B), sem\_A(0)), sem\_S(R := R + A;\ B := B - 1)) =$

$= IF(S2(less, B\Rightarrow, \overline{0}), sem\_S(A := 0 - A) \bullet sem\_S(B := 0 - B), id) \bullet$

$\bullet\ AS^R(\overline{0}) \bullet$

$\bullet\ WH(S^2(gr, B\Rightarrow, \overline{0}), sem\_S(R := R + A) \bullet sem\_S\ (B := B - 1)) =$

$= IF(S2(less, B\Rightarrow, \overline{0}), ASA(\text{sem\_A}(0 - A)) \bullet ASB(\text{sem\_A}(0 - B)), id) \bullet$

$\bullet\ AS^R(\overline{0}) \bullet$

$\bullet\ WH(S^2(gr, B\Rightarrow, \overline{0}), sem\_S(R := R + A) \bullet sem\_S(B := B - 1)) =$

$= IF(S^2(less, B\Rightarrow, \overline{0}), AS^A(S^2(sub, sem\_A(0), sem\_A(A))) \bullet$
$\bullet\ AS^B(S^2(sub, sem\_A(0), sem\_A(B))), id)) \bullet$
$\bullet\ AS^R(\overline{0}) \bullet$
$\bullet\ WH(S^2(gr, B\Rightarrow, \overline{0}), AS^R(sem\_A(R + A)) \bullet AS^B(sem\_A(B - 1)) =$
$= IF(S^2(less, B\Rightarrow, \overline{0}), AS^A(S^2(sub, \overline{0}, A\Rightarrow)) \bullet AS^B(S^2(sub, \overline{0}, B\Rightarrow)),$
$id)) \bullet$
$\bullet\ AS^R(\overline{0}) \bullet$
$\bullet\ WH(S^2(gr, B\Rightarrow, \overline{0}), AS^R(sem\_A(R + A)) \bullet AS^B(sem\_A(B - 1)) =$

$= AS^R(\overline{0}) \bullet$
$\bullet\ IF(S^2(less, B\Rightarrow, \overline{0}), AS^A(S^2(sub, \overline{0}, A\Rightarrow)) \bullet AS^B(S^2(sub, \overline{0}, B\Rightarrow)),$
$id)) \bullet$
$\bullet\ WH(S^2(gr, B\Rightarrow, \overline{0}), AS^R(S^2(add, sem\_A(R), sem\_A(A))) \bullet AS^B(S^2$
$(sub, sem\_A(B), sem\_A(1)))) =$

$= IF(S^2(less, B\Rightarrow, \overline{0}), AS^A(S^2(sub, \overline{0}, A\Rightarrow)) \bullet AS^B(S^2(sub, \overline{0}, B\Rightarrow)),$
$id)) \bullet$
$\bullet\ AS^R(\overline{0}) \bullet$
$\bullet\ WH(S^2(gr, B\Rightarrow, \overline{0}), AS^R(S^2(add, R\Rightarrow, A\Rightarrow)) \bullet AS^B(S^2(sub, B\Rightarrow, \overline{1})))$

3) Доведення часткової коректності.

Позначимо

$F_1 = IF(S^2(less, B\Rightarrow, \overline{0}), AS^A(S^2(sub, \overline{0}, A\Rightarrow)) \bullet AS^B(S^2(sub, \overline{0}, B\Rightarrow)),$
$id)),$

$F_2 = AS^R(\overline{0}),$

$F_3 = WH(S^2(gr, B\Rightarrow, \overline{0}),$
$AS^R(S^2(add, R\Rightarrow, A\Rightarrow)) \bullet AS^B(S^2(sub, B\Rightarrow, \overline{1})))$

Тоді вся програма задається виразом $F = F_1 \bullet F_2 \bullet F_3$

Нехай маємо дані $d = [A \mapsto a, B \mapsto b]$. Доведемо, що на них програма працює правильно, тобто при завершенні обчислень значення R дорівнює $a * b$.

За означенням композиції • маємо:

$F(d) = (F_1 \bullet F_2 \bullet F_3)(d) = F_3(F_2(F_1(d)))$

Обчислимо F_1 на d.

$F_1(d) = IF(S^2(less, B\Rightarrow, \overline{0}), AS^A(S^2(sub, \overline{0}, A\Rightarrow)) \bullet AS^B(S^2(sub, \overline{0}, B\Rightarrow)), id))(d)$

Розглянемо два випадки:

1) b додатне або дорівнює нулю.

Обчислимо умову:

$S^2(less, B\Rightarrow, \overline{0})(d_1) = less(B\Rightarrow(d_1), \overline{0}\ (d_1)) = less(b, \overline{0}) = false$

Оскільки вона хибна, то результатом буде $id(d) = d$, тобто значення змінних не змінюються.

2) b від'ємне. Тоді умова істинна й результат обчислюється як

$AS^A(S^2(sub, \overline{0}, A\Rightarrow)) \bullet AS^B(S^2(sub, \overline{0}, B\Rightarrow)(d) =$

$= AS^A(S^2(sub, \overline{0}, A\Rightarrow))(AS^B(S^2(sub, \overline{0}, B\Rightarrow)(d)) =$

$= AS^A(S^2(sub, \overline{0}, A\Rightarrow))(d \nabla [B \mapsto -b]) =$

$= AS^A(S^2(sub, \overline{0}, A\Rightarrow))([A \mapsto a, B \mapsto -b]) =$

$= [A \mapsto a, B \mapsto -b] \nabla [A \mapsto -a] =$

$= [A \mapsto -a, B \mapsto -b]$

Позначимо новий стан через $d_1 = [A \mapsto -a, B \mapsto -b]$. Тоді

$F(d) = (F_1 \bullet F_2 \bullet F_3)(d) = F_3(F_2(d_1))$

Обчислимо F_2:

$F_2(d) = AS^R(\overline{0})\ (d) = AS^R(\overline{0})\ ([A \mapsto a, B \mapsto -b]) =$

$= [A \mapsto a, B \mapsto -b] \nabla [R \mapsto 0] =$

$= [A \mapsto a, B \mapsto -b, R \mapsto 0] = d_2$

Позначимо новий стан через $d_2 = [A \mapsto a_0, B \mapsto b_0, R \mapsto 0]$, де a_0, b_0 одночасно або додатні, або від'ємні. Далі скористаємося властивістю множення: $a * b = -a * -b$. Таким чином, $a_0 * b_0 = a * b$, причому b_0 завжди додатне або дорівнює нулю.

Обчислимо цикл $F_3(d_2)$.

Гіпотеза: при виконанні n кроків тіла циклу стан змінних $s_n = [R \mapsto a_0 * n, A \mapsto a_0, B \mapsto b_0 - n]$. Доведення будемо виконувати індукцією за кроками n.

База індукції: $n = 0$, тобто тіло циклу не виконується. Маємо $s_0 = [R \mapsto a_0 * 0, A \mapsto a_0, B \mapsto b_0 - 0] = [R \mapsto 0, A \mapsto a_0, B \mapsto b_0] = d_2$. Гіпотеза справджується.

Припущення. Припустимо, що виконано t–1 кроків тіла циклу, дані мають вигляд $s_{t-1} = [R \mapsto a_0 * (t-1), A \mapsto a_0, B \mapsto b_0 - (t-1)]$. Покажемо, як зміняться дані після виконання ще одного кроку циклу $F_3(s_{t-1})$. Якщо умова істинна, то обчислюємо тіло циклу:

$$AS^R(S^2(add, R\Rightarrow, A\Rightarrow)) \bullet AS^B(S^2(sub, B\Rightarrow, \overline{1}))(s_{t-1}) =$$
$$= AS^R(S^2(add, R\Rightarrow, A\Rightarrow))(AS^B(S^2(sub, B\Rightarrow, \overline{1}))(s_{t-1})) =$$
$$= AS^R(S^2(add, R\Rightarrow, A\Rightarrow))(s_{t-1} \nabla [B \mapsto sub(B\Rightarrow(s_{t-1}), \overline{1}\ (s_{t-1}))]) =$$
$$= AS^R(S^2(add, R\Rightarrow, A\Rightarrow))(s_{t-1} \nabla [B \mapsto b_0 - (t-1) + 1]) =$$
$$= AS^R(S^2(add, R\Rightarrow, A\Rightarrow))([R \mapsto a_0 * (t-1), A \mapsto a_0, B \mapsto b_0 - t)]) =$$
$$= [R \mapsto a_0 * (t-1), A \mapsto a_0, B \mapsto b_0 - t)] \nabla [R \mapsto a_0 * (t-1) + a_0] =$$
$$= [R \mapsto a_0 * t, A \mapsto a_0, B \mapsto b_0 - t)] = s_t$$

Якщо умова хибна, то обчислень більше не виконуємо.

Отже, гіпотезу доведено.

Якщо $t = b_0$ то умова циклу стає хибною, цикл завершується та $R = a_0 * b_0$.

Обґрунтування повної коректності.

Покажемо, що при правильних вхідних даних програма завершується (зупиняється). Фактично треба показати завершеність циклу F_3, оскільки інші оператори не впливають на завершення програми.

З часткової коректності видно, що за правильних вхідних даних перед виконанням циклу дані мають вигляд $[R \mapsto 0, A \mapsto a_0, B \mapsto b_0]$, причому b_0 є невід'ємним.

Якщо b_0 строго більше нуля, то виконується цикл. Після кожної ітерації циклу B зменшується на 1. Отже, отримаємо спадну послідовність значень B. Коли B буде дорівнювати нулю, умова циклу стане хибною і він завершиться.

Якщо b_0 дорівнює нулю, то програма не входить у цикл, а послідовно виконує прості оператори, тому вона завжди зупиняється внаслідок скінченності кількості операторів.

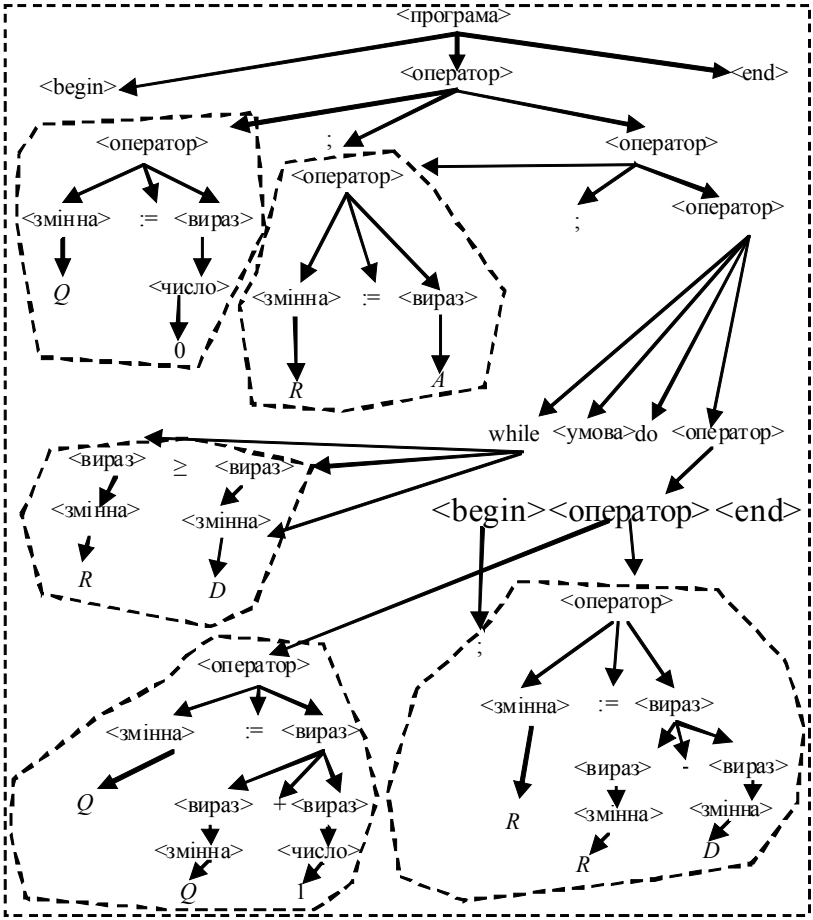
Завдання 1.4. За допомогою операцій віднімання й додавання розробити програму, яка обчислює результат ділення невід'ємного числа a на додатне число d та остачу від цього ділення. Нагадаємо, що результатом і остачею від ділення a на d називаються такі числа q та r, для яких виконується рівність $a = q * d + r$, $0 \le r < d$.

Виконати:
1) побудувати програму для розв'язання задачі мовою SIPL;
2) побудувати дерево синтаксичного виведення програми;
3) побудувати композиційний семантичний терм програми;
4) довести коректність програми.

Розв'язання.
1) Програма для розв'язання задачі:

```
begin
    Q := 0;
    R := A;
    while R ≥ D do
    begin
            Q := Q + 1;
            R := R – D
    end
end
```

Дерево синтаксичного виведення програми:



3) Семантичний композиційний терм:

$sem\_P(\text{Program}) =$

$= sem\_P(\text{begin}$

$Q := 0;$

$R := A;$

while R D do

begin
$Q := Q + 1;$
$R := R - D$
end
end
)=
-- застосовуємо правило $sem\_P(\text{begin } S \text{ end}) = sem\_S(S)$
$= sem\_S($ $Q := 0;$
$R := A;$
while $R \geq D$ do
begin
$Q := Q + 1;$
$R := R - D$
end
)=
-- двічі застосовуємо правило $sem\_S(S_1;S_2) = sem\_S(S_1) \bullet sem\_S(S_2)$
$= sem\_S(Q := 0) \bullet$
$\bullet\ sem\_S(R := A) \bullet$
$\bullet\ sem\_S$ (while $R \geq D$ do
begin
$Q := Q + 1;$
$R := R - D$
end
)=

Розпишемо перший оператор присвоювання:
$sem\_S(Q := 0) =$
-- застосовуємо правило $sem\_S(x := a) = AS^x(sem\_A(a))$
$= AS^Q(sem\_A(0)) =$
-- далі застосовуємо правило $sem\_A(n)) = \overline{n}$
$= AS^Q(\overline{0})$

Аналогічно розпишемо другий оператор присвоювання, але скористаємося правилом $sem\_A(x)) = x\Rightarrow$ замість $sem\_A(n)) = \overline{n}$:
$sem\_S(R := A) = AS^R(sem\_A(A)) =$
$= AS^R(A\Rightarrow)$

Обчислимо семантику оператора циклування:

sem_S (while $R \geq D$ do

begin

$Q := Q + 1;$

$R := R - D$

end

) =

-- згідно з правилом $sem\_S(\text{while } b \text{ do } S) = WH(sem\_B(b), sem\_S(S))$ маємо:

$= WH($

$sem\_B(R \geq D),$

$sem_S($begin

$Q := Q + 1;$

$R := R - D$

end

)

) =

Обчислимо семантику умови циклу. Для цього застосуємо правило

$sem\_B(a_1 \geq a_2) = S^2(gre, sem\_A(a_1), sem\_A(a_2))$.

$sem\_B(R \geq D) = S^2(gre, sem\_A(R), sem\_A(D)) =$

-- застосовуємо двічі правило $sem\_A(x)) = x \Rightarrow$

$= S^2(gre, R \Rightarrow, D \Rightarrow)$

Далі обчислимо семантику тіла циклу:

sem_S(begin

$Q := Q + 1;$

$R := R - D$

end) =

-- згідно з правилом $sem\_S(begin\ S\ end) = (sem\_S(S)) =$

$= sem\_S(Q := Q + 1;\ R := R - D) =$

-- застосовуємо правило $sem\_S(S_1;S_2) = sem\_S(S_1) \bullet sem\_S(S_2) =$

$= sem\_S(Q := Q + 1) \bullet sem\_S(R := R - D)$

-- до кожного з операторів присвоювання застосовуємо правило $sem\_S(x := a) = AS^x(sem\_A(a))$

$= AS^Q(sem\_A(Q + 1)) \bullet AS^R(sem\_A(R - D)) =$

-- застосовуємо правило sem_A(a1+a2) = S2(add, sem_A(a1), sem_A(a2)) до першого оператора присвоювання

$= AS^{Q}(S^{2}(add, sem\_A(Q), sem\_A(1))) \bullet AS^{R}(sem\_A(R - D)) =$

-- використовуємо правила $sem\_A(x)) = x\Rightarrow$ та $sem\_A(n)) = \overline{n}$

$= AS^{Q}(S^{2}(add, Q\Rightarrow, \overline{1})) \bullet AS^{R}(sem\_A(R - D))$

Аналогічно обчислюємо семантику другого оператора присвоювання та отримуємо семантику тіла циклу:

$AS^{Q}(S^{2}(add, Q\Rightarrow, \overline{1})) \bullet AS^{R}(S^{2}(\text{sub}, R\Rightarrow, D\Rightarrow))$

Збираючи все разом, отримуємо семантику програми:

$AS^{Q}(\overline{0}) \bullet$

$\bullet\ AS^{R}(A\Rightarrow) \bullet$

$\bullet\ WH(S^{2}(gre, R\Rightarrow, D\Rightarrow),$

$\quad AS^{Q}(S^{2}(add, Q\Rightarrow, \overline{1})) \bullet$

$\quad \bullet\ AS^{R}(S^{2}(\text{sub}, R\Rightarrow, D\Rightarrow))$

)

4) Доведення часткової коректності.

Програма отримує на вхід дані $[A \mapsto a,\ D \mapsto d]$, $a \geq 0,\ d > 0$, та повертає $[A \mapsto a,\ D \mapsto d,\ Q \mapsto q,\ R \mapsto r]$, де q – результат ділення a на d, r – остача, $0 \leq r < d$.

Після виконання перших двох операторів присвоювання отримуємо стан $st_0 = [A \mapsto a, D \mapsto d, Q \mapsto 0, R \mapsto a]$. Пропонуємо довести це самостійно.

Гіпотеза: після виконання i-го кроку циклу дані матимуть вигляд $[A \mapsto a, D \mapsto d, Q \mapsto i, R \mapsto a - d * i]$, причому $R \geq 0$.

Доведемо це методом математичної індукції.

База індукції. Нехай $i = 0$, тобто цикл не виконувався жодного разу. Тоді $[A \mapsto a,\ D \mapsto d,\ Q \mapsto i,\ R \mapsto a - d * i] = [A \mapsto a,\ D \mapsto d,\ Q \mapsto 0,\ R \mapsto a]$. Дані збігаються з початковими даними циклу st_0, R буде невід'ємним за умовою коректності вхідних даних. Гіпотеза виконується.

Крок індукції. Нехай гіпотеза має місце для $i = k$. Це означає, що після виконання k-ї ітерації стан st_k дорівнює $[A \mapsto a,\ D \mapsto d,\ Q \mapsto k,\ R \mapsto a - d * k]$. Позначимо $r_k = a - d * k$. Згідно з гіпотезою $r_k \geq 0$.

Нехай умова циклу істинна, тобто виконується $k + 1$ крок. Обчислимо тіло циклу на цьому кроці:

$(AS^Q(S^2(add, Q\Rightarrow, \overline{1})) \bullet AS^R(S^2(sub, R\Rightarrow, D\Rightarrow)))(st_k) =$

-- оскільки $f_1 \bullet f_2(st) = f_2(f_1(st))$

$= AS^R(S^2(\text{sub}, R\Rightarrow, D\Rightarrow))(AS^Q(S^2(add, Q\Rightarrow, \overline{1}))(st_k))$

Обчислимо перше присвоювання:

$AS^Q(S^2(add, Q\Rightarrow, \overline{1}))(st_k) =$

$= st_k \nabla [Q \mapsto S^2(add, Q\Rightarrow, \overline{1}))(st_k)] =$

$= st_k \nabla [Q \mapsto add(Q\Rightarrow(st_k), \overline{1}\ (st_k))] =$

$= st_k \nabla [Q \mapsto add(k, 1)] =$

$= st_k \nabla [Q \mapsto k + 1] =$

$= [A \mapsto a, D \mapsto d, Q \mapsto k + 1, R \mapsto a - d * k] = sti$

На отриманому стані обчислимо друге присвоювання:

$AS^R(S^2(\text{sub}, R\Rightarrow, D\Rightarrow))(sti) =$

$= sti \nabla [\text{R} \mapsto S^2(sub, R\Rightarrow, D\Rightarrow)(sti)] =$

$= sti \nabla [\text{R} \mapsto sub(R\Rightarrow(sti), D\Rightarrow(sti))] =$

$= sti \nabla [\text{R} \mapsto sub(a - d * k, d)] =$

$= sti \nabla [\text{R} \mapsto a - d * (k + 1)] =$

$= [A \mapsto a, D \mapsto d, Q \mapsto k + 1, R \mapsto a - d * (k + 1)] = st_{k+1}$

Позначимо $r_{k+1} = a - d * (k + 1)$. Тоді $r_{k+1} = r_k - d$. Оскільки $r_k \geq d$, то $r_{k+1} \geq 0$.

Отже, гіпотезу доведено.

Нехай умова циклу хибна після k-го кроку, тобто $r_k < d$. Тоді програма закінчує роботу в стані $st_k = [A \mapsto a,\ D \mapsto d,\ Q \mapsto q_k, R \mapsto r_k]$, де $q_k = k$, $r_k = a - d * k$, причому $0 \le r_k < d$.

Розрахуємо значення виразу $q_k * d + r_k = k * d + (a - d * k) = a$.

Згідно з формулою, наведеною в умові, це означає, що q_k є результатом ділення a на d, а r_k – остачею.

Ми показали часткову коректність програми при правильних вхідних даних, коли $a \ge 0$ та $d > 0$.

Обґрунтування повної коректності.

Покажемо, що при правильних вхідних даних програма завершується (зупиняється). Фактично треба показати завершеність циклу, оскільки інші оператори не впливають на завершення програми.

З часткової коректності видно, що за правильних вхідних даних перед виконанням циклу дані мають вигляд $st_0 = [A \mapsto a,\ D \mapsto d,\ Q \mapsto 0,\ R \mapsto a]$, де $a \ge 0$, $d > 0$. Оскільки $d > 0$, то маємо спадний ряд $r_0, r_1, r_2 \ldots$. Тому умова циклу $R \ge D$ стане хибною після скінченної кількості ітерацій і цикл завершиться. Отже, якщо програма входить у цикл, то він завжди завершується.

За інших варіантів правильних вхідних даних програма не входить у цикл, а послідовно виконує прості оператори, тому вона завжди зупиняється внаслідок скінченності кількості операторів.

Завдання 1.5. Розробити програму, яка обчислює значення $n!$ за вхідним цілим n з використанням одного циклу.

Факторіали визначені тільки для невід'ємних цілих чисел. Факторіал від нуля дорівнює одиниці. У програмі розширимо означення факторіала, а саме покладемо його рівним нулю для від'ємних чисел. Таким чином буде побудована функція, визначена на цілих числах.

Виконати:

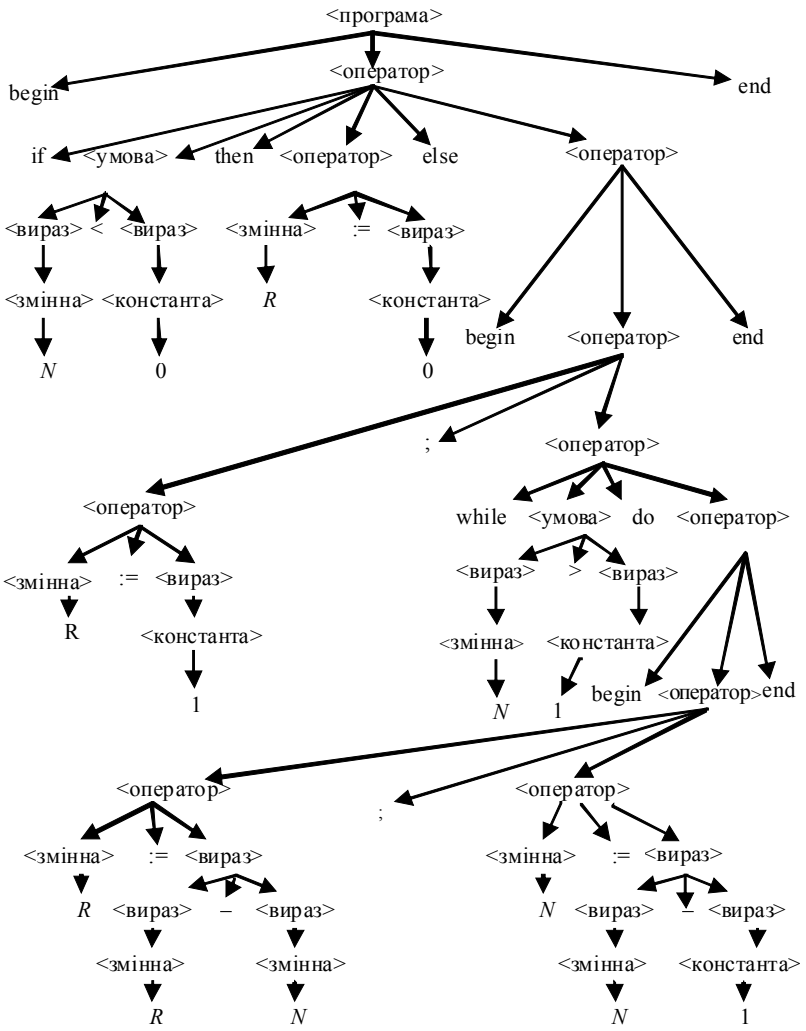
1) побудувати програму для розв'язання задачі мовою SIPL;
2) побудувати дерево синтаксичного виведення програми;
3) побудувати композиційний семантичний терм програми;
4) довести коректність програми.

Розв'язання.

1) Програма для розв'язання задачі:

```
begin
    If N < 0 then
        R := 0
    else
    begin
        R := 1;
        while N > 1 do
        begin
            R := R * N;
            N := N – 1
        end
    end
end
```

2) Дерево синтаксичного виведення програми:



3) Семантичний композиційний терм:

sem_P(Program) =

= *sem_P*(begin

if $N < 0$ then

$R := 0$

else

begin

$R := 1$;

while $N > 1$ do

begin

$R := R * N$;

$N := N - 1$

end

end

end)

Застосовуємо правило *sem_P*(begin *S* end) = *sem_S*(*S*):

sem_P(if $N < 0$ then

$R := 0$

else

begin

$R := 1$;

while $N > 1$ do

begin

$R := R * N$;

$N := N - 1$

end

end)

Застосовуємо правило *sem_S*(if *b* then S_1 else S_2) =

= *IF*(*sem_B*(*b*), *sem_S*(S_1), *sem_S*(S_2)):

IF(*sem_B*($N<0$), *sem_S*($R := 0$), *sem_S*($R := 1$; while $N > 1$ do begin $R := R*N$; $N := N-1$ end))

Обчислимо умову оператора розгалуження:

$sem\_B(N<0) = S^2(less, sem\_A(N), sem\_A(0)) = S^2(less, N\Rightarrow, \underline{0})$

Далі обчислимо оператор присвоювання $R:=0$, який міститься в першій гілці оператора розгалуження:

$sem\_S(R:=0) = AS^R(sem\_A(0)) = AS^R(\underline{0})$

Перейдемо до обчислення другої гілки оператора розгалуження:

$sem\_S(R:=1;$ while $N>1$ do begin $R:=R*N;\ N:=N-1$ end)

Згідно з правилом $sem\_S(S_1;S_2) = sem\_S(S_1) \bullet sem\_S(S_2)$ маємо:

$sem_S(R:=1) \bullet sem_S($while $N>1$ do begin $R:=R*N;\ N:=N-1$ *end*)

Обчислимо перший оператор присвоювання:

$sem\_S(R:=1) = AS^R(sem\_A(1)) = AS^R(\underline{1})$

Далі обчислимо семантику оператора циклу, застосовуючи правило $sem_S($while b do $S) = WH(sem\_B(b), sem\_S(S))$:

$sem\_S($ while $N>1$ do begin $R:=R*N;\ N:=N-1$ end) =

$= WH(sem_B(N>1), sem_S($begin $R:=R*N;\ N:=N-1$ end)) =$

Неважко перевірити, що семантикою умови циклу $N>1$ буде функція $S^2(gr, N\Rightarrow, \underline{1})$.

Обчислимо семантику тіла циклу. Згідно з правилом $sem_P($begin S end$) = sem_S(S)$ маємо:

$sem_S($begin $R:=R*N;\ N:=N-1$ end$) = sem_S(R:=$ R$*N;\ N:=N-1)$

Далі, згідно з правилом $sem\_S(S_1;S_2) = sem\_S(S_1) \bullet sem\_S(S_2)$, отримуємо

$sem\_S(R:=R*N) \bullet sem\_S(N:=N-1)$

Обчислимо семантику першого оператора присвоювання:

$sem\_S(R:=R*N)$

Застосовуємо правило $sem\_S(x:=a) = AS^x(sem\_A(a))$:

$AS^R(sem\_A(R*N))$

Далі, згідно з правилом $sem\_A(a_1*a_2) = S^2(mult,\ sem\_A(a_1), sem\_A(a_2))$, маємо:

$AS^R(S^2(mult, sem\_A(R), sem\_A(N)))$

Нарешті, застосовуючи правило $sem\_A(x) = x\Rightarrow$, отримуємо семантичну функцію

$AS^R(S^2(mult, R\Rightarrow, N\Rightarrow))$

Таким саме чином обчислимо семантику другого оператора присвоювання:

$sem\_S(N{:=}N{-}1) = AS^N(S^2(sub, N\Rightarrow, \bar{1}))$

Збираючи все разом, отримаємо семантичну функцію всієї програми:

$IF($

$\quad S^2(less, N\Rightarrow, \bar{0}),$

$\quad AS^R(\bar{0}),$

$\quad AS^R(\bar{1}) \bullet WH(S^2(gr, N\Rightarrow, \bar{1}), AS^R(S^2(mult, R\Rightarrow,$

$N\Rightarrow)) \bullet AS^N(S^2(sub, N\Rightarrow, \bar{1})))$

$)$

4) Доведення часткової коректності.

Доведемо, що на вхідних даних $d = [N\mapsto n]$, де n – ціле число, програма працює правильно, а саме: при завершенні обчислень отримуємо результат $[N\mapsto k, R\mapsto r]$, де $k\le 1$, а

$$r = \begin{cases} 0, \text{ якщо } n < 0, \\ n!\ \textit{в іншому випадку}. \end{cases}$$

Введемо такі позначення:

$F_1 = AS^R(\bar{0})$

$F_{21} = AS^R(\bar{1})$

$F_{22} = WH(S^2(gr, N\Rightarrow, \bar{1}), AS^R(S^2(mult, R\Rightarrow, N\Rightarrow)) \bullet AS^N(S^2(sub,$

$N\Rightarrow, \bar{1})))$

Тоді семантичний терм програми записуємо як

$F = IF(\mathrm{S}^2(less, N\Rightarrow, \bar{0}), F_1, F_{21} \bullet F_{22})$

1) Нехай $n<0$. Тоді умова $S^2(less, N\Rightarrow, \bar{0})$ істина й виконується функція F_1.

$F_1([N\mapsto n]) = AS^R(\bar{0})([N\mapsto n])$

Обчислюємо за формулою $AS^{\alpha}(fa)(st) = st\,\nabla[x\mapsto fa(st)]$

$[N\mapsto n]\,\nabla\,[R\mapsto \bar{0}([N\mapsto n])] = [N\mapsto n, R\mapsto 0]$

Програма обчислюється коректно.

2) Нехай $0\le n\le 1$. Тоді умова $\mathrm{S}^2(less, N\Rightarrow, \bar{0})$ хибна й виконується функція $F_{21} \bullet F_{22}$, яку обчислюємо за формулою $(fs_1 \bullet fs_2)(st) = fs_2(fs_1(st))$:

$F_{21} \bullet F_{22}([N\mapsto n]) = F_{22}(F_{21}([N\mapsto n]))$

Спочатку обчислимо F_{21}:

$F_{21}([N\mapsto n]) = AS^R(\bar{1})([N\mapsto n]) = [N\mapsto n]$

$\nabla\,[R\mapsto \bar{1}([N\mapsto n])] = [N\mapsto n, R\mapsto 1]$

Щодо обчислення циклу F_{22}, то умова циклу хибна на цих вхідних даних, тобто цикл не виконується жодного разу. Таким чином, факторіал дорівнює 1 для $n = 0,1$, програма працює коректно.

3) Нехай $n>1$. Від попереднього цей випадок відрізняється тим, що умова циклу буде істинною, тобто цикл виконується принаймні один раз. Покажемо, як змінюється стан програми в процесі виконання циклу.

Гіпотеза: після виконання i-го кроку циклу дані матимуть вигляд $[N\mapsto n-i, R\mapsto \prod_{j=n-i+1}^{n} j\,]$.

Доведемо це методом математичної індукції.

Позначимо початковий стан як $st_0 = [N\mapsto n, R\mapsto 1]$.

База індукції. Нехай $i = 1$, тобто цикл виконався один раз. Обчислимо стан програми після першої ітерації, тобто застосуємо тіло циклу до початкового стану st_0.

$$AS^R(S^2(mult, R\Rightarrow, N\Rightarrow)) \bullet AS^N(S^2(sub, N\Rightarrow, \bar{1}))(st_0) =$$

$$= AS^N(S^2(sub, N\Rightarrow, \bar{1}))(AS^R(S^2(mult, R\Rightarrow, N\Rightarrow))(st_0))$$

Обчислимо перший оператор присвоювання:

$$AS^R(S^2(mult, R\Rightarrow, N\Rightarrow))(st_0) =$$

$$= st_0 \nabla [R\mapsto S^2(mult, R\Rightarrow, N\Rightarrow)(st_0)] =$$

$$= st_0 \nabla [R\mapsto mult(R\Rightarrow(st_0), N\Rightarrow(st_0))] =$$

$$= st_0 \nabla [R\mapsto mult(1, n)] =$$

$$= [N\mapsto n, R\mapsto 1] \nabla [R\mapsto n] =$$

$$= [N\mapsto n, R\mapsto n] = st_0'$$

На отриманому стані st_0' обчислимо другий оператор присвоювання:

$$AS^N(S^2(sub, N\Rightarrow, \bar{1}))(st_0') =$$

$$= st_0' \nabla [N\mapsto S^2(sub, N\Rightarrow, \bar{1})(st_0')] =$$

$$= st_0' \nabla [N\mapsto sub(N\Rightarrow(st_0'), \bar{1}(st_0'))] =$$

$$= st_0' \nabla [N\mapsto sub(n, 1)] =$$

$$= [N\mapsto n, R\mapsto n] \nabla [N\mapsto n-1)] =$$

$$= [N\mapsto n-1, R\mapsto n] =$$

$$= [N\mapsto n-1, R\mapsto \prod_{j=n-1+1}^{n} j\ ]$$

Отже, для $i = 1$ гіпотеза виконується.

Крок індукції. Нехай гіпотеза має місце для $i = k$. Це означає, що після виконання k-ї ітерації стан st_k дорівнює $[N \mapsto n-k,\ R \mapsto \prod_{j=n-k+1}^{n} j\ ]$. Нехай умова циклу істинна, тобто виконується $k+1$ крок. Обчислимо тіло циклу на цьому кроці.

$$AS^R(S^2(mult, R\Rightarrow, N\Rightarrow)) \bullet AS^N(S^2(sub, N\Rightarrow, \bar{1}))(st_k) =$$

$$= AS^N(S^2(sub, N\Rightarrow, \bar{1}))(\ AS^R(S^2(mult, R\Rightarrow, N\Rightarrow))(st_k))$$

Обчислимо перший оператор присвоювання:

$$AS^R(S^2(mult, R\Rightarrow, N\Rightarrow))(st_k) =$$

$$= st_k \nabla [R \mapsto mult(R\Rightarrow(st_k), N\Rightarrow(st_k))] =$$

$$= st_k \nabla [R \mapsto (\ \prod_{j=n-k+1}^{n} j\ )*(\ n-k)] =$$

$$= st_k \nabla [R \mapsto \prod_{j=n-k}^{n} j\ ]$$

Додамо й віднімемо одиницю від нижнього індексу добутку:

$$= st_k \nabla [R \mapsto \prod_{j=n-(k+1)+1}^{n} j\ ] =$$

$$= [N \mapsto n-k,\ R \mapsto (\ \prod_{j=n-(k+1)+1}^{n} j\ )] = st_k^{'}$$

На отриманому стані $st_k^{'}$ обчислимо другий оператор присвоювання:

$$AS^N(S^2(sub, N\Rightarrow, \bar{1}))(\ st_k^{'}\ ) =$$

$$= st_k^{'} \nabla [N \mapsto S^2(sub, N\Rightarrow, \bar{1})(\ st_k^{'}\ )] =$$

$$= st_k^{'} \nabla [N \mapsto sub(N\Rightarrow(\ st_k^{'}\ ), \bar{1}(\ st_k^{'}\ ))] =$$

$$= st_k^{'} \nabla [N \mapsto n-k-1] =$$

$= [N \mapsto n-k,\ R \mapsto \prod_{j=n-(k+1)+1}^{n} j\ ] \nabla\ [N \mapsto n-k-1)] =$

$= [N \mapsto n-k-1,\ R \mapsto \prod_{j=n-(k+1)+1}^{n} j\ ] =$

$= [N \mapsto n-(k+1),\ R \mapsto \prod_{j=n-(k+1)+1}^{n} j\ ].$

Отже, гіпотезу доведено.

Після виконання $n-1$ кроку циклу стан програми буде

$st_{n-1} = [N \mapsto n-(n-1),\ R \mapsto \prod_{j=n-(n-1)+1}^{n} j\ ] =$

$= [N \mapsto 1,\ R \mapsto \prod_{j=2}^{n} j\ ] = [N \mapsto 1,\ R \mapsto n!].$

На цьому стані умова циклу стане хибною. На всіх попередніх станах умова буде істинною, тобто це перший стан, на якому умова стає хибною. Отже, цикл завершується на цьому стані. Результатом буде факторіал числа n.

Обґрунтування повної коректності.

Покажемо, що при правильних вхідних даних програма завершується (зупиняється). Фактично треба показати завершуваність циклу F_{22}, оскільки інші оператори не впливають на завершення програми. З часткової коректності видно, що за правильних вхідних даних перед виконанням циклу дані мають вигляд $[N \mapsto n, R \mapsto 1]$, де $n>1$. Після кожного виконання циклу N зменшується на 1. Отже, отримаємо спадну послідовність. Унаслідок її скінченності, якщо програма входить у цикл, то він завжди завершується. При інших варіантах правильних вхідних даних програма не заходить у цикл, а послідовно виконує прості оператори, тому вона завжди зупиняється внаслідок скінченності кількості операторів.

Завдання 1.6. Розробити програму, яка за допомогою операції додавання обчислює результат піднесення значення a до степеня b, де $a, b \geq 0$.

Виконати:

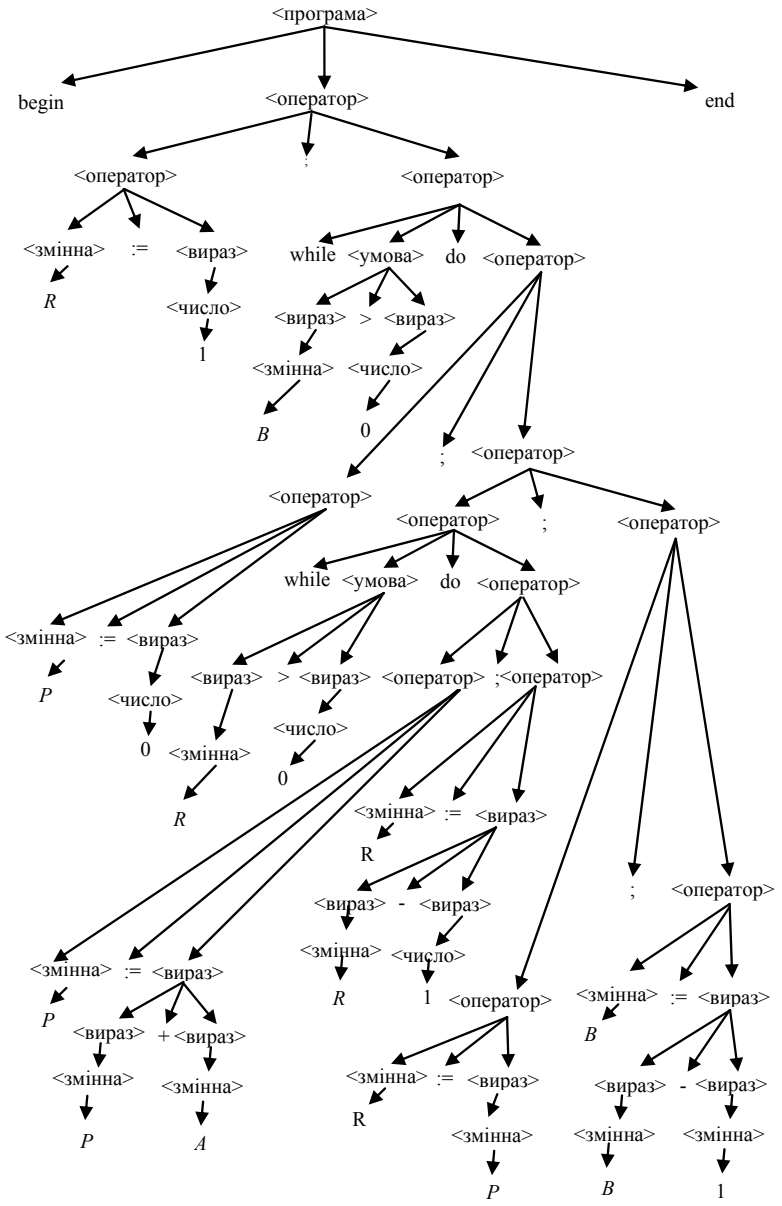
1) побудувати програму для розв'язання задачі мовою SIPL;
2) побудувати дерево синтаксичного виведення програми;
3) побудувати композиційний семантичний терм програми;
4) довести коректність програми.

Розв'язання.

1) Програма мовою SIPL, яка обчислює a^b через додавання й повертає результат r у змінній R. Використовується допоміжна змінна P:

```
begin
R := 1;
while B > 0 do
begin
    P := 0;
    //розраховуємо P = R*A
    while R > 0 do
    begin
        P := P + A;
R := R – 1
end;
R := P;
B := B – 1
end
end
```

2) Дерево синтаксичного виведення програми:



3) Семантичний терм програми.

Введемо позначення для групи операторів, які здійснюють множення через додавання:

$F_1(P, R, A) =$

$P := 0;$

while $R > 0$ do

begin

$P := P + A;$

$R := R - 1$

end

Цей код здійснює множення значень додатних змінних R та A, результат записується в змінну P. Такий саме код, з точністю до імен змінних, використовувався в завд. 1.3. Достатньо переіменувати змінну P в R, а змінну R у B. Це дозволяє використати результати, наведені в завд. 1.3.

Перейдемо до побудови семантичного терму програми:

sem_S(Program) =

$= sem\_S$(begin

$R := 1;$

while $B > 0$ do

begin

$F_1(P, R, A);$

$R := P;$

$B := B - 1$

end

end

)

Згідно з правилом sem_S(begin S end) = ($sem\_S(S)$)

$sem\_S(R := 1;$

while $B > 0$ do

begin

$F_1(P, R, A);$

$R := P;$

$B := B - 1$

end

)

Застосовуємо правило $sem\_S(S_1;S_2) = sem\_S(S_1) \bullet sem\_S(S_2)$:

$sem\_S(R := 1) \bullet$

$\bullet$ *sem_S*(while $B > 0$ do
begin
$F_1(P, R, A)$;
$R := P$;
$B := B - 1$
end)

)

Розрахуємо значення першого оператора:

$sem\_S(R := 1)$

Застосовуємо правило $sem\_S(x := a) = AS^{\alpha}(sem\_A(a))$:

$AS^{R}(sem\_A(1))$

Застосовуємо правило $sem\_A(n)) = \overline{n}$:

$AS^{R}(\overline{1})$

Перейдемо до обчислення другого терму:

sem_S(while $B > 0$ do
begin
$F_1(P, R, A)$;
$R := P$;
$B := B - 1$
end)

Згідно з правилом *sem_S*(while *b* do *S*) = *WH*(*sem_B*(*b*), *sem_S*(*S*)) отримуємо терм *WH*(*sem_B*($B > 0$), *sem_S*(F_1; $R := P$; $B := B - 1$)), де

sem_B($B > 0$)

Згідно з правилом $sem\_B(a_1 > a_2) = S^2(gr, sem\_A(a_1), sem\_A(a_2))$
$= S^2(gr, sem\_A(B), sem\_A(0))$

Згідно з правилами $sem\_A(x)) = x\Rightarrow$, та $sem\_A(n)) = \overline{n}$

$S^2(gr, B\Rightarrow, \overline{0})$

Обчислимо семантику тіла циклу:

$sem\_S(F_1(P, R, A); R := P; B := B - 1)$

Згідно з правилом $sem\_S(S_1;S_2) = sem\_S(S_1) \bullet sem\_S(S_2)$

$sem\_S(F_1(P, R, A)) \bullet sem\_S(R := P; B := B - 1)$

Семантика $F_1(P, R, A)$ з точністю до імен змінних була обчислена в завд. 1.3.

$F_1(P, R, A) = AS^P(\overline{0}) \bullet WH(S^2(gr, R\Rightarrow, \overline{0}), AS^P(S^2(add, P\Rightarrow, A\Rightarrow) \bullet AS^R(S^2(sub, R\Rightarrow, \overline{1})))$

Залишилося обчислити семантику останніх двох операторів присвоювання:

$sem\_S(R := P; B := B - 1)$

Згідно з правилом $sem\_S(S_1;S_2) = sem\_S(S_1) \bullet sem\_S(S_2)$

$sem\_S(R := P) \bullet sem\_S(B := B - 1)$

Згідно з правилом $sem\_S(x := a) = AS^x(sem\_A(a))$

$AS^R(sem\_A(P)) \bullet AS^B(sem\_A(B - 1))$

Згідно з правилами $sem\_A(x)) = x\Rightarrow$, $sem\_A(a_1+a_2) = S^2(add, sem\_A(a_1), sem\_A(a_2))$, $sem\_A(n)) = \overline{n}$

$AS^R(P\Rightarrow) \bullet AS^B(S^2(sub, B\Rightarrow, \overline{1}))$

Збираючи все до купи, отримуємо семантику всієї програми:

$AS^R(\overline{1}) \bullet$

$\bullet WH(S^2(gr, B\Rightarrow, \overline{0}),$

$AS^P(\overline{0}) \bullet$

$\bullet WH(S^2(gr, R\Rightarrow, \overline{0}), AS^P(S^2(add, P\Rightarrow, A\Rightarrow) \bullet AS^R(S^2(sub, R\Rightarrow, \overline{1}))) \bullet$

$\bullet AS^R(P\Rightarrow) \bullet$

$\bullet AS^B(S^2(sub, B\Rightarrow, \overline{1}))$

)

4) Доводимо коректність програми.

Часткова коректність.

Нехай маємо вхідний стан $st = [A \mapsto a, B \mapsto b]$, $a \geq 0$, $b \geq 0$. Покажемо, що якщо програма завершується, то у вихідному стані є змінна R, яка дорівнює a^b, а змінна B дорівнює нулю.

Розглянемо чотири випадки:

1) $b = 0$ та $a = 0$, тоді $r = 1$;
2) $b = 0$ та $a > 0$, тоді $r = 1$;
3) $b > 0$ та $a = 0$, тоді $r = 0$;
4) $b > 0$ та $a > 0$, тоді $r = a * a * ... * a$, де a беремо b разів.

Розглянемо спочатку перший і другий випадки. Для зручності позначимо тіло першого циклу через G_1. Тоді семантичний терм програми матиме вигляд $AS^R(\overline{1}) \bullet WH(S^2(gr, B\Rightarrow, \overline{0}), G_1)$. Обчислимо його на стані $st = [A\mapsto 0, B\mapsto 0]$:

$AS^R(\overline{1}) \bullet WH(S^2(gr, B\Rightarrow, \overline{0}), G_1)(st) = WH(S^2(gr, B\Rightarrow, \overline{0}), G_1)(AS^R(\overline{1})(st))$.

Спершу обчислимо $AS^R(\overline{1})(st)$:

$AS^R(\overline{1})(st) = st \nabla [R\mapsto \overline{1}(st)] =$

$= st \nabla [R\mapsto 1] = [A\mapsto 0, B\mapsto 0, R\mapsto 1]$.

Позначимо новий стан через st_0. Обчислимо на ньому умову циклу:

$S^2(gr, B\Rightarrow, \overline{0})(st_0) = gr(B\Rightarrow(st_0), \overline{0}(st_0)) = gr(0, 0) = false.$

Умова хибна, значенням змінної R буде 1. Аналогічно перевіряємо другий випадок.

Розглянемо третій і четвертий випадки. Нехай $st = [A\mapsto a, B\mapsto b]$, $a \geq 0$, $b > 0$. Після виконання першого оператора отримаємо стан $st_0 = [A\to a, B\to b, R\mapsto 1]$. На ньому умова циклу буде істинною, тобто цикл виконується принаймні один раз.

Методом математичної індукції доведемо, що після виконання k кроків циклу, $k \leq b$, значення змінної R дорівнюватиме a^k, а значення B буде дорівнювати $b - k$.

База індукції: виконаємо одну ітерацію циклу.

Терм $F_1(P, R, A)$, який з точністю до назв змінних був розглянутий у завд. 1.3, на стані $[R\mapsto r, A\mapsto a]$, де r та a – невід'ємні числа, розраховує новий стан $[P\mapsto r * a, R\mapsto 0, A\mapsto a]$, тобто виконує

множення r на a. Звернемо увагу, що значення змінної R після обчислення дорівнює нулю.

Таким чином, на стані st_0 маємо:

$F_1(P,\ R,\ A)(st_0) = [A \mapsto a,\ B \mapsto b,\ R \mapsto 0,\ P \mapsto\ r * a\ ] = [A \mapsto a,\ B \mapsto b,\ R \mapsto 0,\ P \mapsto a^1\ ]$.

Позначимо цей стан через st_{01}. Перейдемо до обчислення останніх двох операторів:

$AS^R(P\Rightarrow) \bullet AS^B(S^2(sub,\ B\Rightarrow,\ \overline{1}\ ))(st_{01}) =$

$= AS^B(S^2(sub,\ B\Rightarrow,\ \overline{1}\ ))(AS^R(P\Rightarrow)\ (st_{01}))$.

Спочатку обчислимо оператор $AS^R(P\Rightarrow)$:

$AS^R(P\Rightarrow)\ (st_{01}) = [A \mapsto a,\ B \mapsto b,\ R \mapsto 0,\ \mathrm{P} \mapsto a^1\ ] \nabla\ [\mathrm{R} \mapsto P\Rightarrow(st_{01})] =$

$= [A \mapsto a,\ B \mapsto b,\ R \mapsto a^1,\ \mathrm{P} \mapsto a^1\ ]$.

Позначимо цей стан через st_{02}. Тепер обчислимо другий оператор $AS^B(S^2(sub,\ B\Rightarrow,\ \overline{1}\ ))$:

$AS^B(S^2(sub,\ B\Rightarrow,\ \overline{1}\ ))(st_{02}) = st_{02} \nabla\ [B \mapsto S^2(sub,\ B\Rightarrow,\ \overline{1}\ )(st_{02}) =$

$= st_{02} \nabla\ [B \mapsto b - 1] = [A \mapsto a,\ B \mapsto b - 1,\ R \mapsto a^1,\ \mathrm{P} \mapsto a^1\ ]$.

На цьому обчислення програми закінчене. Програма завершилась коректно.

Крок індукції. Нехай після виконання i ітерацій, $i \geq 1$, маємо стан $st_i = [A \mapsto a,\ B \mapsto b - i,\ R \mapsto a^i,\ P \mapsto a^i\ ]$, причому змінні A та B невід'ємні.

Нехай $b - i > 0$, тобто умова циклу істинна. Тоді доведемо, що після виконання ще одної ітерації отримаємо стан

$st_{i+1} = [A \mapsto a,\ B \mapsto b - i - 1,\ R \mapsto a^{i+1},\ P \mapsto a^{i+1}\ ]$,

в якому значення A та B будуть невід'ємними.

Спочатку знову розрахуємо значення терму $F_1(P, R, A)(st_i)$. Отримаємо новий стан

$st_{i1} = [A \mapsto a, B \mapsto b - i, R \mapsto 0, P \mapsto a^{i+1}]$.

Далі розрахуємо значення оператора $(AS^R(P\Rightarrow)(st_{i1})$. Отримуємо стан

$st_{i2} = [A \mapsto a, B \mapsto b - i, R \mapsto a^{i+1}, P \mapsto a^{i+1}]$.

Нарешті розрахуємо останній оператор $AS^B(S^2(sub, B\Rightarrow, \overline{1}))$ (st_{i2}). Отримаємо стан $st_{i+1} = [A \mapsto a, B \mapsto b - i - 1, R \mapsto a^{i+1}, P \mapsto a^{i+1}]$. Оскільки за припущенням значення змінної $B = b - i$ було додатним, то нове значення $B = b - i - 1$ буде невід'ємним.

Доведення завершене.

Після виконання b ітерацій отримаємо стан st_b, в якому $R = a^b$, а змінна $B = 0$. Це означає, що на наступному кроці b+1 умова циклу стане хибною і він завершиться.

Таким чином, якщо цикл завершиться, то значенням змінної R буде a^b, що й треба було довести. Отже, твердження доведене і програма частково коректна.

Тотальна коректність. Покажемо, що обчислення за програмою завжди зупиняються.

Завершуваність внутрішнього циклу $WH(S^2(gr, R\Rightarrow, \overline{0}), AS^P(S^2(add, P\Rightarrow, A\Rightarrow) \bullet AS^R(S^2(sub, R\Rightarrow, \overline{1})))$ була доведена в завд. 1.3.

Що стосується зовнішнього циклу $WH(S^2(gr, B\Rightarrow, \overline{0}), \ldots)$, після кожної ітерації значення змінної B зменшується на одиницю. Отримуємо монотонно спадну послідовність значень B, причому перший елемент цієї послідовності є невід'ємним за умовою. Тому рано чи пізно значення B стане дорівнювати 0 і цикл завершиться.

Таким чином, з часткової коректності та факту зупинки циклів (внутрішнього й зовнішнього) випливає тотальна коректність програми, що розглядається.

1.6. Розширення мови SIPL

Розширимо мову SIPL, увівши явно булевий тип, булеві змінні та вирази.

Синтаксис розширеної мови SIPL можна задати за допомогою БНФ (табл. 1.4), доповнивши базову (див. п. 1.1) новими правилами й розширивши існуючі з урахуванням нових конструкцій.

Таблиця 1.4

Синтаксис розширеної мови SIPL

Ліва частина правила – метазмінна (дефінієндум)	Права частина правила (дефінієнс)	Ім'я правила
<програма> ::=	begin <оператор> end	*NP*1
<оператор> ::=	<змінна_чис>:=<вираз_чис> | <змінна_бул>:=<вираз_бул> | <оператор> ; <оператор> | if <вираз_бул> then <оператор> else <оператор> | while <вираз_бул> do <оператор> | begin <оператор> end | skip	*NS*1 *NS*2 *NS*3 *NS*4 *NS*5 *NS*6 *NS*7
<вираз_чис> :=	<число> | <змінна_чис> | <вираз_чис > + <вираз_чис > | <вираз_чис > – <вираз_чис > | <вираз_чис > * <вираз_чис > | <вираз_чис > ÷ <вираз_чис > | (<вираз_чис>)	*NA*1_1– *NA*1_7
<вираз_бул> ::=	<бул> |<змінна_бул>| <вираз_чис> < <вираз_чис>| <вираз_чис> ≤ <вираз_чис> | <вираз_чис> = <вираз_чис> | <вираз_чис> ≠ <вираз_чис> | <вираз_чис> > <вираз_чис> | <вираз_чис> ≥ <вираз_чис> | <вираз_бул> < <вираз_бул> |<вираз_бул> ≤ <вираз_бул> | <вираз_бул> = <вираз_бул> | <вираз_бул> ≠ <вираз_бул> | <вираз_бул> > <вираз_бул> | <вираз_бул> ≥ <вираз_бул> | <вираз_бул> ∨ <вираз_бул> | <вираз_бул> ∧ <вираз_бул> | ¬ <вираз_бул>| (<вираз_бул>)	*NA*2_1– *NA*2_17

Закінчення табл. 1.4

Ліва частина правила – метазмінна (дефінієндум)	Права частина правила (дефінієнс)	Ім'я правила
<змінна_чис> ::=	*M* | *N* | . . .	*NV*1–…
<змінна_бул> ::=	*M_B* | *N_B* | . . .	*NV*1–…
<число> ::=	. . . –1 | 0 | 1 | 2 | 3 | . . .	*NN*1–…
<бул> ::=	true |false	*NB*1-*NB*2

У даному випадку множину всіх виразів розділено на булеві та цілочисельні. Введемо позначення для всіх синтаксичних категорій і нові метазмінні, що вказують на представників цих категорій (табл. 1.5).

Таблиця 1.5

Введення синтаксичних категорій і метазмінних

Метазмінна	Синтаксична категорія	Нова метазмінна
<програма>	*Prog*	*P*
<оператор>	*Stm*	*S*
<вираз_чис>	*Aexp*	*a*
<вираз_бул>	*Bexp*	*c*
<змінна_чис>	*Var*	*x*
<змінна_бул>	*Var_B*	*y*
<число>	*Num*	*n*
<бул>	*Bool*	*b*

У наведених позначеннях БНФ мова SIPL набуває такого вигляду (табл. 1.6):

Таблиця 1.6

БНФ із врахуванням введених позначень

Ліва частина правила – метазмінна (дефінієндум)	Права частина правила (дефінієнс)	Ім'я правила
P ::=	begin *S* end	*NP*1
S ::=	*x*: = *a*| *y*: = *c*| *S*1; *S*2| if *c* then *S*1 else *S*2 | while *c* do *S* | begin *S* end | skip	*NS*1 *NS*2 *NS*3 *NS*4 *NS*5 *NS*6 *NS*7
a :=	*n* | *x* | *a* 1+ *a*2 | *a*1 – *a*2 | *a*1 * *a*2 | *a*1 ÷ *a*2 | (*a*)	*NA*1_1– *NA*1_7
c ::=	*b* |*y*| *a* 1< *a*2| *a*1 ≤ *a*2 | *a*1 = *a*2 | *a*1 ≠ *a*2 | *a*1 > *a*2 | *a*1 ≥ *a*2 | *c*1 < *c*2 |*c*1 ≤ *c*2 | *c*1 = *c*2 | *c*1 ≠ *c*2 | *c*1 > *c*2 | *c*1 ≥ *c*2 | *c*1 ∨ *c*2 | *c*1 ∧ *c*2 | ¬ *c*| (*c*)	*NA*2_1– *NA*2_17
x ::=	*M* | *N* | . . .	*NV*1–…
y ::=	*M_b* | *N_b* | . . .	*NV*1–…
n ::=	. . . –1 | 0 | 1 | 2 | 3 | . . .	*NN*1–…
b ::=	true |false	*NB*1-*NB*2

Далі будемо користуватися введеними позначеннями для запису структури програм та їх складових.

Визначимо типи даних та операції над ними. Для повноти опису повторимо типи, визначені раніше.

Базові типи даних – множини цілих чисел, булевих значень і змінних (імен):

- $Int = \{ \ldots, -2, -1, 0, 1, 2, \ldots \}$
- $Bool = \{true, false\}$
- $Var = \{X, Y, \ldots\}$
- $Var\_B = \{X\_B, Y\_B, \ldots\}$

Похідний тип – множина станів змінних (наборів іменованих значень, наборів змінних з їх значеннями):

$State = \{st \mid st \in (Var \cup Var\_B) \to Int \cup Bool, st(x) \in Int, st(y) \in Bool\}$

Приклади станів змінних: $[M \mapsto 8, N \mapsto 16]$,

$[M \mapsto 3, X \mapsto 4, Y \mapsto true, N \mapsto false]$.

Операції на множині Int. Символам $+$, $-$, $*$, $\div$ відповідають операції *add*, *sub*, *mult*, *div* (додавання, віднімання, множення, ділення). Це бінарні операції типу $Int^2 \to Int$.

Операції на множині Bool. Символам $\vee$, $\wedge$, $\neg$ відповідають операції *or*, *and*, *neg*. Це бінарні операції типу $Bool^2 \to Bool$ (диз'юнкція та кон'юнкція) та унарна операція типу $Bool \to Bool$ (заперечення).

У мові також є *операції змішаного типу*. Символам операцій порівняння $<$, $\leq$, $=$, $\neq$, $\geq$, $>$ відповідають операції *less*, *leq*, *eq*, *neq*, *geq*, *gr*. Це бінарні операції типу $Int^2 \to Bool$.

Уведемо також:

- функцію-константу арифметичного типу $\overline{n} : State \to Int$ таку, що $\overline{n}\ (st) = n$;
- функції-константи булевого типу $\overline{true}$, $\overline{false} : State \to Bool$ такі, що $\overline{true}\ (st) = true$, $\overline{false}\ (st) = false$;
- *тотожну функцію id*: $State \to State$ таку, що $id(st) = st$.

Отримали багатоосновну *алгебру даних розширеної мови SIPL*: *A_Int_Bool_State* = <*Int*, *Bool*, *State*; *add*, *sub*, *mult*, *div*, *or*, *and*, *neg*, *less*, *leq*, *eq*, *neq*, *geq*, *gr*, $\Rightarrow x$, $x\Rightarrow$,$\Rightarrow y$, $y\Rightarrow$, $\overline{n}$, $\overline{b}$, *id*, ∇, $\overline{true}$, $\overline{false}$ >

Формули обчислення задаються аналогічно алгебрі *A_Prog*. Нові правила побудови семантичного терму подано в табл. 1.7.

Таблиця 1.7

Нові правила побудови семантичного терму

Правило заміни	Номер правила
sem_P: *Prog* → *FS* задається правилами:	
sem_P(begin *S* end) = *sem_S*(*S*)	*NS_Prog*
sem_S: *Stm* → *FS* задається правилами:	
sem_S(*x*:=*a*)=AS^x(*sem_A*(*a*)) *sem_S*(*y*:=*c*)=AS^y(*sem_B*(*c*)) *sem_S*(S_1;S_2)= *sem_S*(S_1) •*sem_S*(S_2) *sem_S*(if *c* then S_1 else S_2)= *IF*(*sem_B*(*b*), *sem_S*(S_1), *sem_S*(S_2)) *sem_S*(while *c* do *S*)= *WH*(*sem_B*(*b*), *sem_S*(*S*)) *sem_S*(begin *S* end)=(*sem_S*(*S*)) *sem_S*(*skip*)=*id*	*NS_Stm_As* *NS_Stm_As_B* *NS_Stm_Seq* *NS_Stm_If* *NS_Stm_Wh* *NS_Stm_BE* *NS_Stm_skip*
sem_A: *Aexp* → *FA* задається правилами:	
sem_A(*n*))= $\overline{n}$ *sem_A*(*x*))=$x\Rightarrow$ *sem_A*(a_1+a_2)=S^2(*add*, *sem_A*(a_1), *sem_A*(a_2)) *sem_A*(a_1–a_2)=S^2(*sub*, *sem_A*(a_1), *sem_A*(a_2)) *sem_A*(a_1*a_2)=S^2(*mult*, *sem_A*(a_1), *sem_A*(a_2)) *sem_A*(a_1÷a_2)=S^2(*div*, *sem_A*(a_1), *sem_A*(a_2)) *sem_A*((*a*))=*sem_A*(*a*)	*NS_A_Num* *NS_A_Var* *NS_A_Add* *NS_A_Sub* *NS_A_Mult* *NS_A_Div* *NS_A_Par*

Закінчення табл. 1.7

Правило заміни	Номер правила
sem_B: *Bexp* → *FB* задається правилами:	
$sem\_B(b)=\overline{b}$	*NS_B_*Const
$sem\_B(y)=y\Rightarrow$	*NS_B_Var*
$sem\_B(a_1<a_2)=S^2(less, sem\_A(a_1), sem\_A(a_2))$	*NS_B_less*
$sem\_B(a_1\le a_2)=S^2(leq, sem\_A(a_1), sem\_A(a_2))$	*NS_B_leq*
$sem\_B(a_1=a_2)=S^2(eq, sem\_A(a_1), sem\_A(a_2))$	*NS_B_eq*
$sem\_B(a_1\ne a_2)=S^2(neq, sem\_A(a_1), sem\_A(a_2))$	*NS_B_neq*
$sem\_B(a_1\ge a_2)=S^2(geq, sem\_A(a_1), sem\_A(a_2))$	*NS_B_geq*
$sem\_B(a_1>a_2)=S^2(gr, sem\_A(a_1), sem\_A(a_2))$	*NS_B_gr*
$sem\_B(c_1<c_2)=S^2(less, sem\_B(c_1), sem\_B(c_2))$	*NS_B_lessB*
$sem\_B(c_1\le c_2)=S^2(leq, sem\_B(c_1), sem\_B(c_2))$	*NS_B_leqB*
$sem\_B(c_1=c_2)=S^2(eq, sem\_B(c_1), sem\_B(c_2))$	*NS_B_eqB*
$sem\_B(c_1\ne c_2)=S^2(neq, sem\_B(c_1), sem\_B(c_2))$	*NS_B_neqB*
$sem\_B(c_1\ge c_2)=S^2(geq, sem\_B(c_1), sem\_B(c_2))$	*NS_B_geqB*
$sem\_B(c_1>c_2)=S^2(gr, sem\_B(c_1), sem\_B(c_2))$	*NS_B_grB*
$sem\_B(c_1\vee c_2)=S^2(or, sem\_B(c_1), sem\_B(c_2))$	*NS_B_or*
$sem\_B(c_1\wedge c_2)=S^2(and, sem\_B(c_1), sem\_B(c_2))$	*NS_B_and*
$sem\_B(\neg c)=S^1(neg, sem\_B(c))$	*NS_B_neg*
$sem\_B((c))= sem\_B(c)$	*NS_B_Par*

1.7. Приклади розв'язання задач

Припустимо, що у SIPL введено лише базові необхідні предикати: ¬, ∨, ≤. Виразимо решту предикатів ∧, =, >, ≠, <, ≥ через базові: у змінну res запишемо результат обчислення відповідного предиката над значеннями вхідних змінних *A* та *B*.

Надалі $\overline{a}$, $\overline{b}$, $\overline{x}$, $\overline{y}$ будуть використовуватись для позначення значень відповідних змінних.

Завдання 1.7. Розписати операцію "$\wedge$" через операції "$\vee$" та "$\neg$" .

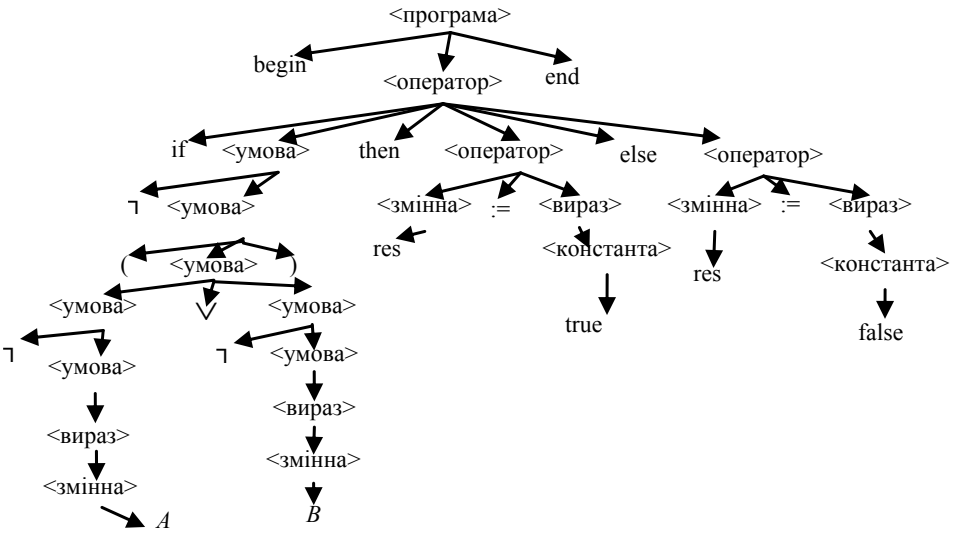
begin

if $\neg(\neg A \vee \neg B)$ then res: = *true*

else res: = *false*

end

Побудуємо дерево виведення:



Побудуємо семантичний терм:

$sem\_P(P) = sem\_S(if \neg(\neg A \vee B) \ \ then \ \ res := true \ \ else \ \ res := false)$

$= IF(S^1(neg, S^2(or, S^1(neg, A\Rightarrow), S^1(neg, B\Rightarrow))), AS^{res}(\overline{true}), AS^{res}(\overline{false}))$

Доведемо часткову коректність.

Маємо початкове дане $d = [A \mapsto \overline{a}, B \mapsto \overline{b}]$.

$sem\_P(P)(d) = IF\ (S^1\ (neg,\ S^2\ (or,\ S^1\ (neg,\ A\Rightarrow),\ S^1\ (neg,\ B\Rightarrow))),$
$AS^{res}(\overline{true}), AS^{res}(\overline{false}))(d) =$

$$= \begin{cases} AS^{res}(\overline{true})(d), \text{якщо } S^1(neg, S^2(or, S^1(neg, A\Rightarrow), S^1(neg, B\Rightarrow)))(d) = true \\ AS^{res}(\overline{false})(d), \text{якщо } S^1(neg, S^2(or, S^1(neg, A\Rightarrow), S^1(neg, B\Rightarrow)))(d) = false \end{cases}$$

$$= \begin{cases} d\nabla[res \to true], \text{якщо } S^1(neg, S^2(or, S^1(neg, \overline{a}), S^1(neg, \overline{b}))) = true \\ d\nabla[res \to true], \text{якщо } S^1(neg, S^2(or, S^1(neg, \overline{a}), S^1(neg, \overline{b}))) = false \end{cases}$$

$$= \begin{cases} [A \to \overline{a}, B \to \overline{b}, res \to true], \text{якщо } S^1(neg, S^2 \\ (or, S^1(neg, \overline{a}), S^1(neg, \overline{b}))) = true \\ [A \to \overline{a}, B \to \overline{b}, res \to false], \text{якщо } S^1(neg, S^2 \\ (or, S^1(neg, \overline{a}), S^1(neg, \overline{b}))) = false \end{cases}$$

$$= \begin{cases} [A \to \overline{a}, B \to \overline{b}, res \to true], \text{якщо } \neg(\neg\overline{a} \vee \neg\overline{b}) = true \\ [A \to \overline{a}, B \to \overline{b}, res \to false], \text{якщо } \neg(\neg\overline{a} \vee \neg\overline{b}) = false \end{cases}$$

$$= \begin{cases} [A \to \overline{a}, B \to \overline{b}, res \to true], \text{якщо } \overline{a} \wedge \overline{b} = true \\ [A \to \overline{a}, B \to \overline{b}, res \to false], \text{якщо } \overline{a} \wedge \overline{b} = false \end{cases}.$$

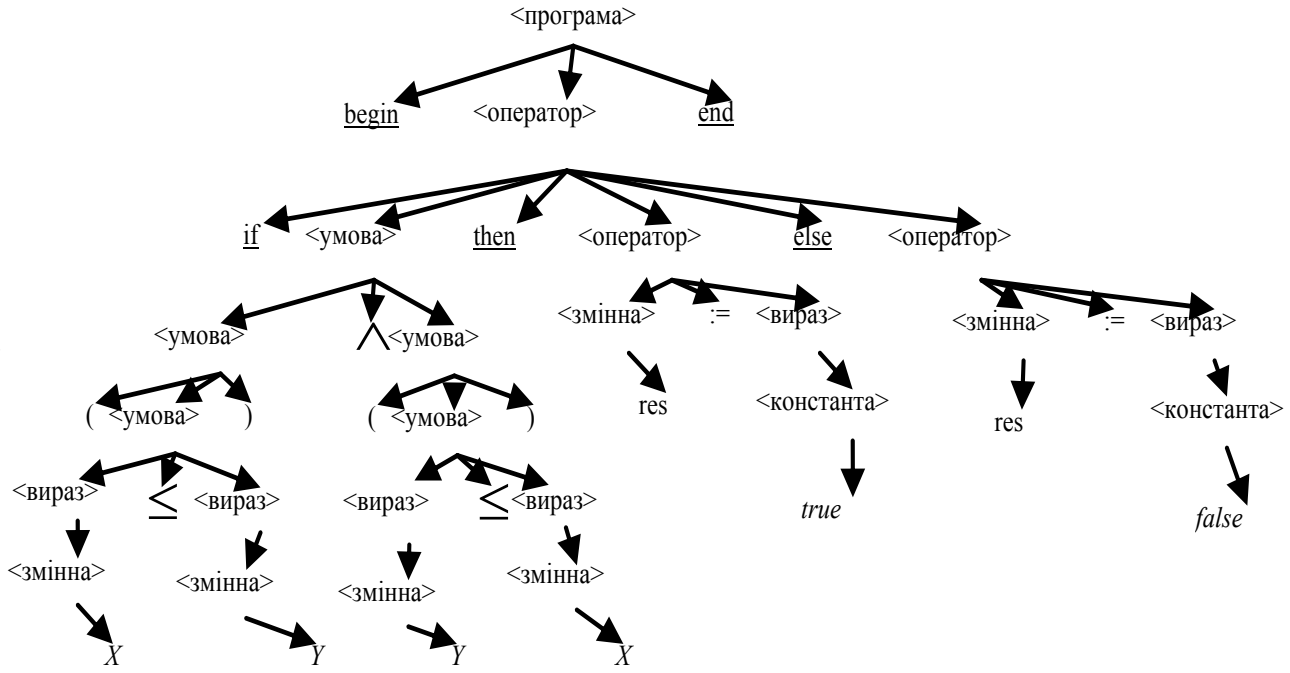
Часткову коректність доведено.

Програма не має циклів, тому вона завершується. А оскільки часткова коректність доведена, то маємо тотальну коректність.

Завдання 1.8. Розписати операцію "=" через "≤".

```
begin
    if (X ≤ Y) ∧ (Y ≤ X) then res:=true
    else res:=false
end
```

Побудуємо дерево виведення:



Побудуємо семантичний терм:

$sem\_P(P) = sem\_S\ (if\ \ (X \le Y) \wedge (Y \le X)\ then\ \ res := true\ \ else\ \ res := false) =$

$= IF(S^2(and,\ S^2(leq,\ X\Rightarrow,\ Y\Rightarrow),\ S^2(leq,\ Y\Rightarrow,\ X\Rightarrow)),\ AS^{res}(\overline{true}),\ AS^{res}(\overline{false}))$

Доведемо часткову коректність:

$d = [X \mapsto \overline{x},\ Y \mapsto \overline{y}\ ]$

$sem\_P(P)(d) =$

$= IF(S^2(and,\ S^2(leq,\ X\Rightarrow,\ Y\Rightarrow),\ S^2\ (leq,\ Y\Rightarrow,\ X\Rightarrow)),\ AS^{res}(\overline{true}),\ AS^{res}(\overline{false}))(d) =$

$$= \begin{cases} AS^{res}(\overline{true})(d), \text{якщо } S^2(and, S^2(leq, X\Rightarrow, Y\Rightarrow), S^2(leq, Y\Rightarrow, X\Rightarrow)) = true \\ AS^{res}(\overline{false})(d), \text{якщо } S^2(and, S^2(leq, X\Rightarrow, Y\Rightarrow), S^2(leq, Y\Rightarrow, X\Rightarrow)) = true \end{cases}$$

$$= \begin{cases} d\nabla[res \mapsto true], \text{якщо } S^2(and, S^2(leq, \overline{x}, \overline{y}), S^2(leq, \overline{y}, \overline{x})) = true \\ d\nabla[res \mapsto false], \text{якщо } S^2(and, S^2(leq, \overline{x}, \overline{y}), S^2(leq, \overline{y}, \overline{x})) = false \end{cases}$$

$$= \begin{cases} X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } ((\overline{x} \le \overline{y}) \wedge (\overline{y} \le \overline{x})) = true \\ X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } ((\overline{x} \le \overline{y}) \wedge (\overline{y} \le \overline{x})) = false \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } (\overline{x} = \overline{y}) = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } (\overline{x} = \overline{y}) = false \end{cases}$$

Часткову коректність доведено.

Програма не має циклів, тому вона завершується. А оскільки часткова коректність доведена, то маємо тотальну коректність.

Завдання 1.9. Визначити операцію ">":

```
begin
    if ¬(X ≤ Y) then res: = true
                else res: = false
end
```

Побудуємо семантичний терм:

$sem\_P(P) = sem\_S(if \neg(X \le Y)\ then\ res := true\ else\ res := false) =$

$= IF(S^1(neg, S^2(leq, X\Rightarrow, Y\Rightarrow)), AS^{res}(\overline{true}), AS^{res}(\overline{false}))$

Доведемо часткову коректність:

$d=[X \mapsto \overline{x}, Y \mapsto \overline{y}]$

$sem\_P(P)(d) = IF(S^1(neg, S^2(leq, X\Rightarrow, Y\Rightarrow)), AS^{res}(\overline{true}), AS^{res}(\overline{false}))(d) =$

якщо

$$= \begin{cases} S^{res}(\overline{true})(d), \text{якщо } S^1(neg, S^2(leq, X \Rightarrow, Y \Rightarrow))(d) = true \\ AS^{res}(\overline{false})(d), \text{якщо } S^1(neg, S^2(leq, X \Rightarrow, Y \Rightarrow))(d) = false \end{cases}$$

$$= \begin{cases} d\nabla[res \mapsto true], \text{якщо } S^1(neg, S^2(leq, \overline{x}, \overline{y})) = true \\ d\nabla[res \mapsto false], \text{якщо } S^1(neg, S^2(leq, \overline{x}, \overline{y})) = false \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } \neg(\overline{x} \le \overline{y}) = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } \neg(\overline{x} \le \overline{y}) = false \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } \overline{x} > \overline{y} = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } \overline{x} > \overline{y} = false \end{cases}$$

Часткову коректність доведено.

Програма не має циклів, тому вона завершується. А оскільки часткова коректність доведена, то маємо тотальну коректність.

Завдання 1.10. Визначити операцію "≠":

```
begin
    if ¬(X = Y) then res:=true
                else res:=false
end
```

Побудуємо семантичний терм:

$sem\_P(P) = sem\_S(if\ \neg(X = Y)\ then\ res := true\ else\ res := false) =$

$= IF(S^1(neg, S^2(eq, X\Rightarrow, Y\Rightarrow)), AS^{res}(\overline{true}), AS^{res}(\overline{false}))$

Доведемо часткову коректність:

$d = [X \mapsto \overline{x}, Y \mapsto \overline{y}]$

$sem\_P(P)\ (d) = IF\ (S^1\ (neg,\ S^2(eq,\ X\Rightarrow,\ Y\Rightarrow)),\ AS^{res}(\overline{true}),$
$AS^{res}(\overline{false}))\ (d) =$

$$= \begin{cases} AS^{res}(\overline{true})(d), \text{якщо } S^1(neg, S^2(eq, X \Rightarrow, Y \Rightarrow))(d) = true \\ AS^{res}(\overline{false})(d), \text{якщо } S^1(neg, S^2(eq, X \Rightarrow, Y \Rightarrow))(d) = false \end{cases}$$

$$= \begin{cases} d\nabla[res \mapsto true], \text{якщо } S^1(neg, S^2(eq, \overline{x}, \overline{y})) = true \\ d\nabla[res \mapsto fals], \text{якщо } S^1(neg, S^2(eq, \overline{x}, \overline{y})) = false \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } \neg(\overline{x} = \overline{y}) = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } \neg(\overline{x} = \overline{y}) = false \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } \overline{x} \neq \overline{y} = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } \overline{x} \neq \overline{y} = false \end{cases}$$

Часткову коректність доведено.

Програма не має циклів, тому вона завершується. А оскільки часткова коректність доведена, то маємо тотальну коректність.

Завдання 1.11. Визначити операцію "<":

begin

if $(X \le Y) \wedge (X \ne Y)$ then res: = *true*

else res: = *false*

end

Побудуємо семантичний терм:

$sem\_P(P) = sem\_S(\text{if} \quad (X \le Y) \wedge (X \ne Y) \quad then \quad res := true \quad else \; res := false) =$

$= IF(S^2(and, S^2(leq, X\Rightarrow,\ Y\Rightarrow),\ S^2(neq,\ X\Rightarrow,\ Y\Rightarrow)),\ AS^{res}(\overline{true}),\ AS^{res}(\overline{false}))$

Доведемо часткову коректність:

$d = [X \mapsto \overline{x}, Y \mapsto \overline{y}\,]$

$sem\_P(P)(d) =$

$= IF\ (S^2\ (and,\ S^2\ (leq,\ X\Rightarrow,\ Y\Rightarrow),\ S^2\ (neq,\ X\Rightarrow,\ Y\Rightarrow)),\ AS^{res}(\overline{true}),\ AS^{res}(\overline{false}))(d) =$

$$= \begin{cases} AS^{res}(\overline{true}), \text{якщо } S^2(and, S^2(leq, X\Rightarrow, Y\Rightarrow), S^2(neq, X\Rightarrow, Y\Rightarrow)(d)) = true \\ AS^{res}(\overline{false}), \text{якщо } S^2(and, S^2(leq, X\Rightarrow, Y\Rightarrow), S^2(neq, X\Rightarrow, Y\Rightarrow)(d)) = false \end{cases}$$

$$= \begin{cases} d\nabla[res \mapsto true], \text{якщо } S^2(and, S^2(leq, \overline{x}, \overline{y}), S^2(neq, \overline{x}, \overline{y})) = true \\ d\nabla[res \mapsto false], \text{якщо } S^2(and, S^2(leq, \overline{x}, \overline{y}), S^2(neq, \overline{x}, \overline{y})) = false \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } (\overline{x} \le \overline{y}) \wedge (\overline{x} \ne \overline{y}) = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } (\overline{x} \le \overline{y}) \wedge (\overline{x} \ne \overline{y}) = false \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } (\overline{x} < \overline{y}) = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } (\overline{x} < \overline{y}) = false \end{cases}$$

Часткову коректність доведено.

Програма не має циклів, тому вона завершується. А оскільки часткова коректність доведена, то маємо тотальну коректність.

Завдання 1.12. Визначити операцію "≥":

begin

if $(X > Y) \vee (X = Y)$ then res: $= true$

else res: $= false$

end

Побудуємо семантичний терм:

$sem\_P(P) = sem\_S(\text{if} \quad (X > Y) \vee (X = Y) \quad \text{then} \quad res := true \quad \text{else} \; res := false) =$

$= IF(S^2(or,\ S^2(gr,\ X\Rightarrow,\ Y\Rightarrow),\ S^2(eq,\ X\Rightarrow,\ Y\Rightarrow)),\ AS^{res}(\overline{true}),\ AS^{res}(\overline{false}))$

Доведемо часткову коректність:

$d = [X \mapsto \overline{x},\ Y \mapsto \overline{y}\,]$

$sem\_P(P)(d) = IF(S^2(or,\ S^2(gr,\ X\Rightarrow,\ Y\Rightarrow),\ S^2(eq,\ X\Rightarrow,\ Y\Rightarrow)),\ AS^{res}(\overline{true}),\ AS^{res}(\overline{false}))(d) =$

$$= \begin{cases} AS^{res}(\overline{true})(d), \text{якщо } S^2(or, S^2(gr, X\Rightarrow, Y\Rightarrow), S^2(eq, X\Rightarrow, Y\Rightarrow))(d) = true \\ AS^{res}(\overline{false})(d), \text{якщо } S^2(or, S^2(gr, X\Rightarrow, Y\Rightarrow), S^2(eq, X\Rightarrow, Y\Rightarrow))(d) = false \end{cases}$$

$$= \begin{cases} d\nabla[res \mapsto \text{true}], \text{якщо } S^2(or, S^2 9gr, \overline{x}, \overline{y}), S^2(eq, \overline{x}, \overline{y})) = \text{true} \\ d\nabla[res \mapsto \text{false}], \text{якщо } S^2(or, S^2 9gr, \overline{x}, \overline{y}), S^2(eq, \overline{x}, \overline{y})) = \text{false} \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } (\overline{x} > \overline{y}) \vee (\overline{x} = \overline{y}) = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } (\overline{x} > \overline{y}) \vee (\overline{x} = \overline{y}) = false \end{cases}$$

$$= \begin{cases} [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto true], \text{якщо } (\overline{x} \geq \overline{y}) = true \\ [X \mapsto \overline{x}, Y \mapsto \overline{y}, res \mapsto false], \text{якщо } (\overline{x} \geq \overline{y}) = false \end{cases}$$

Часткову коректність доведено.

Програма не має циклів, тому вона завершується. А оскільки часткова коректність доведена, то маємо тотальну коректність.

1.8. Розширення мови SIPL. Введення викликів функцій

Подамо у табл. 1.8 синтаксис БНФ мови SIPL, розширеної функціями.

Таблиця 1.8

Синтаксис розширеної БНФ

Ліва частина правила – метазмінна	**Права частина правила**	**Ім'я правила**
<програма> ::=	...| program <список оголошень функцій> begin <оператор> end	*NP*1 *NP*2
<вираз> ::=	...| if <умова> then <вираз> else <вираз> | <виклик функції>	*NA*1–*NA*7 *NA*8 *NA*9
<список оголошень функцій>::=	ε | <оголошення функції> | <оголошення функції> ; <список оголошень функцій>	*LDF*1 *LDF*2 *LDF*3
<оголошення функції>::=	func <ім'я функції>=<вираз> | func <ім'я функції>(<список формальних параметрів>) = <вираз>	*DF*1 *DF*2
<ім'я функції> ::=	<змінна>	*NF*
<список формальних параметрів>::=	<змінна> | <змінна>, <список формальних параметрів>	*LFP*1 *LFP*2
<виклик функції> ::=	<ім'я функції> | <ім'я функції> (<список фактичних параметрів>)	*CF*1 *CF*2
<список фактичних параметрів>::=	<вираз> | <вираз>, <список фактичних параметрів>	*LAP*1 *LAP*2

Отже, ми змінили синтаксис виразу, додавши варіант завдання виразу за допомогою умов

<вираз> ::= if <умова> then <вираз1> else <вираз2>

якщо умова виконується, то <вираз> := <вираз1>,

інакше <вираз> := <вираз2>,

або ж <вираз> ::= <виклик функції>

Значення виразу дорівнюватиме результату виклику функції.

Зміниться й синтаксис програми: додамо варіант

<програма> ::= program <список оголошень функцій> begin <оператор> end,

де <список оголошень функцій> може бути порожнім або складатися з оголошень функцій, перерахованих через кому. Так ми отримаємо можливість використовувати функції зі списку оголошень функцій.

Синтаксис оголошення функції:

<оголошення функції>::= func <ім'я функції>=<вираз> |

func <ім'я функції>(<список формальних параметрів>) = <вираз>,

де список формальних параметрів – це список змінних, перерахованих через кому.

Синтаксис виклику функції:

<виклик функції> ::= <ім'я функції>, якщо функція не має параметрів, та

<виклик функції> ::= <ім'я функції> (<список фактичних параметрів>),

де список фактичних параметрів – це список виразів, перерахованих через кому.

Семантика суперпозиції при цьому може бути двох типів.

Глобальною суперпозицією функцій $g1$, $g2,\ldots,gn$ у номінативну функцію f називається функція, яка задається формулою $S^{(v1,v2,..vn)}(f,g_1,g_2,\ldots,g_n)(st)=f([v_1\to g_1(st),v_2\to g_2(st),\ldots,v_n\to g_n(st)])$.

Локальною суперпозицією функцій g_1, $g_2,\ldots,g_n$ у номінативну функцію f називається функція, яка задається формулою

$S^{[v1,v2,..vn]}(f,g_1,g_2,\ldots,g_n)(st)=$

$=f(st\nabla[v_1\to g_1(st),v_2\to g_2(st),\ldots,v_n\to g_n(st)])$.

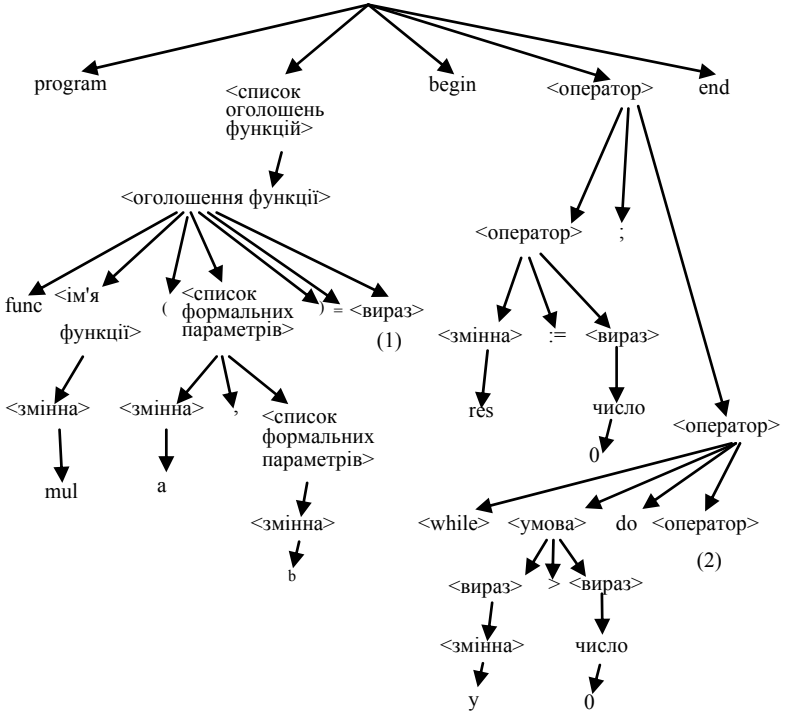
1.9. Приклади задач

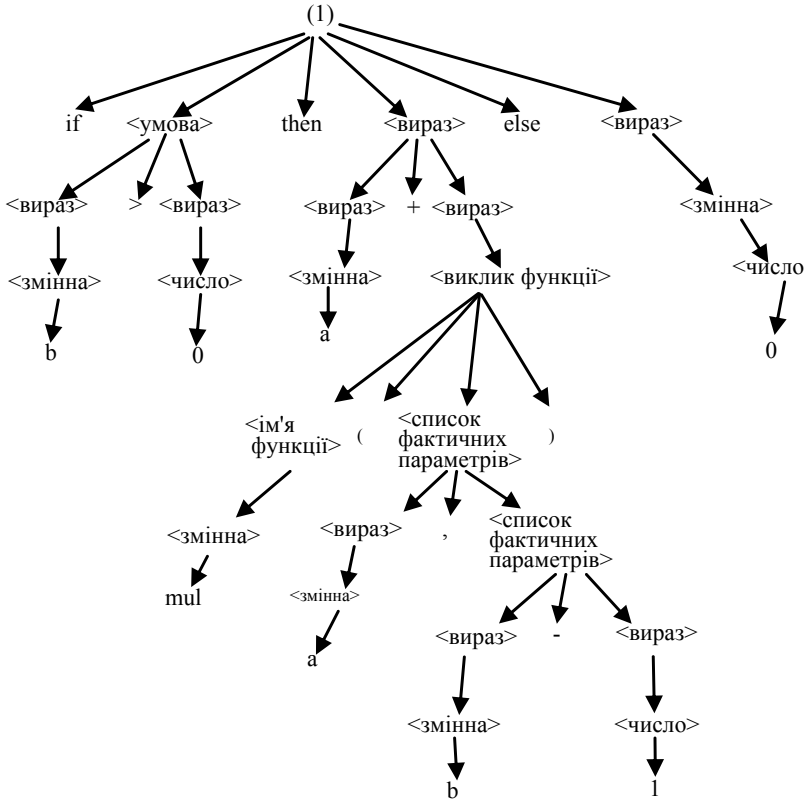
Задача. Обчислити *X^Y* через виклик рекурсивної функції множення двох чисел, реалізованої через додавання.

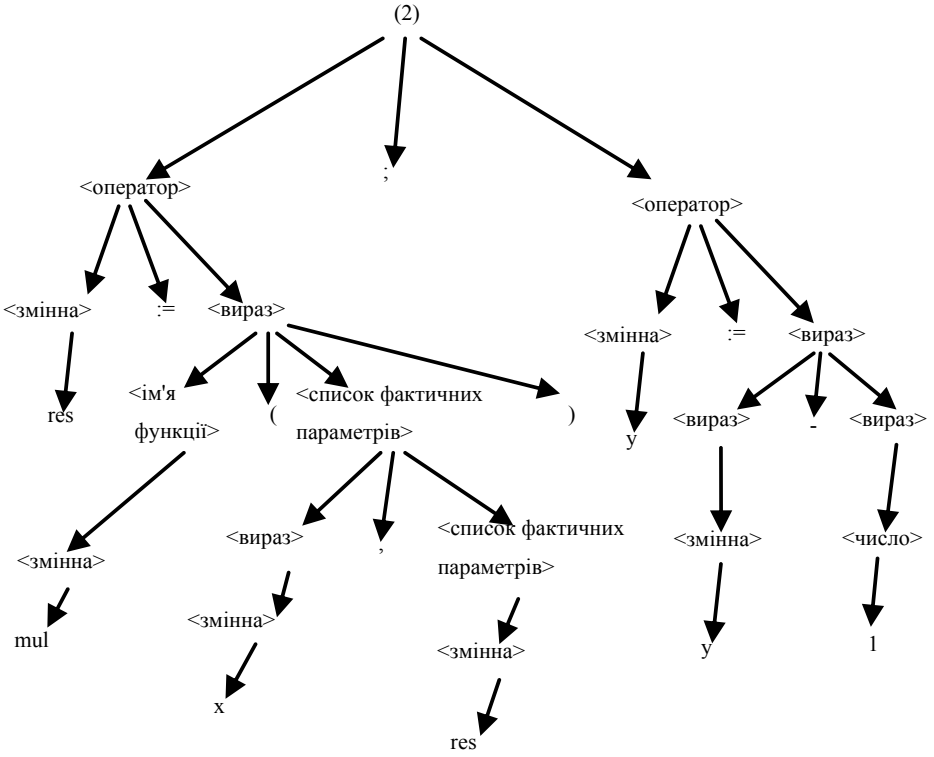
```
program
func mul(a, b)=if b>0 then a+mul(a, b–1) else 0
begin
 res:=1;
 while (y>0) do
 begin
 res:=mul(x,res);
 y:=y-1
 end
end
```

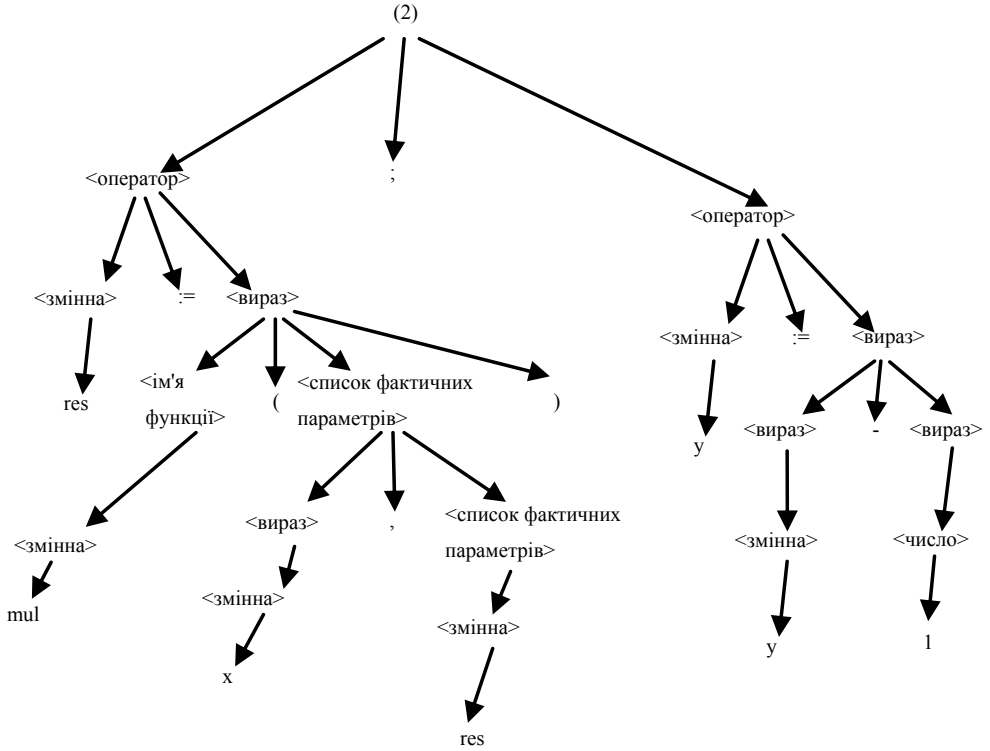
F_4 F_3 F_2 F_1 P

Дерево синтаксичного виведення програми в мові SIPL:




(1)
if
<умова>
then
<вираз>
else
<вираз>
<вираз>
>
<вираз>
<вираз>
+
<вираз>
<змінна>
<змінна>
<число>
<змінна>
<виклик функції>
<число>
b
0
a
0
<ім'я функції>
(
<список фактичних параметрів>
)
<змінна>
<вираз>
,
<список фактичних параметрів>
mul
<змінна>
a
<вираз>
-
<вираз>
<змінна>
<число>
b
1


(2)
<оператор>
;
<оператор>
<змінна>
:=
<вираз>
<змінна>
:=
<вираз>
res
<ім'я функції>
(
<список фактичних параметрів>
)
y
<вираз>
-
<вираз>
<змінна>
<вираз>
,
<список фактичних параметрів>
<змінна>
<змінна>
<число>
mul
<змінна>
y
1
x
<змінна>
res


(2)
<оператор>
;
<оператор>
<змінна>
:=
<вираз>
res
<ім'я функції>
(
<список фактичних параметрів>
)
<змінна>
mul
<вираз>
,
<список фактичних параметрів>
<змінна>
x
<змінна>
res
<змінна>
:=
<вираз>
y
<вираз>
-
<вираз>
<змінна>
y
<число>
1

Семантичний терм:

$sem\_P(P) = sem\_S(F_1) = sem\_S(F_2) = AS^{res}(1) \bullet sem\_S(F_3) = = AS^{res}(1) \bullet WH\ (sem\_B\ (y{>}0),\ sem\_S\ (F_4)) = AS^{res}(1) \bullet WH\ (S^2\ (gr,\ y\Rightarrow,\ 0),\ AS^{res}(sem\_A\ (mul\ (x,\ res)))\bullet\ AS^{y}\ (S^2\ (sub,\ y\Rightarrow,1)))=AS^{res}(1) \bullet WH\ (S^2\ (gr, y\Rightarrow, 0), AS^{res}(S^{a,b}\ (sem\_A\ (mul), x\Rightarrow, res\Rightarrow)\bullet AS^{y}(S^2\ (sub, y\Rightarrow,1))).$

Знайдемо $sem\_A(mul)$:

$sem\_A\ (mul) = IF\_A\ (S^2\ (gr,\ b\Rightarrow,\ 0),\ sem\_A\ (a\ +\ mul\ (a,\ b{-}1)),\ 0)\ = IF\_A\ (S^2\ (gr,\ b\Rightarrow,\ 0),\ S^2\ (add,\ a\Rightarrow,\ sem\_A\ (mul\ (a,\ b{-}1)),\ 0)\ = IF\_A\ (S^2\ (gr,\ b\Rightarrow,\ 0),\ S^2\ (add,a\Rightarrow,\ S^{a,b}\ (sem\_A\ (mul),\ a\Rightarrow,\ S^2\ (sub,\ b\Rightarrow,\ 1))),\ 0).$

Маємо рекурсивно задану семантику для функції.

Доведемо, що $sem\_A(mul)([a\mapsto A,\ b\mapsto B]) = A*B$, індукцією за B.

База індукції. Якщо $B = 0$, то

$sem\_A(mul)(st) = IF(S^2(gr,\ b\Rightarrow,\ 0),\ S^2(add,\ a\Rightarrow,\ S^{a,b}(sem\_A(mul),\ a\Rightarrow,\ S^2\ (sub,\ b\Rightarrow,\ 1))),\ 0)\ (st) = 0,$ оскільки $B = 0$.

Припущення. Нехай $sem\_A\ (mul)\ ([a\mapsto A,\ b\mapsto B]) = A*B$ для всіх $B<k$.

Крок індукції. Нехай $B = k{>}0$. Маємо:

$sem\_A(mul)(st) =$

$= IF\_A(S^2(gr,\ b\Rightarrow,0),\ S^2(add,\ a\Rightarrow,\ S^{a,b}\ (sem\_A\ (mul),\ a\Rightarrow,\ S^2(sub,\ b\Rightarrow,1))),\ 0)\ (st) = S^2\ (add,\ a\Rightarrow,\ S^{a,b}\ (sem\_A\ (mul),\ a\Rightarrow,\ S^2(sub,\ b\Rightarrow,\ 1)))\ (st).$

Використовуючи означення локальної композиції, отримаємо:

$sem\_A(mul)(st) = S^2(add,\ a\Rightarrow\ (st),\ sem\_A\ (mul)\ (st\nabla[a\mapsto A,\ b\mapsto B{-}1])) = A{+}A*(B{-}1){=}A{+}A*B{-}A{=}A*B.$

Формула доведена.

Доведемо часткову коректність програми.

Нехай на вхід програми вводиться стан $st = [x\mapsto X, y\mapsto Y]\ X, Y{\geq}0$

$sem\_P\ (P)\ (st) = AS^{res}\ (1) \bullet WH\ (S^2\ (gr,\ y\Rightarrow,\ 0),\ AS^{res}\ (S^{a,b}\ (sem\_A\ (mul),\ x\Rightarrow,\ res\Rightarrow)\bullet\ AS^{y}\ (S^2\ (sub,\ y\Rightarrow,\ 1)))\ (st){=}WH\ (S^2\ (gr,\ y\Rightarrow,\ 0),\ AS^{res}(S^{a,b}(sem\_\mathrm{A}(mul),\ x\Rightarrow,res\Rightarrow)\bullet AS^{y}(S^2(sub,\ y\Rightarrow,1)))(st\nabla\ [res\mapsto 1]).$

Нехай $WH(S^2(gr, y\Rightarrow, 0), \quad AS^{res}(S^{a,b}(sem\_A(mul), x\Rightarrow, res\Rightarrow)\bullet AS^{y}(S^2(sub, y\Rightarrow,1)))=f$. Тоді висунемо гіпотезу, що $f(st_{in})=st_{out}$, де

$st_{in}=[x\mapsto U, y\mapsto V, res\mapsto R]$, $U,V\geq 0$ – вхідний стан для циклу,

$st_{out}=[x\mapsto U, y\mapsto 0, res\mapsto R*U\hat{}V]$– результат виконання циклу.

Доведемо, що ця рівність виконується, індукцією за кількістю ітерацій циклу:

База індукції. $NumItWH=0$, тому $V\leq 0$. З умови $V\geq 0$ випливає, що $V=0$. Тоді рівність $f(st_{in})=st_{out}$ виконується.

Припущення. Нехай рівність $f(st_{in})=st_{out}$ виконується для всіх циклів з кількістю ітерацій менше деякого додатного k:

$NumItWH<k, k>0$

Крок індукції. Доведемо, що рівність виконується також для циклів з k ітераціями.

$f(st_{in})=WH(S^2(gr, y\Rightarrow,0), AS^{res}(S^{a,b}(sem\_A(mul), x\Rightarrow, res\Rightarrow)\bullet AS^{y}(S^2(sub, y\Rightarrow, 1)))(st_{in})$. Використаємо формулу $WH(p, f)=IF(p, f\bullet WH(p, f), id)$. Ураховуючи, що кількість ітерацій додатна, а тому умова p на початковому стані істинна, маємо: $(AS^{res}(S^{a,b}(sem\_A(mul), x\Rightarrow, res\Rightarrow)\bullet AS^{y}(S^2(sub, y\Rightarrow,1))\bullet f)(st_{in})$. Розпишемо композицію:

$f(st_{in})=f(AS^{res}(S^{a,b}(sem\_A(mul), x\Rightarrow, res\Rightarrow)\bullet AS^{y}(S^2(sub, y\Rightarrow, 1)))(st_{in}))=f\ (AS^{y}\ (S^2\ (sub, y\Rightarrow, 1)\ (AS^{res}\ (S^{a,b}\ (sem\_A\ (mul), x\Rightarrow, res\Rightarrow)(st_{in})))=f\ (AS^{y}\ (S^2\ (sub, y\Rightarrow, 1)\ ([x\mapsto U, y\mapsto V, res\mapsto U*R]))=$

$=f([x\mapsto U, y\mapsto V-1, res\mapsto U*R])$.

Оскільки виконано одну ітерацію циклу, то для стану $[x\mapsto U, y\mapsto V-1, res\mapsto U*R]$ кількість ітерацій становитиме $k-1$. Отже, оскільки з умови $y>0$ випливає, що $V-1\geq 0$, то

$f([x\mapsto U, y\mapsto V-1, res\mapsto U*R])=[x\mapsto U, y\mapsto 0, res\mapsto (U*R)*U\hat{}(V-1)]$ за припущенням індукції. Однак $[x\mapsto U, y\mapsto 0, res\mapsto (U*R)*U\hat{}(V-1)]=[x\mapsto U, y\mapsto 0, res\mapsto R*U\hat{}V]=st_{out.}$

Тому $f(st_{in}) = st_{out}$ для кожного $st_{in} = [x \mapsto U,\ y \mapsto V,\ res \mapsto R]$ за умови, що $U, V \geq 0$. Отже,

$sem\_P(P) = f([x \mapsto X, y \mapsto Y, res \mapsto 1]) = [x \mapsto X, y \mapsto 0, res \mapsto X\hat{}Y]$.

Часткова коректність доведена.

Неважко бачити, що значення y на кожній ітерації циклу утворюють строго спадну послідовність, обмежену нулем. Унаслідок скінченності значення y виконання циклу завжди буде завершеним; за умови $y = 0$ цикл не буде виконуватись. Унаслідок скінченності команд програми вона завжди зупиняється, тому справджується її повна коректність.

Завдання для самоконтролю

1. Написати програму для ділення двох чисел будь-якого знака.
2. Обчислити суму арифметичної прогресії.
3. Обчислити суму геометричної прогресії за формулою суми геометричної прогресії.
4. Знайти подвійний факторіал, якщо n – непарне.
5. Написати програму для додавання двох невід'ємних чисел через плюс два і мінус один.
6. Обчислити такі функції:

a) $y = [\sqrt{x}]$;

b) $y = [\log_2 x]$;

c) $y = 3^{x+2}$;

d) $y = x!x^x$;

e) $f(x, y) = (x + y)^2$ через плюс один;

f) $f(x, y) = y^{[\log_2 x]}$;

g) $f(x) = sg(x)$;

h) $f(x, y) = \max(x, y)$;

i) $f(x) = [x/2]$;

j) $f(x, y) = x \dot{-} y$;

k) $f(x, y) = 2x+y+1$;

l) $f(x) = nsg(x)$;

m) $f(x, y) = \min(x, y)$;

n) $f(x, y, z) = \max(x, y)+z$;
o) $f(x, y, z) = \min(x+y, z)$;
p) $f(x, y) = x^y$;
q) $f(x, y) = |x-y|$;
r) $f(x) = [\sqrt{x}]$;
s) $f(x, y) = (x+1)^2 \cdot y$;
t) $f(x, y) = \text{НСК}(x, y)$;
u) $f(x, y) = (x+1) \cdot 2^y$;
v) $f(x) = x^4 - x$;
w) $f(x) = 2^x - 3x$;
x) $f(x, y) = 3^x \cdot (y+1)$;
y) $f(x) = (2x)!!$;
z) $f(x) = [\log_2 x]$;
aa) $f(x, y, z) = x^{y+z}$;
bb) $f(x, y) = [\sqrt[x]{y}]$;
cc) $f(x, y) = \text{mod}(x, y)$;
dd) $f(x, y) = [(x+1)\sqrt{y}]$;
ee) $f(x, y, z) = \min(\text{mod}(x, y), z)$;
ff) $f(x, y, z) = \text{mod}(x+1, \min(y, z))$;
gg) $f(x, y, z) = \text{НСК}(x, y+z)$.

7. Обчислити предикати:

a) $P(x)$: "x – непарне число";
b) $P(x)$: "x – парне число";
c) $P(x, y)$: "$x = y$";
d) $P(x, y)$: "$x \geq y$";
e) $P(x, y)$: "$x > y$";
f) $P(x, y)$: "$x \neq y$";
g) $P(x, y)$: "$x < y$";
h) $P(x)$: "x не кратне 2";
i) $P(x)$: "x не кратне 3";
j) $P(x, y, z)$: "$z = (x \text{ div } y)$";
k) $P(x, y, z)$: "$z = (x \bmod y)$";
l) $P(x, y)$: "$x = y!$";
m) $P(n,m)$: "m є n-тим числом Фібоначчі".

Приклади завдань для контрольної роботи

Варіант 1

I. Основні поняття програмування.

II. Для задачі обчислення $x - y$ ($x, y>0$) з використанням функції –1 зробити таке:

1) написати SIPL-програму;

2) побудувати дерево її синтаксичного виведення;

3) перевірити синтаксичну правильність програми;

4) побудувати семантичний терм програми;

5) застосувати отриманий семантичний терм до вхідних даних $x = 8, y = 2$ (тестування);

6) довести семантичну правильність програми (верифікація).

Варіант 2

I. Основні програмні поняття.

II. Для задачі обчислення [lg n] з використанням функцій *div*, *mod*, +, – ($n>0$) зробити таке:

1) написати SIPL-програму;

2) побудувати дерево її синтаксичного виведення;

3) перевірити синтаксичну правильність програми;

4) побудувати семантичний терм програми;

5) застосувати отриманий семантичний терм до вхідних даних $n = 17$ (тестування);

6) довести семантичну правильність програми (верифікація).

Варіант 3

I. Сутнісні аспекти програм.

II. Для задачі обчислення 3^x з використанням функцій ∗,+, – ($x>0$) зробити таке:

1) написати SIPL-програму;

2) побудувати дерево її синтаксичного виведення;

3) перевірити синтаксичну правильність програми;

4) побудувати семантичний терм програми;

5) застосувати отриманий семантичний терм до вхідних даних $x = 3$ (тестування);

6) довести семантичну правильність програми (верифікація).

Варіант 4

I. Порівняти природні та формальні мови.

II. Для задачі обчислення [$\sqrt{n}$] з використанням функцій *, +, – (n>0) зробити таке:

1) написати SIPL-програму;

2) побудувати дерево її синтаксичного виведення;

3) перевірити синтаксичну правильність програми;

4) побудувати семантичний терм програми;

5) застосувати отриманий семантичний терм до вхідних даних $n = 7$ (тестування);

6) довести семантичну правильність програми (верифікація).

РОЗДІЛ 2
ФОРМАЛЬНІ МОВИ ТА ГРАМАТИКИ

2.1. Теорія формальних мов і граматик

Означення основних понять формальних мов

Означення 2.1. *Алфавітом* називають скінченну непорожню множину символів (літер). Позначатимемо алфавіт знаком Σ.

Приклад 2.1. Найчастіше використовують такі алфавіти:
$\Sigma = \{0,1\}$ – бінарний чи двійковий алфавіт;
$\Sigma = \{a, b, \ldots z\}$ – множина літер англійського алфавіту.

Означення 2.2. *Ланцюжком* (інколи словом, реченням, рядком) в алфавіті Σ називають скінченну послідовність символів із Σ.

Приклад 2.2. Послідовність 01101 – це ланцюжок у бінарному алфавіті $\Sigma = \{0,1\}$. Ланцюжок 111 також є ланцюжком у цьому алфавіті.

Означення 2.3. *Порожній ланцюжок* – це ланцюжок, який не містить жодного символу. Цей ланцюжок позначається ε. Його можна розглядати як ланцюжок у довільному алфавіті.

Означення 2.4. *Довжина слова* (послідовності) w позначається $|w|$; якщо $\Sigma' \subseteq \Sigma$, то довжина послідовності, утвореної з w видаленням тих символів, що не належать Σ', позначається $|w|_{\Sigma'}$; якщо $a \in \Sigma$, то $|w|_a$ означає $|w|_{\{a\}}$ і задає кількість входжень символу a у w.

Приклад 2.3. Для ланцюжка *abcbacaaccccb* маємо:
$|abcbacaaccccb| = 12$, $|abcbacaaccccb|_{\{a,\,c\}} = 9$, $|abcbacaaccccb|_c = 5$.

Означення 2.5. Якщо w та u – ланцюжки в алфавіті Σ, то ланцюжок wu (результат дописування слова u в кінець слова w) називається *конкатенацією* (катенацією, зчепленням) слів w та u. Іноді конкатенацію слів позначають $w \cdot u$.

Означення 2.6. Якщо w – ланцюжок у алфавіті Σ, то ланцюжок $\underbrace{w \cdot w \cdot \ldots \cdot w}_{n \quad разів}$ називається *n-м степенем* w і позначається w^n.

За означенням $w^0 = \varepsilon$.

Приклад 2.4. $a^3 = aaa$, $a^2b^3c = aabbbc$,
$(abcbacaacccb)^2 = abcbacaacccbabcbacaacccb$.

Означення 2.7. Множина всіх ланцюжків у алфавіті Σ позначається Σ* і називається *вільною напівгрупою*, породженою Σ. Множина всіх непорожніх ланцюжків позначається Σ^+.

Приклад 2.5. Якщо $\Sigma = \{a\}$, то $\Sigma^* = \{\varepsilon, a, aa, aaa, \ldots\}$, $\Sigma^+ = \{a, aa, aaa, \ldots\}$.

Термін "вільна напівгрупа" походить з алгебри. Напівгрупою називається множина з асоціативною бінарною операцією. Якщо така множина має одиничний елемент, то її називають моноїдом. Напівгрупа вільна, якщо жодних інших співвідношень (крім асоціативності) немає. У теорії формальних мов часто вважають, що напівгрупа має одиничний елемент. Множину Σ* можна розглядати як вільну напівгрупу з одиницею. Конкатенація є асоціативною операцією, а порожній ланцюжок ε – одиницею, тому що для довільного ланцюжка w маємо $w \cdot \varepsilon = \varepsilon \cdot w = w$. Такий розгляд множини ланцюжків Σ* дозволяє перейти до багатшої структури напівгрупи й використати алгебраїчні властивості для дослідження цієї множини. Іншою обставиною є те, що, подаючи множину послідовностей у вигляді напігрупи, ми ототожнюємо символ з послідовністю довжиною 1, що складається з цього символу. Тому замість двохосновної моделі (алфавіт і множина послідовностей), яка розрізняє символи й послідовності, розглядається лише одна основа – множина ланцюжків. Це спрощує дослідження.

Означення 2.8. Ланцюжок t є *підланцюжком* ланцюжка w, якщо $w = utv$ для деяких ланцюжків u та v.

Нарешті, введемо поняття формальної мови.

Означення 2.9. Якщо $L \subseteq \Sigma^*$, то L називається *формальною мовою* (або просто мовою) над алфавітом Σ (в алфавіті Σ).

Приклад 2.6. Множина ланцюжків $\{a^n b^n c^n \mid n \geq 0\} = \{\varepsilon, abc, a^2b^2c^2, a^3b^3c^3, \ldots\}$ є формальною мовою над алфавітом $\{a, b, c\}$.

Зауважимо: якщо L є мовою над Σ, то можна стверджувати, що L – це мова над будь-яким алфавітом Σ', яка містить Σ. Інакше кажучи, є певна інваріантність поняття мови при розширенні алфавіту. Крім того, є монотонність множини мов щодо розширення алфавіту, оскільки це веде до збільшення множини мов.

Операції над формальними мовами

Операції над формальними мовами поділяють на два класи. Перший клас операцій відповідає надабстрактній моделі мов, у якій вони мають інтенсіонал множини. Тому всі теоретико-множинні операції можна застосовувати для мов. У першу чергу, це операції *об'єднання* $\cup$, *перетину* $\cap$ та *різниці* $\setminus$. Вважаємо, що мови – аргументи цих операцій – задані над одним і тим самим алфавітом. Якщо це не так, то будуємо новий алфавіт, який є об'єднанням алфавітів мов-аргументів. Тоді всі мови-аргументи будуть мовами над одним алфавітом. Якщо мова L є мовою в алфавіті Σ, то мова $\Sigma^* \setminus L$ називається *доповненням* мови L відносно алфавіту Σ. Коли з контексту ясно, про який алфавіт ідеться, то кажуть, що мова $\Sigma^* \setminus L$ є доповненням мови L і позначається $\overline{L}$.

Приклад 2.7. Нехай $L1 = \{a^n b^n c^m \mid n,m \geq 0\}$, $L2 = \{a^n b^m c^m \mid n,m \geq 0\}$. Тоді

$L1 \cap L2 = \{ a^n b^n c^n \mid n \geq 0\}$, $L1 \setminus L2 = \{a^n b^n c^m \mid m \neq n, n,m \geq 0\}$.

Другий клас операцій над формальними мовами відповідає абстрактній моделі мов, коли їх елементи тлумачаться як послідовності символів. Серед цих операцій, у першу чергу, зазначимо

операцію добутку (конкатенації) мов, яка є похідною від операції конкатенації ланцюжків. Для позначення цієї операції використовуємо той самий символ операції конкатенації.

Означення 2.10. Нехай $L_1, L_2 \subseteq \Sigma^*$.

Тоді $L_1 \cdot L_2 \overset{def}{=} \{xy \mid x \in L_1, y \in L_2\}$. Мова $L_1 \cdot L_2$ називається конкатенацією мов L_1 та L_2.

Приклад 2.8. Якщо $L_1 = \{a, abac\}$ та $L_2 = \{bcc, c\}$, то $L_1 \cdot L_2 = \{ac, abacb, abacbac, abcc\}$.

Найважливішою характеристикою операції добутку мов є її асоціативність. Одиницею цієї операції є мова, що складається з порожнього ланцюжка, тобто мова $\{\varepsilon\}$. Похідною від добутку є операція піднесення до степеня. Вважаємо, що $L^0 \overset{def}{=} \{\varepsilon\}$, $L^n \overset{def}{=} \underbrace{L \cdot \ldots \cdot L}_{n \quad разів}$.

Маючи степінь, дамо індуктивне означення *ітерації*, яку ще називають *замиканням Кліні*, або *зірочкою Кліні*.

Означення 2.11. Ітерацією мови L (позначається L^*) називається мова $\bigcup_{n \in Nat} L^n$.

Приклад 2.9. $\{ab\}^* = \{ \varepsilon, ab, abab, ababab, \ldots\}$.

Для мови, що складається з одного символу, фігурні дужки часто опускають і пишуть, наприклад, a^* заміть $\{a\}^*$.

Означення 2.12. *Оберненням*, або дзеркальним образом ланцюжка w (позначається w^R) називається ланцюжок, складений із символів w, узятих в оберненому порядку.

Приклад 2.10. Якщо $w = abc$, то $w^R = cba$.

Означення 2.13. Нехай $L \subseteq \Sigma^*$. Тоді $L^R \stackrel{def}{=} \{w^R \mid w \in L\}$.

Крім вищенаведених, можна ввести також інші операції цього рівня, наприклад ділення мов, проекції на підалфавіт тощо. Також можна ввести операції індуктивного та рекурсивного означення мов. Такі операції будуть розглянуті нижче.

Хоча ми визначили чимало операцій над формальними мовами, їх недостатньо, щоб задавати важливі типи мов. Тому потрібно переходити до методів дескриптивного подання мов, тобто до граматик.

За способом подання правильних ланцюжків формальні граматики поділяють на породжувальні й розпізнавальні (граматики породження та сприйняття). До породжувальних належать граматики, які дозволяють побудувати будь-який правильний ланцюжок із зазначенням його структури й не дозволяють побудувати жодного неправильного ланцюжка. Розпізнавальна граматика – це граматика, яка дозволяє визначити, чи є довільно обраний ланцюжок правильним і, якщо він правильний, то визначити його структуру.

Формальні граматики широко застосовують у лінгвістиці, інформатиці, логіці та програмуванні у зв'язку з вивченням природних і штучних мов.

Почнемо з розгляду породжувальних граматик.

Породжувальні граматики

Як зазначалося на початку розділу, породжувальні граматики можуть розглядатися як конкретизації транзиційних або дедуктивних систем. Для таких систем головним є відношення переходів. У граматиках відношення переходу (виведення) задається за допомогою правил граматики (продукцій), які мають вигляд $\alpha \to \beta$, де α та β – ланцюжки в певному алфавіті Σ. Таке правило дозволяє перетворити ланцюжок γ_1 на ланцюжок γ_2 ($\gamma_1, \gamma_2 \in \Sigma^*$) тоді й тільки тоді, коли $\gamma_1 = \delta_1\alpha\delta_2$, $\gamma_2 = \delta_1\beta\delta_2$ для деяких ланцюжків δ_1 та δ_2, що належать Σ^*. Змістовно це правило дозволяє замінити підланцюжок γ_1, який збігається з лівою частиною правила, на праву його частину, отримуючи ланцюжок γ_2.

Нехай P – множина продукцій. Тоді кожна продукція з P може розглядатися як правило виведення на Σ^*. Сама послідовність правил, використаних у процесі породження деякого ланцюжка, є його виведенням. Визначена таким чином граматика є формальною системою. Відомими прикладами формальних систем служать логічні числення (числення висловлювань, числення предикатів), які детально вивчаються у відповідних розділах математичної логіки. Низку формальних систем для програм буде розглянуто нижче.

З наведених міркувань випливає таке означення.

Означення 2.14. *Породжувальною граматикою* (граматикою типу 0) називається четвірка $G=(N, T, P, S)$, де N і T – скінченні алфавіти, $N \cap T = \varnothing$, $P \subset (N \bigcup T)^* N (N \bigcup T)^* \times (N \bigcup T)^*$, P скінченна та $S \in N$. Тут:

- N – *нетермінальний алфавіт* (допоміжний), його елементи називаються нетермінальними символами, нетерміналами, змінними;
- T – *термінальний алфавіт* (основний), його елементи називаються термінальними символами, або терміналами;
- P – *множина продукцій*. Продукцію $(\alpha,\beta) \in P$ інколи називають правилом підстановки, правилом виведення або просто правилом і записують $\alpha \to \beta$;
- S – *початковий символ* (аксіома).

Зробимо кілька зауважень до наведеного означення. По-перше, слід пам'ятати, що вказана четвірка є просто екстенсіональним (одиничним) об'єктом поняття породжувальної граматики і без посилань на інтенсіональні властивості граматик не є самодостатньою. Оскільки для нас важливою є єдність екстенсіональних та інтенсіональних аспектів, то вважатимемо, що й у визначенні граматики така єдність має бути зазначена. По-друге, умова, що ліва частина правил належить множині $(N \bigcup T)^* N (N \bigcup T)^*$, означає, що в лівій частині обов'язково повинен бути нетермінал. Це обмеження фактично індукується інтенсіональною ознакою породження, оскільки воно не виводиться із завершеного (термінального) ланцюжка. Разом з тим у лівій частині можуть фігурувати термінальні символи, які можуть замінюватись на символи з правої частини правила. Це дозволяє

говорити, що означення породжувальної граматики не достатньо підтримує розподіл символів на термінальні й нетермінальні. Як уже зазначалось, від такого часткового рішення можна відмовитись, розглядаючи системи Туе (де немає розподілу на термінальні та нетермінальні символи) або контекстні граматики (де дозволена заміна лише нетермінального символу, але в певному контексті). По-третє, множини T, N, P були визначені як скінченні. Проте більшість результатів, що стосуються породжувальних граматик, можуть бути перенесені на нескінченні множини.

У прикладах нетермінальні символи зазвичай позначаємо великими літерами, термінальні – маленькими; для правил з однаковими лівими частинами $\alpha \to \beta_1, ..., \alpha \to \beta_n$ часто будемо використовувати скорочений запис $\alpha \to \beta_1 | ... | \beta_n$. Крім того, у прикладах інколи будемо задавати граматику не як відповідну четвірку, а просто як список правил, вважаючи, що алфавіт N складається з усіх великих, а алфавіт T – з усіх маленьких літер, що зустрічаються в правилах. При цьому лівою частиною першого правила є початковий символ S.

Повторимо ще раз означення відношення *безпосередньої вивідності*, яке задає граматика $G = (N, T, P, S)$.

Означення 2.15. Нехай задано граматику $G = (N, T, P, S)$. Пишемо $\gamma_1 \Rightarrow_G \gamma_2$ ($\gamma_1, \gamma_2 \in (N \bigcup T)^*$), якщо $\gamma_1 = \delta_1 \alpha \delta_2$, $\gamma_2 = \delta_1 \beta \delta_2$ для деяких слів $\delta_1, \delta_2 \in (N \bigcup T)^*$ та $\alpha \to \beta \in P$.

Коли з контексту зрозуміло, про яку граматику йдеться, замість $\Rightarrow_G$ можна писати просто $\Rightarrow$.

Означення 2.16. Рефлексивне транзитивне замикання відношення безпосередньої вивідності називається *відношенням вивідності* та позначається через $\Rightarrow^*_G$ (або $\overset{*}{\to}_G$).

Нагадаємо, що *рефлексивним транзитивним замиканням* бінарного відношення R на множині ST є найменше відношення R', яке містить R, є рефлексивним і транзитивним, тобто:

1) $sR's$ для всіх s із ST;

2) якщо s_1Rs_2 та $s_2R's_3$, то $s_1R's_3$.

Наведене означення не вказує на спосіб побудови рефлексивного транзитивного замикання. Разом з тим із властивостей рефлексивного транзитивного замикання випливає, що ланцюжок γ_n виводиться з ланцюжка γ_0 ($\gamma_0 \Rightarrow^*_G \gamma_n$) тоді й тільки тоді, коли деяка послідовність (можливо, порожня) замін лівих частин продукцій з P їхніми правими частинами переводить ланцюжок γ_0 в γ_n, інакше кажучи, коли існує послідовність вигляду $\gamma_0 \Rightarrow_G \gamma_1 \Rightarrow_G ... \Rightarrow_G \gamma_n$ ($n \geq 0$).

Означення 2.17. Послідовність ланцюжків $\gamma_0,\gamma_1,...\gamma_n$ така, що $\gamma_{i-1} \Rightarrow_G \gamma_i$ для $1 \leq i \leq n$, називається *виведенням* γ_n з γ_0 в G. Число n називається довжиною (кількістю кроків) цього виведення.

Зауважимо, що для довільного ланцюжка γ має місце $\gamma \Rightarrow^*_G \gamma$ (адже можливе виведення довжиною 0). Слід також зазначити, що виведення $\gamma_0,\gamma_1,...\gamma_n$ однозначно не описує, які саме правила граматики були застосовані до яких підланцюжків. Якщо така інформація важлива, то можна застосувати розмічене виведення, указуючи входження лівої частини правила, що замінюється, і позначення правила, що було застосоване.

Означення 2.18. Ланцюжки, що виводяться з певного нетерміна-ла A, називаються його *словоформами*, або *сентенційними формами*.

Приклад 2.11. Для граматики з правилами {$S \to aBSc$, $S \to \varepsilon$, $B \to \varepsilon$} та аксіомою S словоформами будуть ланцюжки $aBSc$, $aBaBaBSccc$, $aaaBccc$ та ін., які виводяться із S.

Центральним є означення, яке задає мову, що породжується граматикою.

Означення 2.19. *Мова, що породжується граматикою* G, – це множина ланцюжків $L(G) = \{w \mid S \Rightarrow^*_G w,\ w \in T^*\}$. Будемо також казати, що граматика G породжує мову $L(G)$.

Говорячи просто, мова $L(G)$ є мовою, що породжується граматикою G, якщо вона складається з ланцюжків (слів) у термінальному алфавіті, які виводяться з аксіоми S.

Інтенсіональні аспекти цього означення були обговорені раніше. У попередньому підрозділі для побудови моделей формальних мов використовувався підхід від загального (абстрактного) до конкретного (одиничного). Спочатку ми вводили поняття мови як множини речень, а в кінці побудов – поняття породжувальної граматики. Це був інтенсіональний аналіз поняття граматики. У цьому підрозділі зробимо навпаки: від екстенсіонального означення породжувальної граматики через поняття виведення та його абстракції перейдемо до поняття мови, що породжується граматикою (екстенсіональні побудови). Тут фактично можна говорити про визначення знизу-вверх: від індивідуальних понять – до загальних.

З цих міркувань випливає, що поняття породжувальної граматики треба розглядати як єдність екстенсіоналу, що визначається четвірками $G = (N, T, P, S)$ з указаними раніше параметрами, та інтенсіоналу, що дає загальне визначення відношення безпосередньої вивідності $\Rightarrow$, його рефлексивного транзитивного замикання $\Rightarrow^*$ і мови, що породжується, за формулою $L(G) = \{w \mid S \Rightarrow^*_G w, w \in T^*\}$.

Приклад породжувальної граматики та її властивості

Проілюструємо прикладом основні означення та методи доведення властивостей граматик і породжуваних мов.

Нехай задана граматика $G3 = (\{S,B\}, \{a,b,c\}, P, S)$, де

$P = \{$

$P1: S \to aBSc,$

$P2: S \to \varepsilon,$

$P3: Ba \to aB,$

$P4: Bb \to bB,$

$P5: Bc \to bc$

$\}$

Тут $P1$, $P2$, $P3$, $P4$, $P5$ – мітки відповідних правил.

Щоб зрозуміти, яку мову породжує ця граматика, побудуємо кілька виведень. Будемо використовувати розмічені виведення. Маємо:

1. $\underline{S} \overset{P2}{\Rightarrow} \varepsilon$.
2. $\underline{S} \overset{P1}{\Rightarrow} aB\underline{S}c \overset{P2}{\Rightarrow} a\underline{Bc} \overset{P5}{\Rightarrow} abc$.
3. $\underline{S} \overset{P1}{\Rightarrow} aB\underline{S}c \overset{P1}{\Rightarrow} aBaB\underline{S}cc \overset{P2}{\Rightarrow} a\underline{Ba}Bcc \overset{P3}{\Rightarrow} aaB\underline{Bc}c \overset{P5}{\Rightarrow} aa\underline{Bb}cc \overset{P4}{\Rightarrow} aab\underline{Bc}c \overset{P5}{\Rightarrow} aabbcc$.
4. $\underline{S} \overset{P1}{\Rightarrow} aB\underline{S}c \overset{P1}{\Rightarrow} aBaB\underline{S}cc \overset{P1}{\Rightarrow} aBa\underline{Ba}BScccc \overset{P3}{\Rightarrow} a\underline{Ba}aBBScccc \overset{P3}{\Rightarrow} aa\underline{Ba}BBScccc \overset{P3}{\Rightarrow} aaaBBB\underline{S}ccc \overset{P2}{\Rightarrow} aaaBB\underline{Bc}cc \overset{P5}{\Rightarrow} aaaB\underline{Bb}ccc \overset{P4}{\Rightarrow} aaa\underline{Bb}Bccc \overset{P4}{\Rightarrow} aaabB\underline{Bc}cc \overset{P5}{\Rightarrow} aaab\underline{Bb}ccc \overset{P4}{\Rightarrow} aaabb\underline{Bc}cc \overset{P5}{\Rightarrow} aaabbbccc$.

Отже, у цих виведеннях породжені ланцюжки ε, abc, $aabbcc$, $aaabbbccc$. Це дозволяє висунути припущення, що граматика $G3$ породжує мову $L3 = \{a^n b^n c^n \mid n \geq 0\}$. Доведемо, що це дійсно так.

Теорема 2.1. $L(G3) = \{a^n b^n c^n \mid n \geq 0\}$.

Спочатку доведемо, що $L(G3) \subseteq \{a^n b^n c^n \mid n \geq 0\}$. Це доведення базується на лемі, яка формулює загальний вигляд ланцюжків (словоформ) в об'єднаному алфавіті, що виводяться в $G3$.

Лема 2.1. Нехай $S \Rightarrow^*_{G3} \alpha$ ($\alpha \in (N \cup T)^*$). Тоді існує $n \geq 0$ таке, що:

1) $\alpha = \beta\gamma c^n$, де $\beta \in \{a,B\}^*$, $\gamma = S$ або $\gamma \in \{b,B\}^*$;

2) $|\alpha|_a = |\alpha|_c = |\alpha|_{\{b,B\}} = n$.

Доведення леми: індукція за довжиною виведення.

База індукції. Для виведення довжиною 0 маємо, що $n = 0$, $\alpha = S$. Отже, α має вигляд $\beta\gamma c^0$ ($\beta = \varepsilon$, $\gamma = S$, $c^0 = \varepsilon$), тому $|\alpha|_a = |\alpha|_c = |\alpha|_{\{b,B\}} = 0$. Лема виконується.

Крок індукції. Нехай лема виконується для всіх виведень довжиною $k \geq 0$. Доведемо, що вона виконується для виведень довжиною $k+1$.

Візьмемо довільне виведення $S \Rightarrow_{G3} \alpha_1 \Rightarrow_{G3} \ldots \Rightarrow_{G3} \alpha_k \Rightarrow_{G3} \alpha_{k+1}$. Особливість виведень полягає в тому, що початкова послідовність $S \Rightarrow_{G3} \alpha_1 \Rightarrow_{G3} \ldots \Rightarrow_{G3} \alpha_k$ буде виведенням довжиною k, і тому за індуктивним припущенням для α_k виконується твердження леми. Інакше кажучи, є деяке $n \geq 0$ таке, що $\alpha_k = \beta\gamma c^n$ та $|\alpha_k|_a = |\alpha_k|_c = |\alpha_k|_{\{b,B\}} = n$. Безпосереднє виведення $\alpha_k \Rightarrow_{G3} \alpha_{k+1}$ здійснюється за допомогою одного з правил $P1$, $P2$, $P3$, $P4$, $P5$. Розглянемо ці правила по черзі.

1. Нехай виведення $\alpha_k \Rightarrow_{G3} \alpha_{k+1}$ відбулося із застосуванням правила $P1$, тобто $\alpha_k \overset{P1}{\Rightarrow}_{G3} \alpha_{k+1}$. Тоді α_k можна зобразити у вигляді βSc^n ($\beta \in \{a,B\}^*$), а $\alpha_{k+1} = \beta aBSc^{n+1}$. Тому α_{k+1} має вигляд $\beta' Sc^{n+1}$ ($\beta' \in \{a,B\}^*$), як того вимагає лема, до того ж $|\alpha_{k+1}|_a = |\alpha_{k+1}|_c = |\alpha_{k+1}|_{\{b,B\}} = n+1$. Для цього випадку лему доведено.

2. Нехай виведення $\alpha_k \Rightarrow_{G3} \alpha_{k+1}$ відбулося із застосуванням правила $P2$, тобто $\alpha_k \overset{P2}{\Rightarrow}_{G3} \alpha_{k+1}$. Тоді α_k можна зобразити у вигляді βSc^n ($\beta \in \{a,B\}^*$) та $\alpha_{k+1} = \beta c^n$. Тому α_{k+1} можна подати у вигляді $\beta\gamma c^n (\gamma = \varepsilon)$, як того вимагає лема, до того ж $|\alpha_{k+1}|_a = |\alpha_{k+1}|_c = |\alpha_{k+1}|_{\{b,B\}} = n$. Для цього випадку лему доведено.

3. Нехай виведення $\alpha_k \Rightarrow_{G3} \alpha_{k+1}$ відбулося із застосуванням правила $P3$. Оскільки α_k можна зобразити у вигляді $\beta\gamma c^n$, то правило $P3$ може бути застосовано лише для підланцюжка β. Це правило не змінює ані загального вигляду α_{k+1}, ані кількості символів, тому $|\alpha_{k+1}|_a = |\alpha_{k+1}|_c = |\alpha_{k+1}|_{\{b,B\}} = n$. Для цього випадку лему доведено.

4. Нехай виведення $\alpha_k \Rightarrow_{G3} \alpha_{k+1}$ відбулося із застосуванням правила $P4$. Оскільки α_k можна зобразити у вигляді $\beta\gamma c^n$, то правило $P4$ може бути застосоване лише для підланцюжка γ (хоча, можливо, замінюване B є останнім символом β). Застосування правила не змінює ані загального вигляду α_{k+1}, ані кількості символів, тому $|\alpha_{k+1}|_a = |\alpha_{k+1}|_c = |\alpha_{k+1}|_{\{b,B\}} = n$. Для цього випадку лему доведено.

5. При застосуванні правила $P5$ твердження леми також залишається справедливим.

Лему доведено. ■

Нехай тепер $S \Rightarrow^*_{G3} t$ ($t \in T^*$). Оскільки t – ланцюжок у термінальному алфавіті, то він не містить нетерміналів, і за твердженням леми існує $n \geq 0$ таке, що $t = \beta\gamma c^n$, де $\beta \in a^*$, $\gamma \in b^*$ і $|t|_a = |t|_c = |t|_b = n$. Звідси випливає, що $t = a^n b^n c^n$. Інакше кажучи, будь-який термінальний ланцюжок, що виводиться в $G3$, належить мові $L3$, тобто $L(G3) \subseteq L3$.

Доведемо тепер зворотне включення, тобто $L3 \subseteq L(G3)$. Для цього продемонструємо, як побудувати виведення ланцюжка $a^n b^n c^n$ для довільного n. Використаємо індукцію за n.

База індукції. Ланцюжок $a^0 b^0 c^0$ $(= \varepsilon)$ отримуємо виведенням $\underline{S} \overset{P2}{\Rightarrow} \varepsilon$.

Крок індукції. Нехай існує виведення слова $a^n b^n c^n$ ($n \geq 0$), тобто $S \Rightarrow^*_{G3} a^n b^n c^n$. Доведемо, що існує виведення слова $a^{n+1} b^{n+1} c^{n+1}$, тобто $S \Rightarrow^*_{G3} a^{n+1} b^{n+1} c^{n+1}$.

Відповідне виведення будуємо таким чином. Спочатку до аксіоми застосуємо перше правило, а потім виконаємо виведення слова $a^n b^n c^n$. Отримаємо виведення: $\underline{S} \overset{P1}{\Rightarrow} aB\underline{S}c \Rightarrow^*_{G3} aBa^n b^n c^n c$. Далі n разів застосовуємо правило $P3$. Отримаємо ланцюжок $aa^n Bb^n c^n c (= a^{n+1} Bb^n c^{n+1})$. Після цього n разів застосовуємо правило $P4$. Отримали ланцюжок $a^{n+1} b^n Bc^{n+1}$. Нарешті, застосуванням правила $P5$ отримуємо ланцюжок $a^{n+1} b^{n+1} c^{n+1}$.

Теорему 2.1 доведено. ■

Наступним кроком у вивченні породжувальних граматик буде їх класифікація.

Ієрархія граматик Хомського

Дослідження будь-якого класу предметів зазвичай починається з їх класифікації. Це методологічне зауваження стосується й граматик. Однією з перших була класифікація, запропонована Н. Хомським. Суть її полягає в поступових обмеженнях, що накладаються на праві та ліві частини продукцій. Визначені Хомським чотири типи граматик виявились інваріантними щодо різних уточнень породжувальних граматик. Крім того, цим

типам граматик природно відповідають типи автоматів, що розпізнають (сприймають) формальні мови.

Означення 2.20.

- Граматиками *типу* 0 називають довільні породжувальні граматики загального вигляду, що не мають жодних обмежень на правила виведення.
- Граматиками *типу* 1 (*нескорочувальними*) називають породжувальні граматики, кожне правило яких має вигляд $\alpha\to\beta$, де $|\alpha|\le|\beta|$ ($\alpha\in(N\cup T)^*N(N\cup T)^*$, $\beta\in(N\cup T)^+$).
- Граматиками *типу* 2 (*контекстно-вільними*, *КВ*-граматиками) називають породжувальні граматики, кожне правило яких має вигляд $A\to\beta$, де $A\in N$, $\beta\in(N\cup T)^*$.
- Граматиками *типу* 3 (*праволінійними*) називають породжувальні граматики, кожне правило яких має вигляд $A\to\alpha B$ або $A\to\alpha$, де A, $B\in N$, $\alpha\in T\cup\{\varepsilon\}$. До граматик *типу* 3 відносять і *ліволінійні* граматики, кожне правило яких має вигляд $A\to B\alpha$ або $A\to\alpha$, де A, $B\in N$, $\alpha\in T\cup\{\varepsilon\}$.

Приклад 2.12. Граматика $G3$ з попереднього підрозділу є граматикою типу 0. Якщо з цієї граматики видалити правило $P2$, то вона стане граматикою типу 1, оскільки не буде скорочувальних правил. Граматика з правилами $\{S\to AR,\ R\to bRc,\ R\to\varepsilon,\ A\to aA,\ A\to\varepsilon\}$ є контекстно-вільною (типу 2), а її підграматика $\{A\to aA,\ A\to\varepsilon\}$ є праволінійною граматикою (типу 3).

Кожному типу граматик відповідають мови, яким будемо приписувати той самий тип, що має граматика, яка її породжує. На. 2.1 зображено співвідношення типів граматик.

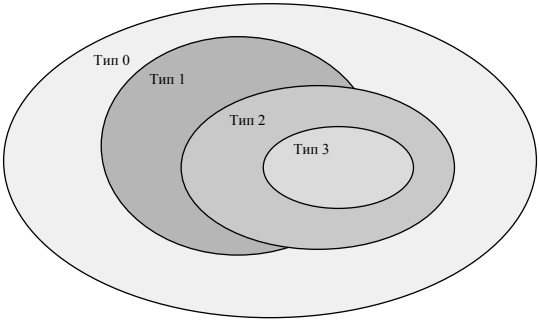


Рис. 2.1. Співвідношення типів граматик

Як бачимо, граматики типів 2 та 3 не є підкласами граматик типу 1. Це викликано тим, що наведені класи граматик мають скорочувальні правила вигляду $A\to\varepsilon$, які заборонені в граматиках типу 1. Як буде показано далі, ці відмінності не дуже позначаються на класах мов, породжуваних граматиками типів 2 та 3, і якщо ігнорувати порожній ланцюжок, то класи мов типів 3 та 2 будуть підкласами мов типу 1.

Перш ніж переходити до вивчення класів мов, покажемо, що без втрати виразної сили можна розглядати обмеженіший клас граматик, які називаються *загально-контекстними*. Цей клас є природнішим для породження, ніж загальні породжувальні граматики.

Означення 2.21.

- *Загально-контекстними* називають породжувальні граматики, кожне правило яких має вигляд $\kappa_1 A\kappa_2\to\kappa_1\beta\kappa_2$, де $A\in N$, κ_1, κ_2, $\beta\in(N\cup T)^*$. Ланцюжки κ_1 та κ_2 називають *лівим* і *правим контекстом* символу A в указаному правилі.
- *Контекстно-залежними* називаються загально-контекстні граматики з нескорочувальними правилами, тобто правилами вигляду $\kappa_1 A\kappa_2\to\kappa_1\beta\kappa_2$, де $\beta\in(N\cup T)^+$ (це означає, що $|\beta|\geq 1$).

Контекстно-залежні граматики інколи називають граматиками *безпосередньо складових* (БС-граматиками).

Слід сказати, що назва контекстно-залежних граматик не зовсім вдала, оскільки не підкреслює нескоротність правил граматики, а характеризує радше довільний клас контекстних граматик. Те саме стосується й назви граматик безпосередньо складових.

Граматики типів 2 та 3 (будучи безконтекстними) безпосередньо є підкласами загально-контекстних граматик.

З означень видно, що класи загально-контекстних і контекстно-залежних граматик є власними підкласами відповідно граматик типів 0 та 1. Разом з тим відповідні класи мов збігаються, тобто використання лише контекстних правил не обмежує породжувальну здатність граматик.

Доведемо цей факт збіжності мов.

Нехай $G_0 = (N, T, P, S)$ – довільна породжувальна граматика типу 0. Побудуємо еквівалентну їй загально-контекстну граматику. Алгоритм побудови такий.

1. Виділяємо термінальні символи, які входять до неконтекстних правил, і вводимо нетермінали, які дублюють ці термінальні символи, тобто нові нетермінали N_a для кожного такого термінального символу $a \in T$.

2. Далі всі неконтекстні правила з P замінюємо на нові, у яких замість термінальних символів присутні їх нетермінальні дублери. Додаємо нові правила $N_a \to a$. Очевидно, що отримана граматика G' еквівалентна початковій.

3. Побудуємо граматику G'', яка буде контекстною та еквівалентною G'. Спочатку пронумеруємо всі неконтекстні правила граматики G' (контекстні не змінюємо). Розглянемо неконтекстне правило G' з номером k вигляду $\alpha \to \beta$, де $\alpha = A_1A_2\ldots A_{n-1}A_n$, $n>1$. Це правило замінимо на множину нових правил:

$A_1A_2\ldots A_{n-1}A_n \to N_{k1}A_2\ldots A_{n-1}A_n$

$N_{k1}A_2\ldots A_{n-1}A_n \to N_{k1}A_2\ldots A_{n-1}N_{kr}$

$N_{k1}A_2\ldots A_{n-1}\,N_{kr} \to N_{k1}A_3\ldots A_{n-1}N_{kr}$

$N_{k1}A_3\ldots A_{n-1}\,N_{kr} \to N_{k1}A4\ldots A_{n-1}N_{kr}$

… … …

$N_{k1}A_{n-1}N_{kr} \to N_{k1}N_{kr}$

$N_{k1}N_{kr} \to N_{kp}N_{kr}$

$N_{kp}N_{kr} \to N_{kp}\beta$

$N_{kp} \to \varepsilon$

Коментар. Ідея побудови наведених правил полягає в такому: спочатку виставляємо лівий N_{kl} та правий N_{kr} маркери вибраного правила, далі видаляємо всі інші символи лівої частини продукції, потім переходимо до породження правої частини, але перед цим вносимо про це інформацію за допомогою N_{kp}. Наприкінці породжуємо в контексті N_{kp} праву частину продукції, а сам символ N_{kp} видаляємо.

Можна довести, що вказана послідовність правил еквівалентна одному початковому правилу. Дійсно, ця послідовність індукує (майже детерміновано) однозначне виведення. Тільки правило $N_{kp}\to\varepsilon$ порушує однозначність. Однак якщо його застосувати до правила $N_{kp}N_{kr}\to N_{kp}\beta$, то не буде можливості позбавитися нетермінала N_{kr}. Тому справедлива така теорема.

Теорема 2.2. Для довільної породжувальної граматики G_0 існує граматика G_C у загально-контекстній формі, яка породжує ту саму мову, тобто $L(G_0)=L(G_C)$.

Означення 2.22. Довільні граматики G_1 та G_2 називаються *еквівалентними*, якщо вони породжують одну й ту саму мову, тобто $G_1 \approx G_2 \Leftrightarrow L(G_1)=L(G_2)$.

Цілком очевидно, що таким чином введене відношення $\approx$ є відношенням еквівалентності (рефлексивним, симетричним і транзитивним).

Приклад 2.13. Побудуємо за граматикою $G3$, що породжує мову $L(G3)=\{a^n b^n c^n | n\geq 0\}$, еквівалентну їй загально-контекстну граматику $G3_C$.

Граматика $G3$ має такі правила:
$P1$: $S \to aBSc$;
$P2$: $S \to \varepsilon$;
$P3$: $Ba \to aB$;
$P4$: $Bb \to bB$;
$P5$: $Bc \to bc$.

З них неконтекстними є правила $P3$ та $P4$. Спочатку перетворимо правило $Ba \to aB$ (правило $P3$) на послідовність контекстних правил.

$Ba \to aB$ замінюємо на $BA_a \to A_aB$, $A_a \to a$.

Далі замість $BA_a \to A_aB$ вводимо послідовність правил

$BA_a \to N_{31}A_a$;

$N_{31}A_a \to N_{31}N_{3r}$;

$N_{31}N_{3r} \to N_{3p}N_{3r}$;

$N_{3p}N_{3r} \to N_{3p}A_aB$;

$N_{3p} \to \varepsilon$.

Аналогічно вчинимо з правилом $P4$: $Bb \to bB$. Після цих перетворень отримуємо нову граматику $G3_C$, в якій пронумеруємо нові правила:

$P1$: $S \to A_aBSc$.

$P2$: $S \to \varepsilon$.

$P31$: $A_a \to a$.

$P32$: $BA_a \to N_{3l}A_a$.

$P33$: $N_{3l}A_a \to N_{3l}N_{3r}$.

$P34$: $N_{3l}N_{3r} \to N_{3p}N_{3r}$.

$P35$: $N_{3p}N_{3r} \to N_{3p}A_aB$.

$P36$: $N_{3p} \to \varepsilon$.

$P41$: $B_b \to b$.

$P42$: $BB_b \to N_{4l}B_b$.

$P43$: $N_{4l}B_b \to N_{4l}N_{4r}$.

$P44$: $N_{4l}N_{4r} \to N_{4p}N_{4r}$.

$P45$: $N_{4p}N_{4r} \to N_{4p}B_bB$.

$P46$: $N_{4p} \to \varepsilon$.

$P5$: $Bc \to B_bc$.

Побудуємо кілька виведень у цій граматиці, тобто протестуємо побудовану граматику. Очевидно, що

$\underline{S} \overset{P2}{\Rightarrow} \varepsilon$ та $\underline{S} \overset{P1}{\Rightarrow} A_aB\underline{S}c \overset{P2}{\Rightarrow} A_a\underline{Bc} \overset{P5}{\Rightarrow} \underline{A_a}B_bc \overset{P31}{\Rightarrow} a\underline{B_b}c \overset{P41}{\Rightarrow} abc$.

Побудуємо виведення ланцюжка $a^2b^2c^2$. Маємо

$$\underline{S} \overset{P1}{\Rightarrow} A_aB\underline{S}c \overset{P1}{\Rightarrow} A_aBA_aB\underline{S}cc \overset{P2}{\Rightarrow} A_aBA_a\underline{Bc}c \overset{P5}{\Rightarrow} A_a\underline{BA_a}B_bcc \overset{P32}{\Rightarrow}$$

$$A_a\,\underline{N_{3l}}\underline{A_a}B_bcc \overset{P33}{\Rightarrow} A_a\,\underline{N_{3l}}\underline{N_{3r}}B_bcc \overset{P34}{\Rightarrow} A_a\,\underline{N_{3p}}\underline{N_{3r}}B_bcc \overset{P35}{\Rightarrow}$$

$$A_a\,\underline{N_{3p}}\,A_aBB_bcc \overset{P36}{\Rightarrow} A_aA_a\underline{BB_b}cc \overset{P42}{\Rightarrow} A_aA_a\underline{N_{4l}}\underline{B_b}cc \overset{P43}{\Rightarrow} A_aA_a\underline{N_{4l}}\underline{N_{4r}}cc \overset{P44}{\Rightarrow}$$

$$A_aA_a\underline{N_{4p}}\underline{N_{4r}}cc \overset{P45}{\Rightarrow} A_aA_aN_{4p}B_bBcc \overset{P46}{\Rightarrow} A_aA_aB_b\underline{Bc}c \overset{P5}{\Rightarrow} A_aA_aB_b\underline{B_b}cc \overset{P41}{\Rightarrow}$$

$$A_aA_a\underline{B_b}bcc \overset{P41}{\Rightarrow} A_a\underline{A_a}bbcc \overset{P31}{\Rightarrow} \underline{A_a}abbcc \overset{P31}{\Rightarrow} aabbcc.$$

Еквівалентність нескорочувальних і контекстно-залежних граматик доводиться аналогічно. Це дозволяє казати, що різні варіанти означень граматик (породжувальні та контекстні) еквівалентні, а ієрархія Хомського досить стабільна щодо варіантів граматик.

Перейдемо до дослідження зв'язків класів породжувальних граматик із класами формалізмів розпізнання (сприйняття), у першу чергу – класами автоматів різного типу.

Автоматні формалізми сприйняття мов

Дуальним методом щодо породження мов є їх сприйняття (розпізнавання). Цей метод також може уточнюватися на основі поняття транзиційної системи. Відмінність полягає в тому, що початковий стан містить ланцюжок, для якого потрібно перевірити його належність до певної мови, а заключний стан свідчить про належність (або неналежність) до мови.

Існують різні уточнення методів сприйняття. Одне з них можна отримати з породжувальних граматик простим оберненням застосування правил, тобто замість правила породжувальної граматики $\alpha\to\beta$ розглядати правило сприймаючої граматики $\beta\to\alpha$. Виведення у сприймаючій граматиці тоді треба вести від термінального ланцюжка до аксіоми. Такий метод уточнення нічого суттєво нового не привносить, хоча він часто розглядається в теорії формальних граматик як метод висхідного аналізу. Тому важливо визначити інші формалізми сприйняття мов. З цією метою часто обирають формалізми автоматів (машин) різного типу.

Автомати характеризуються тим, що відбувається розподіл на інформацію, яка зберігається в пам'яті певного типу, і таку, що керує процесом функціонування автомата та задається його керуючим

пристроєм.. Такий розподіл корисний тим, що робить функціонування автомата детермінованішим. Інакше кажучи, формалізм автоматів більше пристосований до керування виведенням, ніж формалізм граматик, де керування виведенням дуже слабке.

Автомат загалом має пам'ять (зазвичай це потенційно нескінченна стрічка), пристрій керування (задається скінченною множиною станів) і керуючу голівку, яка працює з певним елементом пам'яті. Головним для автомата є спосіб його функціонування, який задається переходами зі стану в стан, зміною пам'яті та рухом голівки. Ці переходи залежать від поточного стану керування та змісту елемента пам'яті, на який "дивиться" голівка.

Є різні варіанти визначень автоматів. Розглянемо ті з них, що відповідають класифікації мов за Хомським.

Найпотужнішими за сприймаючою силою є автомати, що називаються *машинами Тьюрінга*.

Машини Тьюрінга. Машина Тьюрінга M складається з таких частин.

1. Керуючий пристрій, який може набувати певного стану зі скінченної множини Q.

2. Стрічка (потенційно) нескінченної довжини, розділена на комірки, в яких розміщена вхідна інформація у вигляді символів деякого алфавіту A (зазвичай $Q \cap A = \varnothing$). У кожній комірці записано по одному символу з алфавіту. Виділяємо особливий символ $\# \in A$, який інтерпретуємо як порожній, причому в кожен даний момент стрічка містить лише скінченну кількість символів, відмінних від #. Вважаємо також, що всі непорожні символи записані підряд.

3. Голівка читання-запису забезпечує обмін інформацією між стрічкою й керуючим пристроєм. У кожний момент часу голівка може працювати тільки з однією коміркою стрічки. Голівка може:

- замінити прочитаний символ іншим символом алфавіту A;
- переміщуватись на одну позицію праворуч або ліворуч;
- залишатися на місці.

Робота машини задається спеціальними правилами переходу (командами), що називаються *програмою* машини.

Програма складається з команд вигляду $qa\to pbR$, $qa\to pbL$, $qa\to pb$ (q, $p\in Q$, a, $b\in A$, L,R – додаткові символи). *Інтерпретація команд* така:

- Виконання команди $qa\to pbR$. Якщо керуючий пристрій перебуває у стані q, а голівка зчитує комірку на стрічці, у якій записано символ a, то керуючий пристрій переходить у стан p, голівка в комірку записує символ b і сама зміщується на одну комірку праворуч.
- Виконання команди $qa\to pbL$. Відрізняється від попереднього лише тим, що голівка зміщується ліворуч.
- Виконання команди $qa\to pb$. Відрізняється від попередньої команди тим, що голівка залишається на місці.

Надалі не будемо акцентувати увагу на керуючому пристрої та голівці й казатимемо просто, що машина Тьюрінга змінює стан і рухається ліворуч або праворуч.

У фіксований момент часу поточна інформація задається за допомогою *конфігурації* машини Тьюрінга. Конфігурацією машини Тьюрінга зазвичай називають трійку (q, $\#w$, $v\#$), де $q\in Q$, w,$v\in A^*$. Конфігурація описує повний стан машини та інтерпретується таким чином:

- q – стан керуючого пристрою;
- $\#w$ – ланцюжок, записаний на стрічці машини ліворуч від голівки;
- $v\#$ – ланцюжок, записаний на стрічці машини праворуч від голівки, включаючи символ, який зчитує голівка.

Зважаючи на те, що будемо перетворювати команди машини Тьюрінга на правила граматики, конфігурації далі будемо задавати у вигляді ланцюжка $\#wqv\#$ (тому тут важливо, щоб $Q\cap A=\varnothing$).

Оскільки машина Тьюрінга є конкретизацією транзиційної системи, то в означення вводимо початковий стан $q_0\in Q$ і множину заключних станів $F\subseteq Q$. Відношення безпосереднього виведення $\Rightarrow$ (часто позначається $|\!-$) і рефлексивне транзитивне замикання цього відношення $\Rightarrow^*$ ($|\!-^*$) вводимо стандартно. Введені поняття дозволяють формально означити машини Тьюрінга.

Означення 2.23. *Машиною Тьюрінга M* називається послідовність параметрів $(Q, A, \#, \delta, q_0, F)$, де:

- Q – скінченна множина станів;
- A – скінченний алфавіт ($Q \cap A = \varnothing$);
- # – символ з A (порожній символ);
- δ – скінченна множина команд $qa \to pbR$, $qa \to pbL$, $qa \to pb$ ($q, p \in Q$, $a, b \in A$, L, R – додаткові символи);
- q_0 – початковий стан із Q;
- F – підмножина фінальних станів із Q.

Зауважимо, що є багато варіантів означення машин Тьюрінга, наприклад, може бути кілька стрічок, порожній символ можна явно не вводити в алфавіт і вважати його однаковим для класу машин Тьюрінга, не вводити заключних станів тощо. Усі такі означення зазвичай еквівалентні з погляду сприйняття мов.

Наведене означення задає екстенсіонал поняття машини Тьюрінга, тобто клас одиничних машин. Що стосується інтенсіоналу, то він обумовлений призначенням машин, орієнтованих на сприйняття мов, яке засноване на відношенні $\vdash_T$ ($\Rightarrow_T$) безпосереднього переходу від однієї конфігурації до іншої.

Означення 2.24. *Конфігурацією* машини Тьюрінга $M = (Q, A, \#, \delta, q_0, F)$ називається ланцюжок вигляду $\#wqv\#$, $q \in Q$, $w, v \in A^*$.

Означення 2.25. Нехай $M = (Q, A, \#, \delta, q_0, F)$ – машина Тьюрінга. Конфігурація $\#wcqav\#$ *безпосередньо переходить*:

- у конфігурацію $\#wcbpv\#$, якщо виконується команда $qa \to pbR$;
- у конфігурацію $\#wpcbv\#$, якщо виконується команда $qa \to pbL$;
- у конфігурацію $\#wcpbv\#$, якщо виконується команда $qa \to pb$ ($a, b, c \in A$, $w, v \in A^*$, $q, p \in Q$).

Відношення безпосереднього переходу також називатимемо відношенням *виведення конфігурацій*, щоб продемонструвати його єдність із відношенням виведення в породжувальних граматиках і формальних системах. Послідовність конфігурацій, пов'язаних відношенням безпосереднього виведення, називається *протоколом* машини Тьюрінга.

Означення 2.26. Рефлексивне транзитивне замикання відношення $|\!-_T$ позначаємо $|\!-^*_T$.

Початкові конфігурації мають вигляд $\#q_0v\#$, фінальні – $\#w_1q_Fw_2\#$, $q_F \in F$.

І, нарешті, головне смислове означення.

Означення 2.27. *Мова, яка допускається машиною Тьюрінга* $M = (Q, A, \#, \delta, q_0, F)$, – це множина ланцюжків

$L_T(M) = \{w \mid \#q_0w\# |\!-^*_T \#w_1q_F w_2\#, q_F \in F, w, w_1, w_2 \in A^*\}$.

Таким чином, поняття машини Тьюрінга – це єдність двох аспектів: екстенсіонального та інтенсіонального. Екстенсіональний аспект задається як клас машин Тьюрінга, визначених відповідними шістками параметрів; інтенсіональний – уніформним означенням відношення безпосередньої вивідності та мови, що сприймається (допускається, розпізнається) машиною.

Приклад 2.14. Побудуємо машину Тьюрінга, яка допускає мову $\{ r\tilde{r} \mid r \in \{a,b\}^*\}$.

Ідея програми:

1. У початковому стані q_0 машина читає символ a, b або #. У перших двох випадках машина стирає прочитані символи та запам'ятовує прочитаний символ відповідно у стані q_a або q_b. Останній випадок означає, що на вхідній стрічці – порожній ланцюжок і тому машина переходить у заключний стан q_F.

2. Далі голівка рухається у правий бік до кінця ланцюжка. Прочитавши порожній символ #, голівка повертається на одну комірку (клітинку) ліворуч і переходить у стан q_{-a} зі стану q_a або у стан q_{-b} зі стану q_b. Тут індекси $-a$ та $-b$ означають, що машині потрібно стерти відповідно символ a або b.

3. Якщо правий символ ланцюжка збігається із символом, що запам'ятався як такий, який потрібно стерти, то він стирається, машина переходить у стан q_L; якщо ні, то робота машини блокується (немає застосовних правил) і вхідний ланцюжок не належить розглядуваній мові.

4. У стані q_L голівка рухається ліворуч, поки не дійде до символу #. Далі голівка переходить у стан q_0 та зміщується у правий бік. Один цикл перевірки завершено. Робота продовжується наступним циклом перевірки або завершується, якщо все перевірено.

Наведений алгоритм дозволяє побудувати машину Тьюрінга

$M = (\{q_0, q_a, q_{-a}, q_b, q_{-b}, q_L, q_F\}, \{a, b\}, \#, \delta, q_0, \{q_F\})$,

де δ – множина таких команд:

$q_0a \to q_a\#R$;
$q_0b \to q_b\#R$;
$q_aa \to q_aaR$;
$q_ab \to q_abR$;
$q_a\# \to q_{-a}\#L$;
$q_aa \to q_L\#L$;
$q_ba \to q_baR$;
$q_bb \to q_bbR$;
$q_b\# \to q_{-b}\#L$;
$q_{-b}b \to q_L\#L$;
$q_La \to q_LaL$;
$q_Lb \to q_LbL$;
$q_L\# \to q_0\#R$;
$q_0\# \to q_F\#$.

Розглянемо на прикладі сприйняття слів цією машиною Тьюрінга. Візьмемо слово #*abba*#. Побудуємо протокол його сприйняття:

$\#q_0abba\# \Rightarrow \#q_abba\# \Rightarrow \#bq_aba\# \Rightarrow \#bbq_aa\# \Rightarrow \#bbaq_a\# \Rightarrow$
$\#bbq_{-a}a\# \Rightarrow \#bbq_L\# \Rightarrow \#bq_Lb\# \Rightarrow \# q_Lbb\# \Rightarrow q_L\#bb\# \Rightarrow$
$\# q_0bb\# \Rightarrow \# q_bb\# \Rightarrow \# bq_b\# \Rightarrow \#q_{-b}b\# \Rightarrow \#q_L\# \Rightarrow \#q_0\# \Rightarrow \#q_F\#$

Еквівалентність машин Тьюрінга та породжувальних граматик. Наведений у попередньому прикладі протокол машини Тьюрінга підказує ідею побудови породжувальної граматики за програмою машини. Таку побудову виконаємо у два етапи: спочатку за програмою побудуємо продукції переходу (перетворення) конфігурацій машини Тьюрінга, потім здійс-

нимо обернення побудованих продукцій і додамо правило для аксіоми та правила породження й видалення порожніх символів. Отримана породжувальна граматика буде еквівалентна вибраній машині Тьюрінга.

Кожна команда машини Тьюрінга породжує такі правила перетворення конфігурацій:

- Команда вигляду $qa \to pbR$ породжує правило $qa \to bp$.
- Команда вигляду $qa \to pbL$ породжує множину правил перетворення $\{ cqa \to pcb \mid c \in A \}$.
- Команда вигляду $qa \to pb$ породжує правило $qa \to pb$.

На другому етапі побудови обернемо отримані правила (перетворення вигляду $\alpha \to \beta$ замінимо на правило породжувальної граматики $\beta \to \alpha$), додамо правило для аксіоми $S \to \#q_F\#$. Ураховуючи, що за таких побудованих простих правил можуть бути зайві #, переведемо їх у порожній ланцюжок ε за правилом $\# \to \varepsilon$ або створимо в разі необхідності додаткові порожні символи # за правилом $\# \to \#\#$. Нарешті, потрібно буде видалити початковий стан, щоб отримати (породити) початковий ланцюжок за правилом $q_0 \to \varepsilon$. Таким чином створимо граматику $G_M = (N, T, P, S)$, яка має такі параметри:

- $N = \{S\} \cup Q \cup \{\#\}$ $(S \notin Q \cup \{\#\})$;
- $T = A \setminus \{\#\}$;
- $S \in N$;
- P будується за вказаним вище алгоритмом.

Проілюструємо наведений алгоритм прикладом.

Приклад 2.15. Візьмемо машину Тьюрінга, запропоновану в попередньому прикладі, і побудуємо еквівалентну їй граматику. Отримаємо граматику $G_M = (\{S, q_0, q_a, q_{-a}, q_b, q_{-b}, q_L, q_F, \#\}, \{a, b\}, P, S)$.

Потрібні перетворення команд наведено в табл. 2.1, третій стовпчик якої задає продукції P.

Таблиця 2.1

Команди машини Тьюрінга	Правила перетворення конфігурацій	Продукції породжувальної граматики
		0. $S \to \#q_F\#$ $0_{\#}$. $\# \to \#\#$
1. $q_0a \to q_a\#R$	1. $\#q_0a \to \#q_a$	1. $\#q_a \to \#q_0a$
2. $q_0b \to q_b\#R$	2. $\#q_0b \to \#q_b$	2. $\#q_b \to \#q_0b$
3. $q_aa \to q_aaR$	3. $q_aa \to aq_a$	3. $aq_a \to q_aa$
4. $q_ab \to q_abR$	4. $q_ab \to bq_a$	4. $bq_a \to q_ab$
5. $q_a\# \to q_{-a}\#L$	5. $aq_a\# \to q_{-a}a\#$	5. $q_{-a}a\# \to aq_a\#$
6. $q_{-a}a \to q_L\#L$	6. $aq_{-a}a \to q_La\#$; $bq_{-a}a \to q_Lb\#$; $\#q_{-a}a \to q_L\#\#$;	6. $q_La\# \to aq_{-a}a$; $q_Lb\# \to bq_{-a}a$; $q_L\#\# \to \#q_{-a}a$
7. $q_ba \to q_baR$	7. $q_ba \to aq_b$	7. $aq_b \to q_ba$
8. $q_bb \to q_bbR$	8. $q_bb \to bq_b$	8. $bq_b \to q_bb$
9. $q_b\# \to q_{-b}\#L$	9. $bq_b\# \to q_{-b}b\#$	9. $q_{-b}b\# \to bq_b\#$
10. $q_{-b}b \to q_L\#L$	10. $aq_{-b}b \to q_La\#$; $bq_{-b}b \to q_Lb\#$; $\#q_{-b}b \to q_L\#\#$;	10. $q_La\# \to aq_{-b}b$; $q_Lb\# \to bq_{-b}b$; $q_L\#\# \to \#q_{-b}b$
11. $q_La \to q_LaL$	11. $aq_La \to q_Laa$; $bq_La \to q_Lba$; $\#q_La \to q_L\#a$	11. $q_Laa \to aq_La$; $q_Lba \to bq_La$; $q_L\#a \to \#q_La$;
12. $q_Lb \to q_LbL$	12. $aq_Lb \to q_Lab$; $bq_Lb \to q_Lbb$; $\#q_Lb \to q_L\#b$	12. $q_Lab \to aq_Lb$; $q_Lbb \to bq_Lb$; $q_L\#b \to \#q_Lb$;
13. $q_L\# \to q_0\#R$ $q_0\# \to q_F\#$	13. $q_L\# \to \#q_0$ $\#q_0\# \to \#q_F\#$	13. $\#q_0 \to q_L\#$ 14. $\#q_F\# \to \#q_0\#$ 15. $q_0 \to \varepsilon$ 16. $\# \to \varepsilon$

Правила, отримані перетворенням команди з номером n ($n = 6, 10, 11, 12$), будемо далі нумерувати як $n(1)$, $n(2)$, $n(3)$.

Продемонструємо коректність правил виведення для породження ланцюжка *abba* (індекси вказують на застосоване правило):

$\underline{S} \Rightarrow_0 \underline{\#q_F\#} \Rightarrow_{14} \underline{\#q_0\#} \Rightarrow_{13} \#q_L\underline{\#} \Rightarrow_{0\#} \#\underline{q_L\#\#} \Rightarrow_{10(3)} \underline{\#\#}q_{-b}b\# \Rightarrow_{16} \#\underline{q_{-b}b\#} \Rightarrow_9 \#\underline{bq_b\#} \Rightarrow_8 \#\underline{q_bb\#} \Rightarrow_{0\#} \#\underline{\#q_0}b\# \Rightarrow_{13} \#q_L\underline{\#bb\#} \Rightarrow_{12(3)} \underline{\#\#}q_Lbb\# \Rightarrow_{16} \#\underline{q_Lbb\#} \Rightarrow_{12(2)} \#bq_Lb\underline{\#} \Rightarrow_{0\#} \#bq_Lb\#\# \Rightarrow_{6(2)} \#bbq_{-a}\underline{a\#} \Rightarrow_5 \#bb\underline{aq_a\#} \Rightarrow_3 \#b\underline{bq_a}a\# \Rightarrow_4 \#\underline{bq_a}ba\# \Rightarrow_4 \#q_abba\# \Rightarrow_1 \underline{\#q_0}abba\# \Rightarrow_{16} q_0abba\underline{\#} \Rightarrow_{16} q_0abba$.

Зауважимо, що в процесі виведення застосовувались правила породження та знищення порожнього символу #.

Наведені вище міркування та приклад дозволяють сформулювати таке твердження.

Теорема 2.3. За кожною машиною Тьюрінга M можна побудувати еквівалентну їх породжувальну граматику G, що породжує ту саму мову, яку сприймає M, тобто $L_T(M) = L(G)$.

Ця теорема може бути обернена.

Теорема 2.4. За кожною породжувальною граматикою G можна побудувати еквівалентну їй машину Тьюрінга M, що сприймає ту саму мову, яку породжує G, тобто $L_T(M) = L(G)$.

Доведення може бути конструктивним, коли машиною Тьюрінга моделюється зворотний до породження процес сприйняття ланцюжка або можна скористатися тезою Чорча, що стверджує існування машини Тьюрінга для формального подання інтуїтивних алгоритмів. Деталі такої побудови описувати не будемо.

З наведених теорем випливає, що граматики типу 0 породжують усі рекурсивно-зліченні множини (мови) в алфавіті A. Тим самим клас породжувальних граматик має таку саму виразну потужність, як і інші універсальні моделі алгоритмів.

Зазначимо, що є різні варіанти машин Тьюрінга, наприклад машини з різною кількістю стрічок, різними обмеженнями на команди. Зокрема, виділяють детерміновані (не існує команд з однаковими лівими й різними правими частинами) і недетерміновані машини Тьюрінга (існують команди з однаковими лівими й різними правими частинами). Клас формальних мов, що сприймаються машинами Тьюрінга, однаковий для детермінованих і недетермінованих машин та машин з різною кількістю стрічок.

Лінійно обмежені автомати. Лінійно обмежені автомати можуть розглядатися як обмежені машини Тьюрінга, для яких довжина ланцюжка на стрічці завжди лінійно обмежена довжиною вхідного ланцюжка.

Означення 2.28. Машина Тьюрінга $M=(Q, A, \#, \delta, q_0, F)$ називається *лінійно обмеженим автоматом*, якщо існує число k таке, що для будь-якого ланцюжка $v \in A^*$ довжиною n з умови $\#q_0v\# \vdash^* \#w_1q\ w_2\#$ ($q \in Q$, $w_1, w_2 \in A^*$) випливає, що $|w_1\ w_2| \le k \cdot n$.

Наведену умову важко перевірити, маючи машину Тьюрінга, тому наведене означення часто спрощують, вимагаючи, щоб $k=1$, тобто щоб голівка машини Тьюрінга не могла записувати нові символи за межами вхідного ланцюжка. Є також інші варіанти означення лінійно обмежених автоматів.

Використовуючи методи, наведені в попередньому підрозділі, можемо за лінійно обмеженим автоматом побудувати еквівалентну породжувальну граматику типу 1, і навпаки, за граматикою типу 1 – лінійно обмежений автомат. Отже, справедливе таке твердження.

Теорема 2.5. За кожним лінійно обмеженим автоматом можна побудувати еквівалентну йому породжувальну граматику типу 1, і навпаки, за кожною породжувальною граматикою типу 1 можна побудувати еквівалентний їй лінійно обмежений автомат.

Простіше кажучи, теорема стверджує, що клас лінійно обмежених автоматів еквівалентний класу породжувальних граматик типу 1.

Магазинні автомати. Магазинні автомати (автомати з магазинною пам'яттю, МП-автомати, стекові автомати) широко використовуються в програмуванні, оскільки дозволяють моделювати різні важливі алгоритми, пов'язані з компіляцією та інтерпретацією мов програмування, обробкою складних структур даних – дерев, списків тощо. Такі автомати можна розглядати як обмежені машини Тьюрінга, що мають вхідну стрічку, на якій голівка може тільки читати символи й рухатися тільки в один бік, і робочу стрічку, яку відповідна голівка може обробляти лише з одного боку. Як і в попередніх випадках, є різні варіанти означень магазинних автоматів. Наведемо просте екстенсіональне означення, яке разом з тим є еквівалентним іншим означенням за класом мов, що розпізнаються.

Означення 2.29. Магазинний автомат – це шістка $M=(Q, A, \Gamma, \delta, q_0, F)$, де:

- Q – скінченна множина станів;
- A – скінченний вхідний алфавіт ($Q \cap A = \varnothing$);
- Γ – скінченний магазинний алфавіт;
- δ – скінченне відношення переходів (скінченне відображення $\delta: Q \times A \times \Gamma \to Q \times \Gamma^*$);
- q_0 – початковий стан із Q;
- F – підмножина фінальних (заключних) станів із Q.

Виконання команди вигляду $(q, a, Z) \to (p, \gamma)$ означає, що, перебуваючи у стані q та зчитуючи символ a на вхідній стрічці та символ Z у магазині, автомат переходить у стан p, стирає символ a та пересуває голівку до наступного символу на вхідній стрічці, а замість символу Z записує в магазин ланцюжок γ. Зауважимо, що в інших модифікаціях магазинних автоматів перехід у новий стан може відбуватися без стирання символу зі вхідної стрічки (ε-перехід).

Конфігурацію магазинного автомата можна задати як трійку (q, w, ζ), де $q \in Q$, $w \in A^*$, $\zeta \in \Gamma^*$. Відношення безпосереднього переходу $\vdash$ задається так: $(q, aw', Z\zeta') \vdash (p, w', \gamma\zeta')$, якщо команда $(q, a, Z) \to (p, \gamma)$ належить δ.

Аналогічно тому, як це робилося раніше, визначаються також інші необхідні поняття, пов'язані з магазинним автоматом: протоколи й мова, що сприймається таким автоматом.

Для нас важливим є таке твердження.

Теорема 2.6. За кожним магазинним автоматом можна побудувати еквівалентну йому породжувальну граматику типу 2 (контекстно-вільну), і навпаки, за кожною породжувальною граматикою типу 2 можна побудувати еквівалентний їй магазинний автомат.

Зважаючи на те, що синтаксис мов програмування зазвичай задається граматиками типу 2, теорема стверджує, що алгоритми обробки програм можуть базуватися на магазинних автоматах.

Скінченні автомати. Скінченні автомати можна розглядати як обмежені машини Тьюрінга, що мають одну вхідну стрічку – голівка може тільки читати символи й рухатися тільки в один бік. Тому такі автомати не мають необмеженої пам'яті. Екстенсіональне означення таке.

Означення 2.30. Скінченний автомат – це п'ятірка $M=(Q, A, \delta, q_0, F)$, де:

- Q – скінченна множина станів;
- A – скінченний вхідний алфавіт ($Q \cap A = \varnothing$);
- δ – скінченне відношення переходів (скінченне відображення $\delta: Q \times A \to Q$);
- q_0 – початковий стан із Q;
- F – підмножина фінальних (заключних) станів із Q.

Усі основні загальні (інтенсіональні) поняття визначаються подібно до того, як це було зроблено для машин Тьюрінга й магазинних автоматів, тому наводити означення не будемо.

Основним є такий результат.

Теорема 2.7. За кожним скінченним автоматом можна побудувати еквівалентну йому породжувальну граматику типу 3 (ліволінійну чи праволінійну), і навпаки, за кожною породжувальною граматикою типу 3 можна побудувати еквівалентний їй скінченний автомат.

Зазначимо, що детермінованість чи недетермінованість скінченного автомата не впливає на клас мов, що сприймаються автоматами.

Теорія скінченних автоматів розроблена досить детально, для них побудовані алгоритми еквівалентних перетворень, мінімізації тощо. Наведемо лише один результат, що стосується алгебраїчного подання скінченноавтоматних мов. Такі мови ще називаються регулярними.

Регулярні мови задаються виразами (термами) регулярної алгебри мов, що має операції об'єднання, конкатенації та ітерації.

Регулярні вирази можна визначити індуктивно.

Базис складається з трьох означень.

1. Константи ε та Ø є регулярними виразами.
2. Якщо a – довільний символ алфавіту, то a – регулярний вираз. Зауважимо, що часто в написанні розрізняють символ алфавіту й відповідний вираз. Тут це не робимо, щоб не ускладнювати текст.
3. Якщо X – змінна, то X – регулярний вираз.

Крок індуктивної побудови. Індуктивний крок складається з чотирьох означень, по одному для трьох операторів і для введення дужок.

1. Якщо E та F – регулярні вирази, то $E+F$ – регулярний вираз.
2. Якщо E та F – регулярні вирази, то EF – регулярний вираз. Для позначення операції конкатенації над мовами можна використовувати крапку.
3. Якщо E – регулярний вираз, то E^* – регулярний вираз.
4. Якщо E – регулярний вираз, то (E) – регулярний вираз.

Інтерпретація L виразів у класі мов над алфавітом A задається також індуктивно на підставі інтерпретації змінних L_V: $Var \to 2^{A^*}$, де Var – множина змінних.

Інтерпретація базових виразів задається таким чином:

1. $L(\varepsilon) = \{\varepsilon\}$ і $L(\text{Ø}) = \text{Ø}$.
2. $L(a) = \{a\}$.
3. $L(X) = L_V(X)$.

Інтерпретація складних виразів задається таким чином (E та F – регулярні вирази):

1. $L(E+F) = L(E) \cup L(F)$.
2. $L(EF) = L(E)L(F)$.
3. $L(E^*) = (L(E))^*$.
4. $L((E)) = L(E)$.

Використовуватимемо $L(E)$ для позначення мови, яка відповідає E. Як і в інших алгебрах, оператори регулярних виразів мають певні пріоритети, тобто оператори зв'язуються зі своїми операндами в певному порядку.

Найвищий пріоритет має оператор ітерації *. Далі йде оператор конкатенації. Оскільки він асоціативний, то не має значення, в якому порядку групуються послідовні конкатенації. Однак, якщо необхідно групувати вирази, то робимо це, починаючи

зліва. Найнижчий пріоритет має оператор об'єднання. Він також асоціативний. Для нього будемо притримуватися групування, починаючи з лівого краю виразу.

Основний результат стосовно регулярних мов такий.

Теорема 2.8. За кожним скінченним автоматом можна побудувати еквівалентний йому регулярний вираз, і навпаки, за кожним регулярним виразом можна побудувати еквівалентний йому скінченний автомат.

Наведений результат дозволяє використовувати різні формалізми (граматики типу 3, скінченні автомати, регулярні вирази) для дослідження властивостей регулярних мов. Такі мови мають хороші властивості замкненості (щодо об'єднань, перетину, доповнень, конкатенації, ітерації й деяких інших операцій). Ще одну важливу групу властивостей регулярних мов становлять властивості розв'язності. За їх допомогою можна з'ясувати, чи визначають два різні автомати одну й ту саму мову. Розв'язність цієї задачі дозволяє мінімізувати автомати, тобто за даним автоматом знайти еквівалентний йому з найменшою можливою кількістю станів. Задача мінімізації має велике значення при проектуванні інтегральних схем.

Методи подання синтаксису мов програмування

Граматики типу 2 (контекстно-вільні) відіграють велику роль у формалізації мов програмування. Вони задають чітке подання деякого синтаксичного поняття як структури, що складається з певних частин. Ті частини, у свою чергу, теж мають частини і т. д., тобто контекстно-вільні граматики подають ієрархічну організацію синтаксичного поняття. Це відповідає й композиційній структурі програм. Саме тому вивченню цього класу граматик буде приділено більшу увагу. Щоб продемонструвати зв'язок з мовами програмування чіткіше, розглянемо методи подання синтаксису мов програмування. Найвідомішим методом є нормальні форми Бекуса – Наура (БНФ). Цей формалізм (метамова) широко використовується як під час подання синтаксису мов програмування, так і під час вивчення природних мов.

Нормальні форми Бекуса – Наура. Ці форми були запропоновані Дж. Бекусом і П. Науром для опису синтаксису мови програмування АЛГОЛ-60. Основним призначенням БНФ було подання в компактному вигляді строго формальних правил написання основних конструкцій мов програмування.

Приблизно в той самий час Ноам Хомський (1959) ввів аналогічну форму – контекстно-вільну граматику – для означення синтаксису природної мови.

БНФ складається зі скінченної множини правил, які визначають формальну мову (зокрема, синтаксис мов програмування).

Одним із класів об'єктів, що використовуються в БНФ, є символи описуваної мови. Другим класом об'єктів БНФ виступають метазмінні (нетермінальні символи), значеннями яких є ланцюжки основних символів. Для опису синтаксису мов програмування використовують велику кількість нетермінальних символів, тому для наочності їх подають як комбінації слів природної мови, що поміщені в кутові дужки. Ліву частину правил відділяють від правої частини знаком ::=, крім того, правила з однаковою лівою частиною записують як одне правило, при цьому праві частини (альтернативи) розділяють вертикальною рискою |.

Приклад 2.16. Синтаксис операторів мови SIPL задається такою БНФ:

```
<оператор> ::=   <змінна>:=<вираз> |
                 <оператор> ; <оператор>|
                 if <умова> then <оператор> else <оператор> |
                 while <умова> do <оператор> |
                 begin <оператор> end |
                 skip
```

За кожною БНФ легко побудувати контекстно-вільну граматику, зіставляючи правилу БНФ вигляду $A ::= \alpha_1 \mid \alpha_2 \mid \ldots \alpha_n$ сукупність правил граматики $A \to \alpha_1$, $A \to \alpha_2$, … $A \to \alpha_n$.

Модифіковані нормальні форми Бекуса – Наура. Для отримання наочніших і компактніших описів синтаксису застосовують модифіковані БНФ. Найчастіше передбачається введення спеціальних позначень для ітерації та альтернативи.

Наприклад, список якихось елементів задається БНФ

<список>::=<елемент><список>| ε

У модифікованій БНФ можна записати

<список>::={<елемент>}*

Така форма передбачає введення операції, яка називається ітерацією й позначається парою фігурних дужок із зірочкою.

Якщо певна частина синтаксичної конструкції може бути пропущена, то в модифікованих БНФ її виділяють квадратними дужками. Наприклад, замість правила БНФ

<оператор> ::= if <умова> then <оператор> else <оператор>
| if <умова> then <оператор>

можна вжити таке правило модифікованої БНФ:

<оператор> ::= if <умова> then <оператор> [else <оператор>].

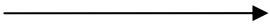
Зрозуміло, що для модифікованих БНФ вживати дужки " { ", " } ", " [" та "] ", а також " * " як символи мови не слід, тому що вони виступають як метасимволи формалізму модифікованих БНФ. За необхідності вживання цих дужок і зірочки для них використовують спеціальні позначення.

Синтаксичні діаграми. Для поліпшення зорового сприйняття й полегшення розуміння синтаксичних описів застосовують подання синтаксичних правил у вигляді синтаксичних діаграм. Такі діаграми мають одну вхідну й одну вихідну стрілки. Схематично їх можна зображувати таким чином (*D* – внутрішність діаграми):

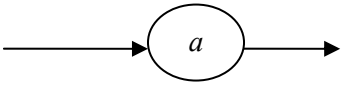
Синтаксичні діаграми є структурованими. Це означає, що вони визначаються індуктивно: є найпростіші діаграми, а більш складні будують за допомогою певних правил.

Найпростішими є такі діаграми:

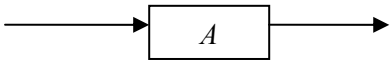
- *Порожня діаграма*:



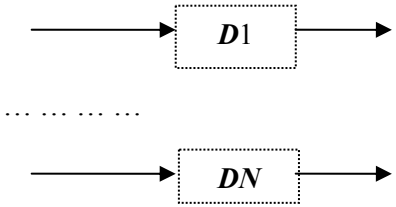
- *Термінальна діаграма* (*a* – термінальний символ):



- *Нетермінальна діаграма* (*A* – нетермінальний символ):

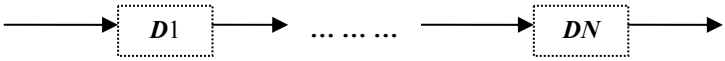


Далі побудуємо такі діаграми:

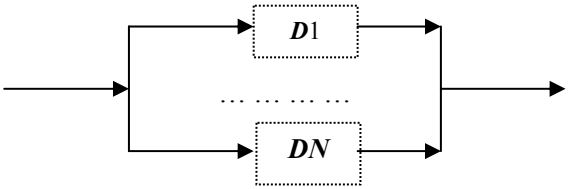


Є три правила побудови нових діаграм з наведених:

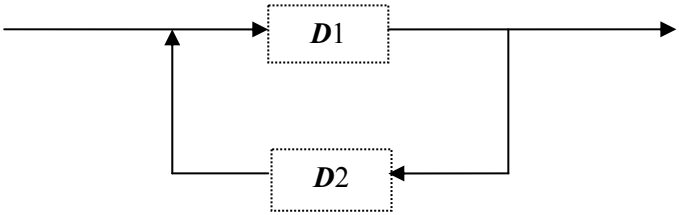
- *Послідовність*:



- *Альтернатива*:



- *Ітерація*:



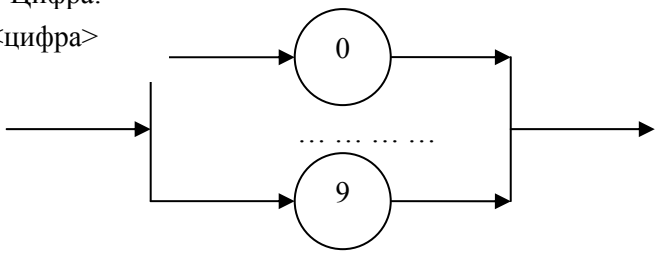
Діаграми можуть мати назву, яка є нетермінальним символом.

Сукупність іменованих діаграм задає певну формальну мову, кожне слово якої складається з термінальних символів, розташованих на шляху від початкової стрілки до заключної (від вхідної до вихідної стрілки), причому нетермінальні символи замінюються на термінальні слова, що задаються діаграмою, яка позначається таким нетермінальним символом.

Приклад 2.17. Наведемо кілька діаграм, які задають числа:

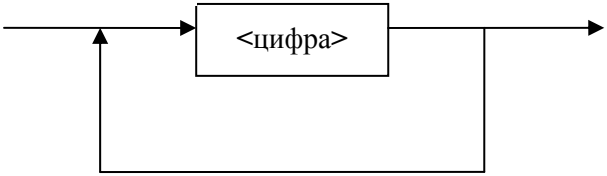
- Цифра:

<цифра>



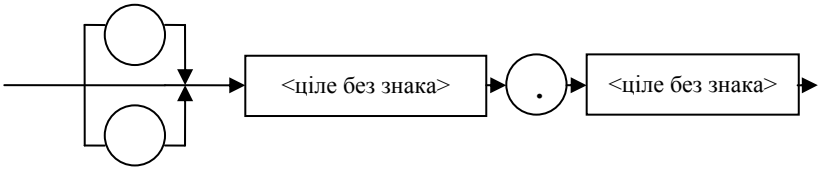
- Ціле без знака:

<ціле без знака>



- Дійсне число (без експоненти):

<дійсне число>



Щоб скоротити кількість діаграм, їх часто об'єднують, заміняючи нетермінальні символи, які входять у діаграму, на побудовані для них діаграми.

За кожною БНФ можна побудувати систему синтаксичних діаграм. Метод побудови такий (індукція за структурою БНФ):

- порожньому слову відповідає порожня діаграма;
- термінальному символу відповідає термінальна діаграма з цим символом;
- нетермінальному символу відповідає нетермінальна діаграма з цим символом;
- послідовності символів, що утворюють одну з альтернатив правої частини правила БНФ, відповідає послідовність діаграм, що задають символи цієї частини;
- альтернативам правила (тобто виразу $::=\alpha_1 \mid \alpha_2 \mid \ldots \alpha_n$) відповідає діаграма – альтернатива для діаграм, побудованих за виразами $\alpha_1, \alpha_2, \ldots, \alpha_n$;
- правилу $A::=\alpha_1 \mid \alpha_2 \mid \ldots \alpha_n$ відповідає деяка діаграма з іменем A, яка визначається для виразу $\alpha_1 \mid \alpha_2 \mid \ldots \alpha_n$.

Неважко побудувати й зворотний алгоритм, який за системою синтаксичних діаграм указаного вигляду будує еквіваленту їй БНФ.

Отже, справедливе таке твердження.

Твердження 2.1. Такі формалізми подання формальних мов: БНФ, модифіковані БНФ, контекстно-вільні граматики, синтаксичні діаграми, є еквівалентними.

Зауважимо, що наведене твердження не сформульовано як теорема лише тому, що не дано достатньо точного означення формальних мов, що породжуються такими формалізмами, як БНФ, модифіковані БНФ і синтаксичні діаграми. Разом з тим надати такі формальні означення не дуже складно (зробіть це самостійно).

Зважаючи на наведене твердження, будемо надалі серед указаних формалізмів розглядати переважно формалізм контекстно-вільних граматик.

Властивості контекстно-вільних граматик

У цьому підрозділі розглянемо властивості контекстно-вільних граматик (КВ-граматик), зокрема їх еквівалентні перетворення та нормальні форми. Спочатку доведемо лему про

еквівалентність граматик щодо бієктивного перейменування нетерміналів.

Лема 2.2 (про перейменування нетерміналів). Нехай задані граматика $G=(N, T, P, S)$ та бієктивне відображення ρ: $N\to N'$ множини нетерміналів N на деяку іншу множину нетерміналів N' ($N'\cap T=\varnothing$). Тоді для граматики $G'=(N', T, S', P')$, де P' – множина правил, отриманих із P заміною нетерміналів N на відповідні нетермінали з N', а $S'=\rho(S)$, маємо: $L(G)=L(G')$.

Для доведення леми слід скористатися тією обставиною, що кожне виведення в одній граматиці має відповідне виведення (шляхом перейменування нетерміналів) в іншій граматиці.

Зрозуміло, що умова бієктивності є суттєвою.

Видалення несуттєвих символів. Продовжимо розгляд очевидних, але важливих перетворень. У деяких випадках КВ-граматика може містити символи та правила, що не вживаються для виведення термінальних ланцюжків.

Приклад 2.18. У граматиці $G=(\{S, A, B\}, \{a, b, c\}, P, S)$, де $P=\{S\to a, S\to cS, A\to b\}$, нетермінал A і термінал b не можуть з'явитися в жодному ланцюжку виведення (у жодній словоформі). Таким чином, ці символи не беруть участі в породженні ланцюжків мови $L(G)$ і правила, що їх містять, можна видалити, не змінивши мови $L(G)$.

Означення 2.31. Нетермінал $A\in N\cup T$ назвемо *недосяжним* у граматиці $G=(N, T, P, S)$, якщо A не з'являється в жодному вивідному ланцюжку, тобто не існує виведення вигляду $S\Rightarrow_G{}^* \gamma A\delta$, γ, $\delta\in(N\cup T)^*$.

Для знаходження недосяжних нетерміналів спочатку визначимо множину досяжних нетерміналів. Ця множина для заданої КВ-граматики $G=(N, T, P, S)$ легко визначається за допомогою таких індуктивних означень.

1. $R_0=\{S\}$.
2. $R_i=\{B \mid$ існує правило $A\to\alpha B\beta\in P$, що $A\in R_{i-1}, B\in N\}\cup R_{i-1}$ ($i=1, 2, \ldots$).

Оскільки множина N скінченна, а формула для означення R_i задає монотонне за i відображення, то існує k ($k \geq 0$) таке, що $R_k = R_{k+1}$. Інакше кажучи, послідовність R_0, R_1, R_2, … стабілізується на k-му кроці. Покладемо $R = R_k$.

Множина UR недосяжних нетерміналів задається формулою $UR = N \setminus R$.

За граматикою $G = (N, T, P, S)$ будуємо граматику $G' = (N', T', P', S')$:

1. $N' = N \cap R$.
2. $T' = T$.
3. $S' = S$.
4. $P' = \{A \to \alpha \in P \mid \alpha \in (R \cup T)^*\}$.

Очевидною є така лема.

Лема 2.3. Для граматики $G' = (N', T', P', S')$ виконуються такі властивості:

1. $L(G') = L(G)$ (еквівалентність).
2. Для всіх $A \in N'$ існують такі ланцюжки α та β із $(N' \cup T)^*$, що $S \Rightarrow^*_{G'} \alpha A \beta$ (усі нетермінали є досяжними).

Означення 2.32. Нетермінал $A \in N$ назвемо *непродуктивним* у граматиці $G = (N, T, P, S)$, якщо з A не можна вивести жодного термінального ланцюжка, тобто не існує виведення вигляду $A \Rightarrow^*_G t$, $t \in T^*$.

Спочатку визначимо множину *продуктивних* нетерміналів. Множина продуктивних нетерміналів для заданої КВ-граматики $G = (N, T, P, S)$ легко визначається за допомогою таких індуктивних означень:

1. $Pr_0 = \{A \mid$ існує правило $A \to t \in P$, що $t \in T^*\}$.
2. $Pr_i = \{A \mid$ існує правило $A \to \gamma \in P$, що $\gamma \in (R_{i-1} \cup T)^*\} \cup R_{i-1}$ ($i = 1,2, \ldots$).

Оскільки множина N скінченна, а формула для означення Pr_i задає монотонне за i відображення, то існує k ($k \geq 0$) таке, що $Pr_k = Pr_{k+1}$. Інакше кажучи, послідовність Pr_0, Pr_1, Pr_2, … стабілізується на k-му кроці. Покладемо $Pr = Pr_k$.

Множина UPr непродуктивних нетерміналів задається формулою: $UPr = N \setminus Pr$.

За граматикою $G=(N, T, P, S)$ будуємо граматику $G'=(N', T', P', S')$:

1. $N'=(N\cap Pr)\cup\{S\}$.
2. $T'=T$.
3. $S'=S$.
4. $P'=\{A\to\alpha\in P \mid \alpha\in (Pr\cup T)^*\}$.

Очевидною є така лема.

Лема 2.4. Для граматики $G'=(N', T', P', S')$ виконуються такі властивості:

1. $L(G')=L(G)$ (еквівалентність).
2. Для всіх $A\in N'\setminus\{S\}$ існують термінальні ланцюжки, що виводяться з A.

Зазначимо, що аксіома S є спеціальним випадком, тому що вона може бути непродуктивною й разом з тим має належати множині нетермінальних символів (за означенням породжувальних граматик), але в такому випадку множина правил буде порожньою.

До речі, наведений метод побудови множини продуктивних нетерміналів дозволяє дати відповідь на питання, *чи є порожньою мова*, що породжується граматикою $G=(N, T, P, S)$.

Відповідь проста: якщо аксіома є продуктивним нетерміналом, то мова непорожня, якщо ж аксіома – непродуктивний нетермінал, то мова – порожня.

Означення 2.33. Символ $X\in N\cup T$ назвемо *несуттєвим* у КВ-граматиці $G=(N, T, P, S)$, якщо в ній немає виведення вигляду $S\Rightarrow^* wXy\Rightarrow^* wxy$, де w, x, y належать T^*.

З наведеного означення бачимо, що нетермінал X несуттєвий, якщо X є недосяжним або непродуктивним нетерміналом. Термінальний символ X несуттєвий, коли немає жодного породжуваного ланцюжка, що містить X.

Означення 2.34. Граматика $G=(N, T, P, S)$ називається *зведеною*, якщо в ній немає несуттєвих символів (можливо, крім аксіоми S).

Для побудови зведеної граматики спочатку видаляємо недосяжні й непродуктивні нетермінали. Потім із множини термінальних символів видаляємо несуттєві. Такі термінальні символи не містяться в правих частинах отриманих правил. Отримана граматика буде зведеною.

Лема 2.5. За кожною КВ-граматикою можна побудувати еквівалентну їй зведену граматику, що не містить несуттєвих символів.

Приклад 2.19. Розглянемо граматику $G=(\{S, A, B, C\}, \{a, b, c\}, P, S)$, де P складається з правил:

$S\to a$

$S\to A$

$A\to AB$

$B\to b$

$C\to c$

Спочатку видаляємо недосяжні нетермінали. Отримаємо $R=\{S, A, B\}$. Недосяжним є C. З граматики треба видалити вказаний символ та останнє правило. Далі визначаємо множину продуктивних нетерміналів. Знаходимо, що $Pr=\{S, B\}$. Непродуктивним є A. Після видалення відповідних правил нетермінал B стає недосяжним. Видаляємо також його. Залишається одне правило $S\to a$. Тому суттєвим термінальним символом є лише a. Видаляємо несуттєві термінальні символи. Отримуємо таку зведену граматику: $G'=(\{S\}, \{a\}, \{S\to a\}, S)$.

Видалення ε-правил. Означення 2.35. Назвемо КВ-граматику $G=(N, T, P, S)$ граматикою *без ε-правил* (або нескорочувальною), якщо P не містить ε-правил, тобто правил вигляду $A\to\varepsilon$.

Лема 2.6. За кожною КВ-граматикою можна побудувати еквівалентну їй (з точністю до порожнього ланцюжка – ε-еквівалентну) граматику без ε-правил.

Доведення. Нехай дана КВ-граматика $G=(N, T, P, S)$, що породжує мову $L(G)$. Проведемо серію перетворень множини P. Якщо множина P містить правила $B\to\alpha A\beta$ та $A\to\varepsilon$ для деяких A, $B\in N$, α, $\beta\in(N\cup T)^*$, то додамо правило $B\to\alpha\beta$ у P. Повторюватимемо цю процедуру, поки множина правил не стабілізується. Далі виключимо з множини P усі правила вигляду $A\to\varepsilon$.

Наведений алгоритм можна уточнити за допомогою таких індуктивних означень.

1. $P_0=P$.

2. $P_i=\{B\to\alpha\beta \mid$ існує правило $B\to\alpha A\beta\in P$, $A\to\varepsilon\in P_{i-1}\}\cup R_{i-1}$ $(i=1,2,\ldots)$.

Оскільки множина правил скінченна, а праві частини нових правил коротші, то формула для означення P_i задає монотонне за i відображення. Тому існує k ($k\geq 0$) таке, що $P_k=P_{k+1}$. Інакше кажучи, послідовність $P_0, P_1, P_2, \ldots$ стабілізується на k-му кроці. Покладемо $P'=P_k\setminus\{A\to\varepsilon\in P_k\}$.

Одержана граматика $G'=(N, T, P', S)$ породжує мову $L(G)\setminus\{\varepsilon\}$.

Дійсно, кожне виведення в граматиці G, яке не породжує порожнє слово, може бути промодельоване в граматиці G', що легко доводиться індукцією за довжиною виведення. Так само можна промоделювати виведення в G' виведеннями в G, використовуючи замість модифікованих правил вигляду $B\to\alpha\beta$ початкові правила вигляду $B\to\alpha A\beta$ з подальшим виведенням порожнього ланцюжка з A. ■

Приклад 2.20. Розглянемо граматику G із правилами

$S\to aSb$

$S\to bSa$

$S\to\varepsilon$

Застосувавши до цієї граматики описаний метод видалення ε-правил, отримаємо граматику G' із правилами

$S\to aSb$

$S\to bSa$

$S\to ab$

$S\to ba$,

яка еквівалентна з точністю до порожнього ланцюжка граматиці G.

2.2. Побудова мови за допомогою системи рівнянь з регулярними коефіцієнтами

Завдання 2.1. Нехай задано мову L, яку описує регулярний вираз $r = a^*ba^*$. Побудувати за регулярним виразом граматику й систему лінійних рівнянь, розв'язати систему.

Розв'язання.

1) Побудуємо праволінійну граматику G для мови L:

$G = <VT, VN, Pr, S>$.

$VT = \{a, b\}$; $VN = \{S, S_1, S_2\}$; $Pr = \{S \to aS_1;\ S \to bS_2;\ S_1 \to aS_1;\ S_1 \to bS_2;\ S_2 \to aS_2;\ S_2 \to \varepsilon\}$, $S = \{S\}$.

Отже,

$G = <\{a,\ b\};\ \{S,\ S_1,\ S_2\};\ \{S \to aS_1;\ S \to bS_2;\ S_1 \to aS_1;\ S_1 \to bS_2;\ S_2 \to aS_2;\ S_2 \to \varepsilon\},\ \{S\}>$

2) Тепер побудуємо систему лінійних рівнянь для цієї граматики. Кількість рівнянь дорівнює кількості нетерміналів (у нашому випадку 3). Кожному нетерміналу ставимо у відповідність змінну: $S \to X_1$; $S_1 \to X_2$; $S_2 \to X_3$. Система матиме вигляд

$X_1 = p_1 + q_{11}X_1 + q_{12}X_2 + q_{13}X_3$

$X_2 = p_2 + q_{21}X_1 + q_{22}X_2 + q_{23}X_3$

$X_3 = p_3 + q_{31}X_1 + q_{32}X_2 + q_{33}X_3$

Нехай p_i визначається так: для всіх продукцій вигляду $X_i \to \gamma_j$, $p_i = \sum_k \gamma_k$. Усі q_{ij} визначаються так: для всіх продукцій вигляду $X_i \to \gamma_j X_j$, $q_{ij} = \sum_k \gamma_k$. Маємо:

$X_1 = X_2a + X_3b$ (1)

$X_2 = X_2a + X_3b$ (2)

$X_3 = \varepsilon + X_3a$ (3)

Розв'язком рівняння вигляду $X = X\alpha + \beta$ буде регулярний вираз $\beta\alpha^*$. Систему рівнянь розв'язуємо методом Гаусса. Розв'язком системи вважаємо X_k – вираз при змінній, яка відповідає

нетерміналу аксіоми. У нашому випадку це буде регулярний вираз при змінній X_1. Робимо послідовні підстановки:

$X_1 = X_2a + X_3b$ підставляємо в усі рівняння. Рівняння $X_2 = X_2a + X_3b$ має розв'язок $X_2 = (X_3b)a^*$. Підставляємо X_2 в останнє рівняння. Маємо $X_3 = \varepsilon + X_3a$. Тепер робимо зворотну підстановку: розв'язком останнього рівняння буде $X_3 = \varepsilon a^* = a^*$. Підставляємо X_3 в перше та друге рівняння: $X_1 = X_2a + a^*b$; $X_2 = X_2a + a^*b$. Розв'язуємо друге рівняння: $X_2 = a^*ba^*$. Підставляємо в перше рівняння: $X_1 = a^*ba^*a + a^*b = a^*b(a^*a + \varepsilon) = a^*ba^*$. Отже, отримали регулярний вираз a^*ba^*, який описує ту саму мову, що й граматика, побудована за початковим регулярним виразом.

2.3. Побудова мови методом наближень

Завдання 2.2. Задано мову L, яку описує регулярний вираз $r = a^nba^n$. Побудувати граматику. Методом послідовних наближень побудувати мову.

Розв'язання.

Побудуємо породжувальну граматику G для мови L: $G = <\{a, b\}, \{S\}, \{S \to aSa, S \to \varepsilon\}, \{S\}>$. Покажемо, що мова, яку описує побудована граматика, складається зі слів вигляду a^nba^n і тільки з них.

Запишемо для цієї граматики рівняння: $S = \{b\} \cup \{a\}S\{a\}$. Тоді функція $\varphi(L_i) = \{b\} \cup \{a\}L_i\{a\}$.

Розглянемо L_0 як перше наближення мови L: $L_0 = \{b\}$. Застосовуючи метод послідовних наближень, маємо: $L_1 = \varphi(L_0) = \{b\} \cup \{a\}\{b\}\{a\} = \{b\} \cup \{aba\}$. Нехай $L_k = \bigcup_{i=0}^{k} \{a^iba^i\}$, тоді

$$L_{k+1} = \varphi(L_k) = \varphi(\bigcup_{i=0}^{k} \{a^iba^i\}) = \{b\} \cup \{a\} \bigcup_{i=0}^{k} \{a^iba^i\} \{a\} =$$

$$= \{b\} \cup \bigcup_{i=0}^{k+1} \{a^iba^i\} = \bigcup_{i=0}^{k+1} \{a^iba^i\}.$$

Очевидно, що послідовність (L_k) – зростаюча, отже, існує границя $L' = \bigcup_{k}^{\infty} L_k = \bigcup_{i=0}^{\infty} \{a^i b a^i\}$. Тоді L містить L' (це очевидно, оскільки для кожного слова w мови L' існує таке невід'ємне n, що слово $w = a^n b a^n$).

Покажемо зворотне включення. Візьмемо довільне слово w з мови L вигляду $w = a^n b a^n$. Покажемо, що воно належить мові L'. Це випливає з існування виведення такого слова в граматиці. Для виведення слова необхідно застосувати до аксіоми S правило $S \to aSa$ n разів, а потім правило $S \to b$ один раз. Отже, мови L та L' однакові, і граматика G задає мову L і тільки її.

2.4. Побудова граматики за мовою

Завдання 2.3. Нехай задано мову $L = \{a^n b^n | n >= 0\}$. Побудувати граматику й показати, що вона породжує всі слова мови й тільки їх.

Розв'язання

1) Побудуємо граматику G для мови L. $G = <VT, VN, Pr, S>$.

$VT = \{a, b\}$; $VN = \{S\}$; $Pr = \{S \to aSb; S \to \varepsilon\}$, $S = \{S\}$.

Отже, $G = <\{a, b\}; \{S\}; \{S \to aSb; S \to \varepsilon\}, \{S\}>$

2) Перевіримо, що граматика породжує *всі* слова мови.

Поділимо все, що породжує граматика, на типи (класи):

1) $a^k S b^k$.

2) $a^k b^k$, $k >= 0$.

Можливо, доведеться їх змінити. Адже нам треба отримати такі класи, щоб застосування правил виведення граматики не виводило нас за їх межі. Побудуємо таку таблицю:

Типи / Правила	1	2
1: $S \to aSb$	1	-
2: $S \to \varepsilon$	2	-

Бачимо, що застосування першого правила до першого типу залишає слово в першому типі; другого правила до першого типу – переводить його в другий тип.

Жодне правило не застосовне до слів другого типу (-).

Перший тип, перше правило: $a^k S b^k \to^1 a^k a S b b^k = a^{(k+1)} \mathrm{Sb}^{(k+1)}$.

Перший тип, друге правило: $a^kSb^k \to^2 a^kb^k$.

Таким чином, ми поділили все, що породжує граматика, на замкнені у своїх межах класи (усе, що вона породжує, не виходить за їх межі).

3) Порівняємо тепер вихідну мову та мову, породжену побудованою граматикою:

$(a^kSb^k) \cap VT^* = Ø$;

$(a^kb^k, k>=0) \cap VT^* = a^kb^k, k>=0$.

Отже, ця граматика породжує всі слова мови й тільки їх.

Завдання 2.4. Нехай задано мову $L = \{a^nb^nc^n | n>=0\}$. Побудувати граматику й показати, що вона породжує всі слова мови й тільки їх.

Розв'язання.

1) Побудуємо граматику G для мови L. $G = <VT, VN, Pr, S>$.

$VT = \{a, b, c\}$; $VN = \{S; B\}$; $Pr = \{S \to aBSc; S \to \varepsilon; Ba \to aB; Bc \to bc; Bb \to bb\}$, $S = \{S\}$.

Отже, $G = <\{a, b, c\}; \{S; B\}; \{S \to aBSc; S \to \varepsilon; Ba \to aB; Bc \to bc; Bb \to bb\}, \{S\}>$

2) Перевіримо, що граматика породжує *всі* слова мови.

Поділимо словоформи, що породжує граматика, на такі типи (класи):

1) $(aB)^nSc^n$;

2) yc^n, де $|y|_a = |y|_B = n, y = (a|B)^*$;

3) yc^n, де $|y|_a = |y|_b + |y|_B = n, y = (a|B)^*b^*b$.

Можливо, доведеться їх змінити. Адже поділ на класи має відбутися так, щоб застосування правил виведення граматики не виходило за межі цих класів.

Тепер побудуємо таку таблицю:

Типи / Правила	1	2	3
1: $S \to aBSc$	1	-	-
2: $S \to \varepsilon$	2	-	-
3: $Ba \to aB$	1	2	3
4: $Bc \to bc$	-	3	-
5: $Bb \to bb$	-	-	3

Застосування першого правила до першого типу залишає слово в першому типі; другого правила до першого типу – переводить його в другий тип. Випадки позначаємо таким чином: Тип. Правило). Наприклад, випадок застосування першого правила до словоформ першого типу позначимо 1.1).

1.1) $(aB)^nSc^n \to^1 (aB)^n aBScc^n = (aB)^{(n+1)}Sc^{(n+1)}$;

1.2) $(aB)^nSc^n \to^2 (aB)^n c^n = yc^n$, де $|y|_a = |y|_B = n$, $y = (a|B)^*$;

1.3) $(aB)^nSc^n \to^3 ySc^n$, де $|y|_a = |y|_B = n$, $y = (a|B)^*$.

Оскільки клас ySc^n, де $|y|_a = |y|_B = n$, $y = (a|B)^*$, який ми отримали внаслідок застосування правила 3 до першого класу, включає клас 1 ($(aB)^nSc^n$), то змінимо клас 1 на цей отриманий клас. Попередні правила й виведення не зміняться.

Правила 4 та 5 незастосовні до класу 1.

Правила 1, 2, 5 незастосовні до класу 2.

2.3) $yc^n \to^3 yc^n$, де $|y|_a = |y|_B = n$, $y = (a|B)^*$;

2.4) $yc^n \to^3 qc^n$, де $|y|_a = |y|_B = n$, $y = (a|B)^*$, $|q|_a = |q|_b + |q|_B = n$, $q = (a|B)^*b^*b$.

Правила 1, 2, 4 незастосовні до класу 3.

3.3) $yc^n \to^3 yc^n$, де $|y|_a = |y|_b + |y|_B = n$, $y = (a|B)^*b^*b$;

3.5) $yc^n \to^3 yc^n$, де $|y|_a = |y|_b + |y|_B = n$, $y = (a|B)^*b^*b$.

Таким чином, ми поділили все, що породжує граматика, на замкнені у своїх межах класи:

1) ySc^n, де $|y|_a = |y|_B = n$, $y = (a|B)^*$;

2) yc^n, де $|y|_a = |y|_B = n$, $y = (a|B)^*$;

3) yc^n, де $|y|_a = |y|_b + |y|_B = n$, $y = (a|B)^*b^*b$,

тобто все, що вона породжує, не виходить за їх межі.

3) Порівняємо тепер вихідну мову й мову, породжену побудованою граматикою:

(ySc^n, де $|y|_a = |y|_B = n$, $y = (a|B)^*) \cap T^* = \varnothing$;

(yc^n, де $|y|_a = |y|_B = n$, $y = (a|B)^*) \cap T^* = \{\varepsilon\}$;

(yc^n, де $|y|_a = |y|_b + |y|_B = n$, $y = (a|B)^*b^*b) \cap T^* = yc^n$, де $|y|_a = |y|_b + |y|_B = n$, $y = (a|B)^*b^*$, причому $|y|_B = 0$, тому $y = a^*b^*b = a^nb^n$.

Отже, (yc^n, де $|y|_a = |y|_b + |y|_B = n$, $y = (a|B)^*b^*b) \cap T^* = a^nb^nc^n$.

Таким чином, ця граматика породжує всі слова мови й тільки їх.

Завдання 2.5. Нехай задано мову $L = \{a^n b^m | m>n\}$. Побудувати граматику G, яка породжує мову L, і довести $L(G) = L$.

Розв'язання.

Побудуємо граматику G для мови L. $G = <VT, VN, Pr, S>$.

$VT = \{a, b\}$; $VN = \{S; B; C\}$; $Pr = \{S \to aSb; S \to B; B \to Cb; C \to Cb; C \to \varepsilon\}$; $S = \{S\}$.

Отже, $G = <\{a, b\}; \{S; B; C\}; \{S \to aSb; S \to B; B \to Cb; C \to Cb; C \to \varepsilon\}, \{S\}>$.

1) Граматика G для мови $L = \{a^n, b^m \mid m > n\}$ має такі правила виведення:

$S \to aSb$

$S \to B$

$B \to Bb$

$B \to b$

2) Доведемо, що $L(G) = L$.

У граматиці G можна побудувати такі типи слів:

1) $a^n Sb^n, n \geq 0.$

2) $a^n Bb^m, m \geq n.$

3) $a^n b^m, m > n.$

Покажемо, що це всі можливі типи слів, які породжуються в граматиці. Для цього відобразимо, як правила виведення перетворюють один тип на інший, таким чином показавши замкненість множини слів (точніше, їх типів) щодо виведення:

Типи слів / Правила виведення	1	2	3
1	1	-	-
2	2	-	-
3	-	2	-
4	-	3	-

Отже, очевидно, що в граматиці виводяться лише слова третього типу, оскільки лише вони складаються з терміналів.

Таким чином, $L(G) \subseteq L$ доведено. Доведемо тепер $L(G) \supseteq L$. Покажемо можливість виведення будь-якого слова з *L*.

Виведення слова $a^n b^m$ з аксіоми *S*:

правило 1) застосовується *n* разів;

правило 2) застосовується *m*–*n*–1 разів;

правило 3) застосовується один раз.

Взаємне включення множин *L*(*G*) та *L* доведене.

Завдання 2.6

1) Побудувати породжувальну граматику *G* для мови $L = \{a^n b^m c^n \mid m \geq n\}$.

2) Довести, що $L(G) = L$.

3) Довести, що мова *L* не є КВ-мовою.

Розв'язання.

1) Граматика *G* для мови $L = \{a^n b^m c^n \mid m \geq n\}$ має такі правила виведення:

$S \to aBSc$

$S \to BS$

$BS \to b$

$Ba \to aB$

$Bb \to bb$

2) Доведемо, що $L(G) = L$.

У граматиці *G* можна побудувати такі типи слів:

1) $\gamma Sc^n, \gamma = (a \mid B)^*, |\gamma|_a = n, |\gamma|_B = m, m \geq n$;

2) $\gamma bc^n, \gamma = (a \mid B)^*, |\gamma|_a = n, |\gamma|_B = m-1, m \geq n$;

3) $\gamma c^n, \gamma = (a \mid B)^* b^*, |\gamma|_a = n, |\gamma|_B + |\gamma|_b = m, m \geq n$;

4) $a^n b^m c^n, m \geq n$.

Покажемо, що це всі можливі типи слів, які породжуються в граматиці. Для цього відобразимо, як правила виведення перетворюють один тип на інший, таким чином показавши замкненість множини слів (точніше, їх типів) щодо виведення:

Типи слів / Правила виведення	1	2	3	4
1	1	-	-	-
2	1	-	-	-
3	2	-	-	-
4	1	2	3	-
5	-	3	3,4	-

Отже, очевидно, що в граматиці виводяться лише слова четвертого типу.

Таким чином, доведено $L(G) \subseteq L$. Доведемо тепер $L(G) \supseteq L$. Покажемо можливість виведення будь-якого слова з L.

Виведення слова $a^n b^m c^n$ з аксіоми S:

правило 1) застосовується n разів;

правило 2) застосовується $m–n$ разів;

правило 3) застосовується один раз;

правило 4) застосовується $(n–1)*(n–2)/2$ разів, якщо це можливо;

правило 5) застосовується $m–1$ разів.

3) Довести, що мова L не є КВ-мовою.

Якби мова L була КВ-мовою, то існували б ланцюжки u, v, t_1, $t_2, x \in \{a^*, b^{**}, c^*\}$ такі, що $ut_1^i x\, t_2^i v \in L$ для всіх $i = 0,1,\ldots$. Зрозуміло, що t_1 (так само як і t_2) не може складатися з різних символів (інакше для деякого i ланцюжок $ut_1^i x\, t_2^i v$ не буде належати L). Однак якщо t_1 складається лише з одного символу, то збільшуючи i, можна порушити баланс символів a,b,c. Тому мова L не може бути КВ-мовою.

2.5. Нормальні форми Хомського та Грейбах

Нормальна форма Хомського

Означення 2.36. Назвемо КВ-граматику $G=(N, T, P, S)$ граматикою в *нормальній формі Хомського*, якщо її правила мають вигляд $S\to\varepsilon$, $A\to a$, $A\to BC$ для деяких $A, B, C\in N$, $a\in T$.

Лема 2.7. За кожною КВ-граматикою можна побудувати еквівалентну їй граматику в нормальній формі Хомського.

Доведення. Нехай дана КВ-граматика $G=(N, T, P, S)$. Здійснимо низку перетворень цієї граматики так, що породжувана нею мова залишиться незмінною.

Спочатку побудуємо еквівалентну граматику $G'=(N', T', P', S')$ без ε-правил. Далі замінимо в усіх правилах кожен термінальний символ a на новий нетермінальний символ $N(a)$ і додамо до множини P правила $N(a)\to a$ для всіх $a\in T$.

Видалимо правила вигляду $A\to A_1A_2\ldots A_n$, де $n>2$, замінюючи їх на низку нових правил $A\to A_1N(A_2\ldots A_n)$, $N(A_2\ldots A_n)\to A_2N(A_3\ldots A_n)$, …, $N(A_{n-1}\ldots A_n)\to A_{n-1}A_n$ (при цьому додаємо нові нетермінальні символи $N(A_2\ldots A_n)$, $N(A_3\ldots A_n)$, …, $N(A_{n-1}\ldots A_n)$).

Якщо для деяких $A\in N, B\in N$ та $\alpha\in(N\cup T)^*$ множина P містить правила $A\to B$, $B\to\alpha$, але не містить правила $A\to\alpha$, то додамо це правило в P. Повторюємо процедуру, доки можливо. Після цього видалимо з множини P усі правила вигляду $A\to B$.

Нарешті, змінимо S на S' і додамо правила $S'\to\varepsilon$, $S\to S'$ у тому випадку, коли мова $L(G)$ містить порожній ланцюжок.

Приклад 2.21. Граматика $S\to\varepsilon$, $S\to abSc$ еквівалентна такій граматиці в нормальній формі Хомського: $S\to\varepsilon$, $S\to AB$, $B\to CD$, $D\to SE$, $C\to b$, $A\to a$, $E\to c$.

Завдання 2.7. Звести КВ-граматику до нормальної форми Хомського:

$S \to bS \mid pCD$

$C \to aaD \mid CCD$

$D \to \varepsilon$

Розв'язання.

1) Нетермінал S присутній у правих частинах продукцій, тому додаємо продукцію $S_0 \to S$, де S_0 – нова аксіома.

2) Термінал b замінюємо на B, a замінюємо на A, p – на P, тобто вводимо нові нетермінали й замінюємо входження термінальних у правила на нові, щойно введені, нетермінали:

$S \to BS \mid PCD$

$C \to AAD \mid CCD$

$D \to \varepsilon$

Додаємо продукції $A \to a$, $B \to b$, $P \to p$.

3) Усі продукції з правими частинами, довжина яких (l) більша за 2, розкладаємо на (l–1)-продукції так, щоб у правих частинах було тільки по 2 нетермінальних символи, при цьому вводимо нові допоміжні нетермінали. Отже, $Q \to AA$, $W \to PC$, $E \to CC$ – нові правила. Тепер:

вводимо також правила $S \to WD$, $C \to QD$, $C \to ED$;

виключаємо правила $S \to PCD$, $C \to AAD$, $C \to CCD$;

$\{Q, W, E\} \in N$ – нові нетермінали.

Отримали:

$S \to BS \mid WD$

$C \to QD \mid ED$

$Q \to AA$

$W \to PC$

$E \to CC$

$D \to \varepsilon$

$A \to a$, $B \to b$, $P \to p$

4) Маємо продукцію $D \to \varepsilon$, тому позбуваємось ε-нетермінала стандартним способом (виключення з правих частин), а також:

виключаємо правило $D \to \varepsilon$;

додаємо $S \to W$, $C \to Q$, $C \to E$.

5) Маємо продукції з одним нетерміналом у правій частині: $S \to W$, $C \to Q$, $C \to E$, тому робимо відповідну каскадну підстановку:

$S \to W$, $W \to PC$;

$C \to Q$, $Q \to AA$;

$C \to E$, $E \to CC$,

отже, отримуємо нові правила: $S \to PC$, $C \to AA$, $C \to CC$, а старі $S \to W$, $C \to Q$, $C \to E$ – виключаємо.

6) Оскільки в правилі $S_0 \to S$ права частина теж закоротка, то підставимо для S_0 усі альтернативи для S. У результаті отримаємо граматику в нормальній формі Хомського, еквівалентну початковій:

$S_0 \to BS \mid WD \mid PC$

$S \to BS \mid WD \mid PC$

$C \to QD \mid ED \mid AA \mid CC$

$Q \to AA$

$W \to PC$

$E \to CC$

$A \to a$, $B \to b$, $P \to p$

Завдання 2.8. Звести КВ-граматику до нормальної форми Хомського:

$S \to bC \mid aD$

$C \to bCC \mid aS \mid a$

$D \to aDD \mid bS \mid b$

$P \to aD$

Розв'язання.

1) Відкинемо недосяжні й непродуктивні нетермінали. Ітераційно побудуємо множини досяжних:

$R_0 = \{S\}$

$R_1 = \{C,D,S\}$

$R_2 = \{C,D,S\} = R_1 = R$

і продуктивних нетерміналів:

$P_0 = \{C,D\}$

$P_1 = \{C,D,S,P\}$

$P_2 = \{C,D,S,P\} = P_1 = P$

Тоді недосяжні нетермінали $N \backslash R = \{P\}$, а непродуктивних немає: $N \backslash P = \varnothing$. Тому правило $P \to aD$ слід виключити з граматики й подальшого розгляду.

2) Нетермінал S присутній у правих частинах продукцій, тому додаємо продукцію $S_0 \to S$, де S_0 – нова аксіома граматики.

3) Вводимо нові допоміжні нетермінали A та B, у правилах b замінюємо на B, a – на A. Отримуємо:

$S \to BC \mid AD$

$C \to ACC \mid AS \mid A$

$D \to ADD \mid BS \mid B$

Додаємо також продукції $A \to a$, $B \to b$.

4) Вводимо допоміжні нетермінали для правил з довшими ніж 2 елементи правими частинами, а також відповідні правила для них: $Q \to AC$, $W \to AD$, $C \to QC$, $D \to WD$, виключаючи при цьому $C \to ACC$ та $D \to ADD$ ($\{Q, W\} \in N$).

5) Маємо продукції $C \to A$, $A \to a$, $D \to B$, $B \to b$; вводимо продукції $C \to a$, $D \to b$, натомість виключаємо $C \to A$, $D \to B$.

6) Отримуємо еквівалентну граматику в нормальній формі Хомського, підставивши у правило $S_0 \to S$ усі альтернативи для S:

$S_0 \to BC \mid AD$

$S \to BC \mid AD$

$C \to QC \mid AS \mid a$

$D \to WD \mid BS \mid b$

$Q \to AC$

$W \to AD$

$A \to a, B \to b$

Завдання 2.9. Звести КВ-граматику до нормальної форми Хомського:

$S \to aKd$

$K \to aKd \mid P \mid \varepsilon$

$P \to bPc \mid F \mid \varepsilon$

Розв'язання.

1) Відкинемо недосяжні й непродуктивні нетермінали. F – непродуктивний.

2) Позбудемось ε-правил. Замість $K \to \varepsilon$ додамо $K \to ad$, $S \to ad$; замість $P \to \varepsilon$ додамо $P \to bc$. Отримаємо:

$S \to aKd \mid ad$

$K \to aKd \mid ad \mid P$

$P \to bPc \mid bc$

3) Замість правила $K \to P$ (із закороткою правою частиною), ураховуючи $P \to bPc \mid bc$, додамо нові правила: $K \to bPc \mid bc$.

4) Вводимо допоміжні нетермінали та правила: $A \to a$, $B \to b$, $D \to d$, $C \to$ c, отримуємо, відповідно:

$A \to a, B \to b, D \to d, C \to c$

$S \to AKD \mid AD$

$K \to AKD \mid AD \mid BC \mid BPC$

$P \to BPC \mid BC$

5) Нарешті, розбивши довгі праві частини правил шляхом введення допоміжних нетерміналів, отримаємо результуючу граматику в нормальній формі Хомського:

$S \to XD \mid AD$

$K \to XD \mid AD \mid BC \mid YC$

$P \to YC \mid BC$

$A \to a, B \to b, D \to d, C \to c$

$X \to AK, Y \to BP$

Завдання 2.10. Звести КВ-граматику до нормальної форми Хомського:

$S \to aUbU \mid CCb \mid \varepsilon$

$U \to S \mid C \mid ba$

Розв'язання.

1) Відкинемо недосяжні й непродуктивні нетермінали: C – непродуктивний.

2) Введемо нову аксіому S_0, оскільки символ S присутній у правих частинах правил, при цьому позбудемось ε-правила $S \to \varepsilon$ (S та U – ε-нетермінали). Отримаємо:

$S_0 \to \varepsilon \mid S$

$S \to aUbU \mid ab \mid abU \mid aUb$

$U \to S \mid ba$

3) Додамо допоміжні нетермінали й правила: $T_a \to a$, $T_b \to b$, отримаємо:

$S_0 \to \varepsilon \mid S$

$S \to T_aUT_bU \mid T_aT_b \mid T_aUT_b \mid T_aT_bU$

$U \to S \mid T_bT_a$

$T_a \to a$, $T_b \to b$

4) Для довгих правих частин введемо допоміжні нетермінали, щоб розбити їх на частини:

$S_0 \to \varepsilon \mid S$

$S \to AB \mid T_aT_b \mid AT_b \mid T_aB$

$A \to T_aU$

$B \to T_bU$

$U \to S \mid T_bT_a$

$T_a \to a$, $T_b \to b$

5) Закороткі правила $U \to S$ та $S_0 \to S$ замінимо всіма можливими замінами S:

$S_0 \to \varepsilon \mid AB \mid T_aT_b \mid AT_b \mid T_aB$

$S \to AB \mid T_aT_b \mid AT_b \mid T_aB$

$A \to T_aU$

$B \to T_bU$

$U \to AB \mid T_aT_b \mid AT_b \mid T_aB \mid T_bT_a$

$T_a \to a$, $T_b \to b$

Отримали результуючу еквівалентну граматику в нормальній формі Хомського.

Завдання 2.11. Звести КВ-граматику до нормальної форми Хомського:

$S \to aD \mid D$

$U \to bb \mid Cb$

$D \to \varepsilon \mid Db \mid Ua$

$L \to UD$

Розв'язання.

1) Оскільки С – непродуктивний, а L – недосяжний нетермінали, то вхідна граматика еквівалентна такій:

$S \to aD \mid D$

$U \to bb$

$D \to \varepsilon \mid Db \mid Ua$

2) Оскільки граматика породжує, зокрема, порожнє слово, то необхідно ввести нову аксіому S_0 і правило $S_0 \to S \mid \varepsilon$.

3) Введемо допоміжні нетермінали й правила: $A \to a$, $B \to b$, отримаємо:

$S_0 \to S \mid \varepsilon$

$S \to AD \mid D$

$U \to BB$

$D \to \varepsilon \mid DB \mid UA$

$A \to a, B \to b$

4) Позбавляємось ε-правила $D \to \varepsilon$:

$S_0 \to S \mid \varepsilon$

$S \to AD \mid D \mid A$

$U \to BB$

$D \to DB \mid UA \mid B$

$A \to a, B \to b$

5) Для односимвольних правих частин робимо заміни: для $S \to A$ та $D \to B$ виконуємо заміни на відповідно термінали, а для $S \to D$ підставляємо всі альтернативи для нетермінала D:

$S_0 \to \varepsilon \mid AD \mid DB \mid UA \mid b \mid a$

$S \to AD \mid DB \mid UA \mid b \mid a$

$U \to BB$

$D \to DB \mid UA \mid b$

$A \to a, B \to b$

Отримали результуючу еквівалентну граматику в нормальній формі Хомського.

Нормальна форма Грейбах

Означення 2.37. КВ-граматику $G=(N, T, P, S)$ будемо називати граматикою в *нормальній формі Грейбах*, якщо її правила мають вигляд $A\to a\alpha$ ($a\in T$, $\alpha\in(N\cup T)^*$), тобто кожне правило починається з термінального символу.

Сформулюємо без доведення таке твердження.

Лема 2.8. За кожною КВ-граматикою можна побудувати ε-еквівалентну їй (з точністю до порожнього ланцюжка) граматику в нормальній формі Грейбах.

Зауважимо, що наявність нормальної форми Грейбах дозволяє досить легко за кожною граматикою побудувати еквівалентний їй недетермінований магазинний автомат. Також у таких граматиках довжина виведення термінального ланцюжка не перевищує його довжину.

Приклад 2.22. Граматика $S\to\varepsilon$, $S\to abSc$ еквівалентна такій граматиці в нормальній формі Грейбах: $S\to abc$, $S\to abSc$.

Завдання 2.12. Звести КВ-граматику до нормальної форми Грейбах:

$S \to AB$

$A \to \varepsilon \mid aC$

$B \to Bb \mid \varepsilon$

$C \to Cc \mid \varepsilon$

Розв'язання.

1) Легко перевірити, що всі нетермінали суттєві, тобто досяжні й продуктивні: $R_0=\{S\}$, $R_1=\{S, A, B\}$, $R_2=\{S, A, B, C\}=N$; $P_0=\{A, B, C\}$ $P_1=\{S, A, B, C\}=N$.

2) Позбудемось ε-переходів (ε-правил), для чого додамо правила, отримані шляхом підстановок ε-нетерміналів:

$S \to B$, $S \to A$, $S \to \varepsilon$, $B \to b$, $C \to c$, $A \to a$

і вилучимо ε-правила:

$A \to \varepsilon$, $B \to \varepsilon$, $C \to \varepsilon$.

Отримаємо еквівалентну граматику:

$S \to AB \mid A \mid B \mid \varepsilon$

$A \to aC \mid a$

$B \to Bb \mid b$

$C \to Cc \mid c$

3) Позбудемось лівої рекурсії. Для цього правило

$C \to Cc \mid c$

замінимо сукупністю еквівалентних правил (ввівши новий нетермінал D):

$C \to cD, D \to cD, D \to \varepsilon.$

Позбудемось отриманого ε-правила $D \to \varepsilon$: додамо правила $C \to c$, $D \to c$ і одержимо

$C \to cD \mid c, D \to cD \mid c.$

Аналогічно для $B \to Bb \mid b$:

$B \to bF \mid b, F \to bF \mid b.$

Нарешті отримаємо:

$S \to AB \mid A \mid B \mid \varepsilon$

$A \to aC \mid a$

$B \to bF \mid b, F \to bF \mid b$

$C \to cD \mid c, D \to cD \mid c$

4) Для всіх правил, де в правій частині перший символ – нетермінал, робимо всі можливі для нього підстановки, а саме:

$S \to a, S \to aC$ для $S \to A$;

$S \to b, S \to bF$ для $S \to B$;

$S \to aB, S \to aCB$ для $S \to AB$,

вилучаючи самі ці правила, тобто:

$S \to A, S \to B, S \to AB.$

Отримали еквівалентну даній граматику в нормальній формі Грейбах:

$S \to aB \mid aCB \mid a \mid aC \mid b \mid bF \mid \varepsilon$

$A \to aC \mid a$

$B \to bF \mid b, F \to bF \mid b$

$C \to cD \mid c, D \to cD \mid c$

Завдання 2.13. Звести КВ-граматику до нормальної форми Грейбах:

$S \to aAC \mid BA$

$B \to Pk \mid \varepsilon$

$A \to a \mid aA$

$C \to c \mid cC$

Розв'язання.

1) Знайдемо продуктивні нетермінали:

$P_0 = \{A, B, C\}$

$P_1 = \{S, A, B, C\}$

$P_2 = \{S, A, B, C\} = P_1 = P$

Отже, $CP = \{P\}$ – непродуктивний нетермінал, тому виключаємо з граматики правило $B \to Pk$.

2) Позбудемось ε-правила $B \to \varepsilon$, для чого додамо правило $S \to A$. При цьому виключимо продукції $B \to \varepsilon$ та $S \to BA$, оскільки більше не залишається продукцій з нетерміналом B у лівій частині. Отримаємо:

$S \to aAC \mid A$

$A \to a \mid aA$

$C \to c \mid cC$

3) У правилі $S \to A$ заміняємо перший у правій частині нетермінал на всі можливі для нього альтернативи, тобто додаємо правила $S \to a$, $S \to aA$.

Отримали еквівалентну граматику в нормальній формі Грейбах:

$S \to aAC \mid a \mid aA$

$A \to a \mid aA$

$C \to c \mid cC$

Завдання 2.14. Звести КВ-граматику до нормальної форми Грейбах:

$S \to AS \mid \varepsilon$

$A \to aS \mid P \mid b$

$P \to Pc \mid b$

Розв'язання.

1) Усі нетермінали граматики суттєві, тобто досяжні та продуктивні.

2) Оскільки слово ε виводиться в граматиці, то вводимо окрему нову аксіому, для якої є лише два правила, щоб відділити виведення слова ε від усіх інших можливих виведень у граматиці: $S_0 \to \varepsilon$ та $S_0 \to S$. Також позбудемось ε-переходу $S \to \varepsilon$ шляхом підстановки в інші правила, де S є в правих частинах. Отримаємо:

$S_0 \to \varepsilon \mid S$

$S \to AS \mid A$

$A \to aS \mid P \mid b \mid a$

$P \to Pc \mid b$

3) Позбудемось лівої рекурсії для нетермінала P, замінивши правила $P \to Pc \mid b$ сукупністю правил $P \to bP^1$, $P^1 \to cP^1$, $P^1 \to \varepsilon$. Далі позбудемось отриманого ε-правила $P^1 \to \varepsilon$, для чого виконаємо можливі підстановки P^1 і одержимо нові правила: $P \to b$ та $P^1 \to c$ замість $P^1 \to \varepsilon$, тобто врешті-решт одержимо такі нові правила:

$P \to bP^1 \mid b$

$P^1 \to cP^1 \mid c$

Отримаємо:

$S_0 \to \varepsilon \mid S$

$S \to AS \mid A$

$A \to aS \mid P \mid b \mid a$

$P \to bP^1 \mid b$

$P^1 \to cP^1 \mid c$

4) Підставимо замість нетермінала P на першому місці правих частин правил слова, що безпосередньо виводяться з P:

$S_0 \to \varepsilon \mid S$

$S \to AS \mid A$

$A \to aS \mid bP^1 \mid b \mid a$

$P \to bP^1 \mid b$

$P^1 \to cP^1 \mid c$

Аналогічно для нетермінала A:

$S_0 \to \varepsilon \mid S$

$S \to aSS \mid bP^1S \mid bS \mid aS \mid bP^1 \mid b \mid a$

$A \to aS \mid bP^1 \mid b \mid a$

$P \to bP^1 \mid b$

$P^1 \to cP^1 \mid c$

Оскільки нетермінали A та P стали в результаті перетворень недосяжними, то виключимо їх із правил граматики – отримаємо граматику, еквівалентну початковій, у нормальній формі Грейбах:

$S_0 \to \varepsilon \mid aSS \mid bP^1S \mid bS \mid aS \mid bP^1 \mid b \mid a$

$S \to aSS \mid bP^1S \mid bS \mid aS \mid bP^1 \mid b \mid a$

$P^1 \to cP^1 \mid c$

Завдання 2.15. Звести КВ-граматику до нормальної форми Грейбах:

$S \to A \mid bD$

$D \to \varepsilon \mid aa \mid CD$

$A \to u \mid \varepsilon \mid C$

$K \to Bf$

Розв'язання.

1) Знайдемо недосяжні й непродуктивні нетермінали:

$R_0 = \{S\}$

$R_1 = \{S, A, D\}$

$R_2 = \{S, A, D, C\}$

$R_3 = \{S, A, D, C\} = R_2 = R$

Недосяжні нетермінали $N \backslash R = \{K\}$, тобто нетермінал K, виключаємо з правил граматики. Продуктивні нетермінали:

$P_0 = \{D, A\}$

$P_1 = \{S, D, A\}$

$P_2 = \{S, D, A\} = P_1 = P$

Отже, непродуктивні нетермінали $N \backslash P = \{C\}$, тому виключаємо з граматики також нетермінал C. Залишається

$S \to A \mid bD$

$D \to \varepsilon \mid aa$

$A \to u \mid \varepsilon$

2) Оскільки нетермінал A займає першу позицію в правій частині правила $S \to A$, то замінимо його можливими підстановками: $A \to u \mid \varepsilon$. Отримаємо

$S \to u \mid \varepsilon \mid bD$

$D \to \varepsilon \mid aa$

$A \to u \mid \varepsilon$

Тепер нетермінал A став недосяжним, отже, виключаємо його з граматики й отримуємо

$S \to u \mid \varepsilon \mid bD$

$D \to \varepsilon \mid aa$

3) Позбудемось ε-продукцій, замінивши входження ε-нетерміналів (S та D) у всіх правилах, а також увівши нову аксіому S_0 (для збереження можливості породження слова ε):

$S_0 \to S \mid \varepsilon$

$S \to u \mid bD \mid b$

$D \to aa$

Отримуємо еквівалентну початковій граматику в нормальній формі Грейбах:

$S_0 \to \varepsilon \mid u \mid bD \mid b$

$S \to u \mid bD \mid b$

$D \to aa$

Завдання 2.16. Звести КВ-граматику до нормальної форми Грейбах:

$S \to c \mid aABU \mid cC$

$A \to Aa \mid bU \mid \varepsilon$

$B \to aa \mid Cc$

$U \to bA \mid BB \mid \varepsilon$

$M \to AB$

Розв'язання.

1) Легко визначити, що C – непродуктивний, M – недосяжний нетермінали, тому виключимо їх із правил граматики:

$S \to c \mid aABU$

$A \to Aa \mid bU \mid \varepsilon$

$B \to aa$

$U \to bA \mid BB \mid \varepsilon$

2) Виключимо ε-переходи ($A \to \varepsilon$ та $U \to \varepsilon$) з граматики, отримаємо:

$S \to c \mid aABU \mid aBU \mid aAB \mid aB$

$A \to Aa \mid bU \mid b \mid a$

$B \to aa$

$U \to bA \mid BB \mid b$

3) Позбавимось лівої рекурсії для нетермінала A:

$S \to c \mid aABU \mid aBU \mid aAB \mid aB$

$A \to bUW \mid bW \mid aW$

$W \to aW \mid \varepsilon$

$B \to aa$

$U \to bA \mid BB \mid b$

4) Залишилось позбутись ε-переходу $W \to \varepsilon$ і першого входження нетермінала B у правилі $U \to BB$ (шляхом підстановки можливих виведень згідно з продукцією граматики):

$S \to c \mid aABU \mid aBU \mid aAB \mid aB$

$A \to bUW \mid bW \mid aW \mid bU \mid b \mid a$

$W \to aW \mid a$

$B \to aa$

$U \to bA \mid aaB \mid b$

Отримали граматику в нормальній формі Грейбах, яка еквівалентна початковій.

Завдання для самоконтролю

1. Класифікація граматик Хомського.
2. Співвідношення класів породжувальних граматик і класів автоматів (машин).
3. Операції над контекстно-вільними мовами.
4. Розв'язність проблеми еквівалентності для класів граматик.
5. Замкненість операцій над КВ-мовами.
6. Розв'язність проблеми належності для класів граматик.
7. Еквівалентні перетворення контекстно-вільних граматик.
8. Включення класів граматик і мов до класифікації Хомського. Обґрунтування.

9. Нормальні форми контекстно-вільних граматик.
10. Доведення теореми Кнастера – Тарського – Кліні.
11. Дерева виведення. Однозначні та неоднозначні КВ-граматики.
12. Розв'язні й нерозв'язні проблеми для контекстно-вільних граматик.

Задачі

1. Побудувати породжувальну граматику G для мови L. Довести, що $L(G) = L$. Варіанти мови L:

a) $a^*|b^*|c^*$;
b) $(a|b|c)^*$;
c) $(a|b)^*c$;
d) a^*ba^*b;
e) $a^*b|b^*a$;
f) $a^*|ba^*b$;
g) $a^*b^*c^*$;
h) $a|b^*c|cb^*$;
i) $a^*b|a|b$;
j) $(a|b)^*a^*$;
k) $(abc)^*$;
l) $(ab)^*|ab$;
m) $a^*(a|b)^*b^*$;
n) $(ab)^*|(a|b)^*$.

2. Побудувати породжувальну граматику G для мови L. Довести, що $L(G) = L$ та мова L не є КВ-мовою. Варіанти мови L:

a) $\{e^n\, d\, b^{2n} c^{3n} \mid n \geq 0\}$;
b) $\{a^{3n} b^n\, d\, c^{2n} \mid n \geq 0\}$;
c) $\{e^{2n}\, d\, b^{3n} c^n \mid n \geq 0\}$;
d) $\{a^n b^{3n}\, d\, c^{2n} \mid n \geq 0\}$;
e) $\{e^{3n}\, d\, b^n c^{2n} \mid n \geq 0\}$;
f) $\{a^{2n} b^n\, d\, c^{3n} \mid n \geq 0\}$;
g) $\{a\, e^{3n}\, d\, b^{2n} c^n \mid n \geq 0\}$;
h) $\{a^n b^{2n}\, d\, c^{2n} \mid n \geq 0\}$;

i) $\{a\, e^{2n}\, b\, d^{n}\, c^{2n} \mid n \geq 0\}$;
j) $\{d^{3n}\, b^{2n}\, e\, c^{n} \mid n \geq 0\}$;
k) $\{e^{n}\, b\, d^{n}\, c^{2n} \mid n \geq 0\}$;
l) $\{a^{2n}\, b^{3n}\, c\, d^{n} \mid n \geq 0\}$;
m) $\{a\, d^{3n}\, b^{2n}\, e\, c^{n} \mid n \geq 0\}$;
n) $\{a^{n}\, cb^{2n}c^{3n} \mid n \geq 0\}$;
o) $\{a^{3n}\, b\, d^{2n}c^{n} \mid n \geq 0\}$;
p) $\{a\, b^{3n}\, d\, c^{2n}\, e\, a^{n} \mid n \geq 0\}$.

3. Побудувати нормальну форму Хомського та нормальну форму без ε-правил для КВ-граматики. Варіанти КВ-граматик:

a) $S \to AbBE$, $S \to ADcBE$, $S \to ADeE$, $S \to AEcC$,
$E \to \varepsilon \mid bEc$, $A \to \varepsilon \mid aA$, $B \to \varepsilon \mid bB$, $C \to \varepsilon \mid cC$, $D \to dD$;

b) $S \to bBA$, $S \to DcBA$, $S \to DAA$, $S \to AcC$,
$A \to \varepsilon \mid bAc$, $B \to \varepsilon \mid bB$, $C \to \varepsilon \mid cC$, $D \to dD$;

c) $S \to AbB$, $S \to ADcB$, $S \to ADBB$, $S \to ABcC$,
$B \to \varepsilon \mid bBc$, $A \to \varepsilon \mid aA$, $C \to \varepsilon \mid cC$, $D \to dD$;

d) $S \to AbBC$, $S \to ADcBC$, $S \to ADCC$, $S \to ACc$,
$C \to \varepsilon \mid bCc$, $A \to \varepsilon \mid aA$, $B \to \varepsilon \mid bB$, $D \to dD$;

e) $S \to AbBD$, $S \to AcBD$, $S \to ADD$, $S \to ADcC$,
$D \to \varepsilon \mid bDc$, $A \to \varepsilon \mid aA$, $B \to \varepsilon \mid bB$, $C \to \varepsilon \mid cC$;

f) $S \to EbBA$, $S \to EDcBA$, $S \to EDAA$, $S \to EAcC$,
$A \to \varepsilon \mid bAc$, $E \to \varepsilon \mid EE$, $B \to \varepsilon \mid bB$, $C \to \varepsilon \mid cC$, $D \to dD$;

g) $S \to aSb$, $S \to ABCD$, $S \to DCBA$, $S \to BBCC$,
$A \to \varepsilon \mid aA$, $B \to \varepsilon \mid bB$, $C \to \varepsilon \mid cC$, $D \to dD$;

h) $S \to BbB$, $S \to cCc$, $S \to AaA$, $S \to dDd$,
$A \to \varepsilon \mid aA$, $B \to \varepsilon \mid bB$, $C \to \varepsilon \mid cC$, $D \to dD$.

4. Побудувати КВ-граматику G та систему рівнянь E для мови L. Довести незалежно, що $L(G) = L$ та $R(E) = L$ ($R(E)$ – розв'язок системи рівнянь E). Варіанти мови L:

a) $\{a^{2n}\, b^{n}\, c \mid n \geq 0\}$;
b) $\{a^{n}\, c\, b^{3n} \mid n \geq 0\}$;
c) $\{a^{2n}\, c\, b^{2n} \mid n \geq 0\}$;

d) $\{d^{2n}\, c\, b^{2n} \mid n \geq 0\}$;
e) $\{c\, a^{n}\, b^{3n} \mid n \geq 0\}$;
f) $\{d^{2n}\, b^{2n}\, c \mid n \geq 0\}$;
g) $\{a^{3n}\, c\, b^{n} \mid n \geq 0\}$;
h) $\{c\, d^{2n}\, b^{2n} \mid n \geq 0\}$;
i) $\{a^{n}\, b^{3n}\, c \mid n \geq 0\}$;
j) $\{d^{2n}\, c^{2}\, b^{2n} \mid n \geq 0\}$;
k) $\{c\, a^{3n}\, b^{n} \mid n \geq 0\}$;
l) $\{d^{2n}\, b^{n}\, c \mid n \geq 0\}$;
m) $\{a^{n}\, d\, b^{3n} \mid n \geq 0\}$;
n) $\{c^{n}\, d^{2}\, b^{n} \mid n \geq 0\}$;
o) $\{a^{3n}\, b^{n}\, c \mid n \geq 0\}$;
p) $\{c^{n}\, a^{2}\, b^{n} \mid n \geq 0\}$.

Приклади завдань для контрольної роботи

Варіант 1

1. Класифікація граматик Хомського.

2. Співвідношення класів породжувальних граматик і класів автоматів (машин).

3. Побудувати породжувальну граматику G для мови $L = (a|b|c)^{*}$. Довести, що $L(G) = L$.

4. Побудувати породжувальну граматику G для мови $L = \{e^{n}\, d\, b^{2n} c^{3n} \mid n \geq 0\}$. Довести, що $L(G) = L$ та мова L не є КВ-мовою.

5. Побудувати нормальну форму Хомського для КВ-граматики з множиною правил:

$S \to AbBE$, $S \to ADcBE$, $S \to ADeE$, $S \to AEcC$,
$E \to \varepsilon \mid bEc$, $A \to \varepsilon \mid aA$, $B \to \varepsilon \mid bB$, $C \to \varepsilon \mid cC$, $D \to dD$.

6. Побудувати КВ-граматику G та систему рівнянь E для мови $L = \{a^{2n}\, b^{n}\, c \mid n \geq 0\}$. Довести незалежно, що $L(G) = L$ та $R(E) = L$ ($R(E)$ – розв'язок системи рівнянь E).

Варіант 2

1. Операції над контекстно-вільними мовами.

2. Розв'язність проблеми еквівалентності для класів граматик.

3. Побудувати породжувальну граматику G для мови $L=(a|b)^*c$. Довести, що $L(G)=L$.

4. Побудувати породжувальну граматику G для мови $L=\{a^{3n} b^n d\, c^{2n} \mid n\geq 0\}$. Довести, що $L(G)=L$ та мова L не є КВ-мовою.

5. Побудувати нормальну форму без ε-правил для КВ-граматики з множиною правил:

$S\to AbBE$, $S\to ADcBE$, $S\to ADeE$, $S\to AEcC$,

$E\to\varepsilon \mid bEc$, $A\to\varepsilon \mid aA$, $B\to\varepsilon \mid bB$, $C\to\varepsilon \mid cC$, $D\to dD$.

6. Побудувати КВ-граматику G та систему рівнянь E для мови $L=\{a^n\ c\ b^{3n} \mid n\geq 0\}$. Довести незалежно, що $L(G)=L$ та $R(E)=L$ ($R(E)$ – розв'язок системи рівнянь E).

Варіант 3

1. Замкненість операцій над КВ-мовами.

2. Розв'язність проблеми належності для класів граматик.

3. Побудувати породжувальну граматику G для мови $L=a^*ba^*b$. Довести, що $L(G)=L$.

4. Побудувати породжувальну граматику G для мови $L=\{e^{2n} d\, b^{3n} c^n \mid n\geq 0\}$. Довести, що $L(G)=L$ та мова L не є КВ-мовою.

5. Побудувати нормальну форму Хомського для КВ-граматики з множиною правил:

$S\to bBA$, $S\to DcBA$, $S\to DAA$, $S\to AcC$,

$A\to\varepsilon \mid bAc$, $B\to\varepsilon \mid bB$, $C\to\varepsilon \mid cC$, $D\to dD$.

6. Побудувати КВ-граматику G та систему рівнянь E для мови $L=\{a^{2n}\ c\ b^{2n} \mid n\geq 0\}$. Довести незалежно, що $L(G)=L$ та $R(E)=L$ ($R(E)$ – розв'язок системи рівнянь E).

Варіант 4

1. Еквівалентні перетворення контекстно-вільних граматик.

2. Включення класів граматик і мов до класифікації Хомського. Обґрунтування.

3. Побудувати породжувальну граматику G для мови $L=(a|b)^*a^*$. Довести, що $L(G)=L$.

4. Побудувати породжувальну граматику G для мови $L = \{a^n b^{3n} d\, c^{2n} \mid n \geq 0\}$. Довести, що $L(G) = L$ та мова L не є КВ-мовою.

5. Побудувати нормальну форму без ε-правил для КВ-граматики з множиною правил:

$S \to bBA$, $S \to DcBA$, $S \to DAA$, $S \to AcC$,
$A \to \varepsilon \mid bAc$, $B \to \varepsilon \mid bB$, $C \to \varepsilon \mid cC$, $D \to dD$.

6. Побудувати КВ-граматику G та систему рівнянь E для мови $L = \{d^{2n}\, c\, b^{2n} \mid n \geq 0\}$. Довести незалежно, що $L(G) = L$ та $R(E) = L$ ($R(E)$ – розв'язок системи рівнянь E).

РОЗДІЛ 3
РЕКУРСІЯ ТА НАЙМЕНША НЕРУХОМА ТОЧКА

Означення. ω-областю називаємо повну, частково впорядковану множину з найменшим елементом.

Означення. Ланцюгом будемо називати довільну послідовність елементів $\{d_0, d_1, \ldots\}$ ω-області, що задовольняють умову $d_0 \le d_1 \le \ldots$, де $\le$ – відношення часткового порядку на ω-області.

Означення. Відображення $\varphi : D \to D$ неперервне, якщо для $\forall$ -ланцюга $d_0 \le d_1 \le \ldots$ з ω-області D виконується

$$\phi(\coprod_{i \in w} d_i) = \ = \coprod_{i \in w} \phi(d_i) \ .$$

Лема. Нехай φ – n-вимірне відображення : φ : $D \to D$, де D – ω-область, φ – неперервне відображення, тоді й тільки тоді, коли φ неперервне щодо кожного зі своїх аргументів.

Завдання 3.1. Довести неперервність оператора $IF(p, f, g)$, де f та g неперервні.

Розв'язання.

Доведемо неперервність $IF(p, f, g)$ за аргументом f. Візьмемо довільний ланцюг $f_0 \le f_1 \le \ldots$. Доведемо рівність $IF(p, \coprod_{i \in \omega} f_i, g)(st) = \coprod_{i \in \omega} IF(p, f_i, g)(st)$. Маємо:

$$IF(p, \coprod_{i \in \omega} f_i, g)(st) = \begin{cases} \coprod_{i \in \omega} f_i(st), & \text{якщо} p(st) = \text{true} \\ g(st), & \text{якщо} p(st) = \text{false} \\ \bot, & \text{else} \end{cases} =$$

$$= \begin{cases} \exists k (f_k(st) \downarrow = r), & \text{якщо } p(st) = \text{true} \\ g(d), & \text{якщо } p(st) = \text{false} \\ \bot, & \text{else.} \end{cases}$$

Ураховуючи неперервність і те, що f_i утворює ланцюг, маємо:

$$\begin{cases} \exists k(f_k(st)\downarrow = r), \text{ якщо } p(st) = true \\ g(d), \text{ якщо } p(st) = false \\ \bot, \text{ в інших випадаках} \end{cases} = \exists k \begin{cases} f_k(st)\downarrow = r, \text{ якщо } p(st) = true \\ g(d), \text{ якщо } p(st) = false \\ \bot, \text{ в інших випадаках} \end{cases} =$$

$$= \coprod_{k\in\omega} \begin{cases} f_k(st)\downarrow = r, \text{ якщо } p(st) = true \\ g(d), \text{ якщо } p(st) = false \\ \bot, \text{ в інших випадаках} \end{cases} = \coprod_{k\in\omega} IF(p, f_k, g).$$

Аналогічно доводимо неперервність $IF(p, f, g)$ відносно аргументу g.

Доведемо тепер неперервність $IF(p, f, g)$ відносно аргументу p. Візьмемо довільний ланцюг $p_0 \le p_1 \le \ldots$. Доведемо рівність $IF(\coprod_{i\in\omega} p_i, f, g)(st) = \coprod_{i\in\omega} IF(p_i, f, g)(st)$.

$$IF(\coprod_{i\in\omega} p_i, f, g)(st) = \begin{cases} f(st), \text{ якщо } \coprod_{i\in\omega} p(st) = true \\ g(st), \text{ якщо } \coprod_{i\in\omega} p(st) = false \\ \bot, \text{ в інших випадаках} \end{cases} = \begin{cases} f(st), \text{ якщо } \exists k_1(p_{k_1}(st) = true) \\ g(st), \text{ якщо } \exists k_2(p_{k_2}(st) = false) \\ \bot, \text{ в інших випадаках.} \end{cases}$$

Використаємо неперервність p_i і те, що p_i утворюють ланцюг, покладемо $k = \max(k_1, k_2)$. Маємо:

$$\begin{cases} \mathrm{f(st)}, \text{ якщо } \exists \mathrm{k}_1(\mathrm{p}_{\mathrm{k}_1}(\mathrm{st}) = \mathrm{true}) \\ \mathrm{g(st)}, \text{ якщо } \exists \mathrm{k}_2(\mathrm{p}_{\mathrm{k}_2}(\mathrm{st}) = \mathrm{false}) \\ \bot, \text{ в інших випадаках} \end{cases} = \begin{cases} \mathrm{f(st)}, \text{ якщо } \exists \mathrm{k}(\mathrm{p}_{\mathrm{k}}(\mathrm{st}) = \mathrm{true}) \\ \mathrm{g(st)}, \text{ якщо } \exists \mathrm{k}(\mathrm{p}_{\mathrm{k}}(\mathrm{st}) = \mathrm{false}) \\ \bot, \text{ в інших випадаках} \end{cases} = \coprod_{\mathrm{k}\in\omega} \mathrm{IF}(\mathrm{p}_{\mathrm{k}}, \mathrm{f}, \mathrm{g})$$

Неперервність щодо аргументів доведено, отже, $IF(p, f, g)$ – неперервна функція своїх аргументів.

Завдання 3.2. Довести неперервність оператора $F(f, g) = IF(p, f \circ g, g \circ f)$, де f та g неперервні.

Розв'язання.

Для того, щоб показати неперервність оператора F, потрібно довести його неперервність за f та g:

1) $IF(p,\left(\coprod_{i\in\omega} f_i\right)\circ g, g\circ\left(\coprod_{i\in\omega} f_i\right)) = \coprod_{i\in\omega} IF(p, f_i\circ g, g\circ f_i).$

2) $IF(p, f\circ\left(\coprod_{i\in\omega} g_i\right),\left(\coprod_{i\in\omega} g_i\right)\circ f) = \coprod_{i\in\omega} IF(p, f\circ g_i, g_i\circ f).$

Доведемо 1) – неперервність за f:

1) $IF(p,\left(\coprod_{i\in\omega} f_i\right)\circ g, g\circ\left(\coprod_{i\in\omega} f_i\right)) = \coprod_{i\in\omega} IF(p, f_i\circ g, g\circ f_i)$:

а) $p(st) = true$:

$$IF\left(p,\left(\coprod_{i\in\omega} f_i\right)\circ g, g\circ\left(\coprod_{i\in\omega} f_i\right)\right)(st)\downarrow \;= \left(\left(\coprod_{i\in\omega} f_i\right)\circ g\right)(st)\downarrow = \tau \Leftrightarrow$$

$$\Leftrightarrow \exists b\left(\left(\coprod_{i\in\omega} f_i\right)\right)(st)\downarrow = b \;\&\; g(b)\downarrow = \tau \Leftrightarrow$$

$$\Leftrightarrow \exists b \exists k\; f_k(st)\downarrow = b \;\&\; g(b)\downarrow = \tau \Leftrightarrow$$

$$\Leftrightarrow \exists k\left(f_k\circ g\right)(st)\downarrow = \tau \Leftrightarrow \exists k\; IF\left(p, f_k\circ g, g\circ f_k\right)(st)\downarrow = \tau \Leftrightarrow$$

$$\Leftrightarrow \coprod_{i\in\omega} IF(p, f_i\circ g, g\circ f_i)(st)\downarrow = \tau;$$

б) $p(st) = false$:

$$IF\left(p,\left(\coprod_{i\in\omega} f_i\right)\circ g, g\circ\left(\coprod_{i\in\omega} f_i\right)\right)(st)\downarrow \;= \left(g\circ\left(\coprod_{i\in\omega} f_i\right)\right)(st)\downarrow = \tau \Leftrightarrow$$

$$\Leftrightarrow \exists b\; g(st)\downarrow \;= b \;\&\left(\coprod_{i\in\omega} f_i\right)(b)\downarrow \;= \tau \Leftrightarrow$$

$$\Leftrightarrow \exists b g(st)\downarrow \;= b \;\&\; \exists k\, f_k\left(b\right)\downarrow \;= \tau \Leftrightarrow$$

$$\Leftrightarrow \exists k\left(g\circ f_k\right)(st)\downarrow \;= \tau \Leftrightarrow \exists k\; IF\left(p, f_k\circ g, g\circ f_k\right)(st)\downarrow \;= \tau \Leftrightarrow$$

$$\Leftrightarrow \coprod_{i\in\omega} IF(p, f_i\circ g, g\circ f_i)(st)\downarrow \;= \tau.$$

Завдання для самоконтролю

Для вказаної задачі зробити таке:

1) написати рекурсивну SIPL-програму (SIPL-функцію) для обраної задачі;

2) побудувати семантичний терм програми;

3) побудувати три апроксимації семантичної функції;

4) протестувати побудовані апроксимації на наведених вхідних даних;

5) довести правильність рекурсивної програми.

Варіанти задачі

a) Обчислити F_n – n-й елемент ряду Фібоначчі (n>0). Вхідні дані: $n = 3$.

b) Обчислити C_x^y (x, y>0). Вхідні дані: $x = 4$, $y = 3$.

c) Обчислити $x - y$, використовуючи функцію –1 (x, y>0). Вхідні дані: $x = 4$, $y = 1$.

d) Перевірити парність числа n (n>0). Вхідні дані: $n = 3$.

e) Обчислити суму арифметичної прогресії 1, 2, 4,…, 2n, використовуючи функції +, – (n>0). Вхідні дані: $n = 1$.

f) Обчислити x^y, використовуючи функції ∗,+, – (x,y>0). Вхідні дані: $x = 5$, $y = 2$.

g) Обчислити $[\log_x y]$, використовуючи функції *div*, *mod*, +, – (x>1, y>0). Вхідні дані: $x = 3$, $y = 7$.

h) Перевірити, чи ділиться x на y, використовуючи функції +, – (x, y>0). Вхідні дані: $x = 9$, $y = 5$.

i) Обчислити 3^x, використовуючи функції ∗,+, – (x>0). Вхідні дані: $x = 2$.

j) Обчислити $[\log_2 n]$, використовуючи функції *div*, *mod*, +, – (n>0). Вхідні дані: $n = 3$.

Приклади завдань для контрольної роботи

Варіант 1

I. Визначення та властивості ω-областей.

II. Для функції обчислення F_n (n-го елемента ряду Фібоначчі, $n>0$) зробити таке:

1) написати рекурсивну SIPL-функцію;

2) побудувати семантичний терм;

3) побудувати три апроксимації;

4) протестувати побудовані апроксимації на вхідних даних $n=3$;

5) довести правильність рекурсивної програми.

Варіант 2

I. Визначення та властивості неперервних відображень.

II. Для функції обчислення C_x^y ($x, y>0$) зробити таке:

1) написати рекурсивну SIPL-функцію;

2) побудувати семантичний терм;

3) побудувати три апроксимації;

4) протестувати побудовані апроксимації на вхідних даних $x=4, y=3$;

5) довести правильність рекурсивної програми.

Варіант 3

I. Доведення теореми Кнастера – Тарського – Кліні.

II. Для функції обчислення 3^x ($x>0$) зробити таке:

1) написати рекурсивну SIPL-функцію, використовуючи функції *,+, –;

2) побудувати семантичний терм;

3) побудувати три апроксимації;

4) протестувати побудовані апроксимації на вхідних даних $x=2$;

5) довести правильність рекурсивної програми.

Варіант 4

I. Визначення та властивості ω-областей.

II. Для функції $[\log_2 n]$ ($n>0$) зробити таке:

1) написати рекурсивну SIPL-функцію, використовуючи функції *div*, *mod*, +, –;

2) побудувати семантичний терм;

3) побудувати три апроксимації;

4) провести тестування побудованих апроксимацій на вхідних даних $n = 3$;

5) довести правильність рекурсивної програми.

РОЗДІЛ 4
ОПЕРАЦІЙНА (НАТУРАЛЬНА) СЕМАНТИКА

Операційна (натуральна) семантика задає семантику (тобто спосіб обчислення, пошуку результатів) програм (або взагалі виразів) як покроковий (поопераційний) процес обчислень. Для кожного окремого оператора (згідно із синтаксичним деревом виведення програми в граматиці мови програмування) зіставляється правило обчислення, тобто перетворення поточного даного-стану на нове – результуюче дане-стан – для поточного оператора, з множини правил семантики, пов'язаних з кожним правилом виведення граматики. Правила семантики поділяють на *засновки* та *висновки.* Якщо всі засновки (предикати) правила істинні при обчисленні над поточним станом, то істинними є також висновки (предикати), і обчислення тривають. У такий спосіб будують дерево обчислень, яке повторює шматки дерева виведення програми і трансформує їх в операції з перетворення стану (тобто обчислення). Причому для кожного конкретного кроку побудови дерева обчислень альтернатив застосування того чи іншого правила немає, тобто завжди застосовне лише одне з наявних правил семантики. Процес побудови дерева обчислень буде скінченним, якщо програма зупиняється, а на кроні опиняться операції, що не вимагають додаткових обчислень (взяття значень констант, змінних).

Правила операційної (натуральної) семантики запишемо формулами:

$$\text{SKIP}\ \frac{}{< skip, st_0 > \to st_0};$$

$$\text{AS}\ \frac{< y, st_0 > \to v}{< x := y, st_0 > \to st_1, st_1 = st_0 \nabla [x \mapsto v]};$$

$$\text{BE}\ \frac{< S, st_0 > \to st_1}{< begin\ S\ end, st_0 > \to st_1};$$

$$\text{SEQ}\ \frac{< S_1, st_0 > \to st_2, < S_2, st_2 > \to st_1}{< S_1; S_2, st_0 > \to st_1};$$

$$\text{IF}_{\text{true}}\ \frac{< p, st_0 > \to true, < S_1, st_0 > \to st_1}{< if\ p\ then\ S_1\ else\ S_2, st_0 > \to st_1};$$

$$\text{IF}_{\text{false}}\ \frac{< p, st_0 > \to false, < S_2, st_0 > \to st_1}{< if\ p\ then\ S_1\ else\ S_2, st_0 > \to st_1};$$

$$\text{WH}_{\text{true}}\ \frac{< p, st_0 > \to true, < S, st_0 > \to st_2, < while\ p\ do\ S, st_2 > \to st_1}{< while\ p\ do\ S, st_0 > \to st_1};$$

$$\text{WH}_{\text{false}}\ \frac{< p, st_0 > \to false}{< while\ p\ do\ S, st_0 > \to st_0};$$

$$\text{B}_{>}\ \frac{< A, st > \to a, < B, st > \to b}{< A > B, st > \to r}, \text{ де } r = \begin{cases} true, \text{якщо } a > b \\ false, \text{якщо } a \le b \end{cases};$$

$$\text{A}_{+}\ \frac{< A, st > \to a, < B, st > \to b}{< A + B, st > \to r}, \text{ де } r = a + b.$$

Для всіх арифметичних операцій і булевих обчислень правила аналогічні: A_{+}, A_{-}, A_{*}, $\text{A}_{/}$, $\text{A}_{()}$, $\text{B}_{>}$, $\text{B}_{<}$, $\text{B}_{=}$, $\text{B}_{\neq}$, $\text{B}_{\le}$, $\text{B}_{\ge}$, $\text{B}_{\wedge}$, $\text{B}_{\vee}$, $\text{B}_{\neg}$ або замість усіх цих правил (арифметичні та булеві функції) існує узагальнене правило суперпозиції:

$$\text{A}_{\text{Var}}\ \frac{}{< V, st > \to v}, \text{ якщо } [V \mapsto v] \subseteq st;$$

$$\text{A}_{\text{Num}}\ \frac{}{< c, st > \to c}, \text{ якщо } c \in Z - \text{цілі числа.}$$

Зауваження: для зручності нотації правила часто записують при побудові дерева в перегорнутому вигляді (міняють місцями чисельник і знаменник), щоб контролювати ріст незбалансованого дерева. Так будемо робити далі.

Завдання 4.1. Нехай *DIV*(*A*, *D*) – програма пошуку результату *q* ділення невід'ємного числа *a* на додатне число *d* і залишку від цього ділення *r* (див. завд. 1.4). Обчислити за натуральною семантикою результат застосування програми до даного $[A \mapsto 5, D \mapsto 3]$.

Розв'язання.
Текст програми:
DIV(*A*, *D*) =
begin
    $Q := 0$;
    $R := A$;
    while $R \geq D$ do
    begin
        $Q := Q + 1$;
        $R := R - D$
    end
end

Спочатку введемо скорочення для фрагментів програми:
$P_1 = R := A$; while $R \geq D$ do begin $Q := Q + 1$; $R := R - D$ end
P_2 = while $R \geq D$ do begin $Q := Q + 1$; $R := R - D$ end
P_3 = begin $Q := Q + 1$; $R := R - D$ end
$P_4 = Q := Q + 1$
$P_5 = R := R - D$

Нехай $st_0 = [A \mapsto 5, D \mapsto 3]$, тоді обчислимо результуючий стан *st* застосування програми *Program*(*A*, *D*) до даного st_0:
Sem_P_{Nat} (*DIV*(*A*, *D*)) (st_0) = *st*.

$$\text{BE}\frac{< begin\, Q := 0; P_1\, end, st_0 > \to st}{\text{SEQ}\dfrac{< Q := 0; P_1, st_0 > \to st}{\text{AS}\dfrac{< Q := 0, st_0 > \to st_1, st_1 = st_0 \nabla [Q \mapsto 0]}{\text{A}_{\text{Num}}\dfrac{< 0, st_0 > \to 0}{}}, \dfrac{< P_1, st_1 > \to st}{(1)\text{SEQ}}}},$$

де $st_1 = [A \mapsto 5, D \mapsto 3, Q \mapsto 0]$,

$$(1)\ \mathrm{SEQ}\frac{< R := A \bullet P_2, st_1 > \rightarrow st}{\mathrm{AS}\dfrac{< R := A, st_1 > \rightarrow st_2, st_2 = st_1 \nabla [R \mapsto 5]}{\mathrm{A_{Var}}\dfrac{< A, st_1 > \rightarrow 5}{}}, \dfrac{< P_2, st_2 > \rightarrow st}{(2)\ \mathrm{WH_{true}}}},$$

де $st_2 = [A \mapsto 5, D \mapsto 3, Q \mapsto 0, R \mapsto 5]$,

$$(2)\ \mathrm{WH_{true}}\frac{< P_2, st_2 > \rightarrow st}{\dfrac{< R \geq D, st_2 > \rightarrow true}{(3)\ \mathrm{B_{\geq}}}, \dfrac{< P_3, st_2 > \rightarrow st_3}{(4)\ \mathrm{BE}}, \dfrac{< P_2, st_3 > \rightarrow st}{(7)\ \mathrm{WH_{false}}}},$$

$$(3)\ \mathrm{B_{\geq}}\frac{< R \geq D, st_2 > \rightarrow 5 \geq 3 = true}{\mathrm{A_{Var}}\dfrac{< R, st_2 > \rightarrow 5}{}, \mathrm{A_{Var}}\dfrac{< D, st_2 > \rightarrow 3}{}},$$

$$(4)\ \mathrm{BE}\frac{< begin\ P_4; P_5\ end, st_2 > \rightarrow st_3}{\mathrm{SEQ}\dfrac{< P_4; P_5, st_2 > \rightarrow st_3}{\mathrm{AS}\dfrac{< Q := Q + 1, st_2 > \rightarrow st_4}{\dfrac{< Q + 1, st_2 > \rightarrow 1}{(5)\ \mathrm{A_{+}}}}, (6)\ \mathrm{AS}}},$$

де $st_4 = st_2 \nabla [Q \mapsto 1] = [A \mapsto 5, D \mapsto 3, Q \mapsto 1, R \mapsto 5]$,

$$(5)\ \mathrm{A_{+}}\frac{< Q + 1, st_2 > \rightarrow 1}{\mathrm{A_{Var}}\dfrac{< Q, st_2 > \rightarrow 0}{}, \mathrm{A_{Num}}\dfrac{< 1, st_2 > \rightarrow 1}{}},$$

$$(6)\ \mathrm{AS}\frac{< R := R - D, st_4 > \rightarrow st_3}{\mathrm{A_{-}}\dfrac{< R - D, st_4 > \rightarrow 2}{\mathrm{A_{Var}}\dfrac{< R, st_4 > \rightarrow 5}{}, \mathrm{A_{Var}}\dfrac{< D, st_4 > \rightarrow 3}{}}},$$

де $st_3 = st_4 \nabla [R \mapsto 2] = [A \mapsto 5, D \mapsto 3, Q \mapsto 1, R \mapsto 2]$,

$$(7)\ \mathrm{WH}_{\mathrm{false}}\frac{< P_2, st_3 > \rightarrow st}{\mathrm{B}_{\geq}\dfrac{< R \geq D, st_3 > \rightarrow 2 \geq 3 = false}{\mathrm{A}_{\mathrm{Var}}\dfrac{< R, st_3 > \rightarrow 2}{}, \mathrm{A}_{\mathrm{Var}}\dfrac{< D, st_3 > \rightarrow 3}{}}},$$

де $st = st_3 = [A \mapsto 5, D \mapsto 3, Q \mapsto 1, R \mapsto 2]$ – шукане данерезультат.

Завдання 4.2. Нехай *GCD*(*M*, *N*) – програма обчислення найбільшого спільного дільника за алгоритмом Евкліда (див. завд. 1.1).

Обчислити за натуральною семантикою результат застосування програми *GCD*(*M*, *N*) до даного $[M \mapsto 15, N \mapsto 10]$.

Розв'язання.

GCD(*M*, *N*) =
begin
    while $M \neq N$ do
    if *M*>*N* then *M*: = *M*–*N* else *N*: = *N*–*M*
end

Введемо скорочення для фрагментів програми:

P_1 = while $M \neq N$ do if *M*>*N* then *M*: = *M*–*N* else *N*: = *N*–*M*

P_2 = if *M*>*N* then *M*: = *M*–*N* else *N*: = *N*–*M*

Нехай $st_0 = [M \mapsto 15, N \mapsto 10]$, тоді обчислимо за натуральною семантикою результуючий стан *st* застосування програми *GCD*(*M*, *N*) до даного st_0:

Sem_P_{Nat} (*GCD*(*M*, *N*)) (st_0) = *st*.

$$\mathrm{BE}\frac{< begin\ P_1\ end, st_0 > \rightarrow st}{\mathrm{WH}_{\mathrm{true}}\dfrac{< P_1, st_0 > \rightarrow st}{\dfrac{< M \neq N, st_0 > \rightarrow true}{(1)\,\mathrm{B}_{\neq}}, \dfrac{< P_2, st_0 > \rightarrow st_1}{(2)\,\mathrm{IF}_{\mathrm{true}}}, \dfrac{< P_1, st_1 > \rightarrow st}{(5)\,\mathrm{WH}_{\mathrm{true}}}}},$$

далі

$$(1)\,\mathrm{B}_{\neq}\frac{< M \neq N, st_0 > \rightarrow 15 \neq 10 = true}{\mathrm{A}_{\mathrm{Var}}\dfrac{< M, st_0 > \rightarrow 15}{}, \mathrm{A}_{\mathrm{Var}}\dfrac{< N, st_0 > \rightarrow 10}{}},$$

$$(2)\ \mathrm{IF}_{\mathrm{true}}\ \frac{< \mathit{if}\ M > N\ \mathit{then}\ M := M - N\ \mathit{else}\ N := N - M, st_0 > \rightarrow st_1}{\dfrac{< M > N, st_0 > \rightarrow \mathit{true}}{(3)\ \mathrm{B}_{>}},\ \dfrac{< M := M - N, st_0 > \rightarrow st_1}{(4)\ \mathrm{AS}}},$$

$$(3)\ \mathrm{B}_{>}\ \frac{< M > N, st_0 > \rightarrow 15 > 10 = \mathit{true}}{\mathrm{A}_{\mathrm{Var}}\ \dfrac{< M, st_0 > \rightarrow 15}{},\ \mathrm{A}_{\mathrm{Var}}\ \dfrac{< N, st_0 > \rightarrow 10}{}},$$

$$(4)\ \mathrm{AS}\ \frac{< M := M - N, st_0 > \rightarrow st_1}{\mathrm{A}_{-}\ \dfrac{< M - N, st_0 > \rightarrow 15 - 10 = 5}{\mathrm{A}_{\mathrm{Var}}\ \dfrac{< M, st_0 > \rightarrow 15}{},\ \mathrm{A}_{\mathrm{Var}}\ \dfrac{< N, st_0 > \rightarrow 10}{}}},$$

де $st_1 = st_0 \nabla [M \mapsto 5] = [M \mapsto 5, N \mapsto 10]$,

$$(5)\ \mathrm{WH}_{\mathrm{true}}\ \frac{< P_1, st_1 > \rightarrow st}{\dfrac{< M \neq N, st_1 > \rightarrow \mathit{true}}{(6)\ \mathrm{B}_{\neq}},\ \dfrac{< P_2, st_1 > \rightarrow st_2}{(7)\ \mathrm{IF}_{\mathrm{false}}},\ \dfrac{< P_1, st_2 > \rightarrow st}{(10)\ \mathrm{WH}_{\mathrm{false}}}},$$

$$(6)\ \mathrm{B}_{\neq}\ \frac{< M \neq N, st_1 > \rightarrow 5 \neq 10 = \mathit{true}}{\mathrm{A}_{\mathrm{Var}}\ \dfrac{< M, st_1 > \rightarrow 5}{},\ \mathrm{A}_{\mathrm{Var}}\ \dfrac{< N, st_1 > \rightarrow 10}{}},$$

$$(7)\ \mathrm{IF}_{\mathrm{false}}\ \frac{< \mathit{if}\ M > N\ \mathit{then}\ M := M - N\ \mathit{else}\ N := N - M, st_1 > \rightarrow st_2}{\dfrac{< M > N, st_1 > \rightarrow \mathit{false}}{(8)\ \mathrm{B}_{>}},\ \dfrac{< N := N - M, st_1 > \rightarrow st_2}{(9)\ \mathrm{AS}}},$$

$$(8)\ \mathrm{B}_{>}\ \frac{< M > N, st_1 > \rightarrow 5 > 10 = \mathit{false}}{\mathrm{A}_{\mathrm{Var}}\ \dfrac{< M, st_1 > \rightarrow 5}{},\ \mathrm{A}_{\mathrm{Var}}\ \dfrac{< N, st_1 > \rightarrow 10}{}},$$

$$(9)\ \mathrm{AS}\ \frac{< N := N - M, st_1 > \rightarrow st_2}{\mathrm{A}_{-}\ \dfrac{< N - M, st_1 > \rightarrow 10 - 5 = 5}{\mathrm{A}_{\mathrm{Var}}\ \dfrac{< N, st_1 > \rightarrow 10}{},\ \mathrm{A}_{\mathrm{Var}}\ \dfrac{< M, st_1 > \rightarrow 5}{}}},$$

де $st_2 = st_1 \nabla [N \mapsto 5] = [M \mapsto 5, N \mapsto 5]$,

$$(10)\ \mathrm{WH}_{\mathrm{false}}\ \frac{< P_1, st_2 > \to st}{\mathrm{B}_{\neq}\ \dfrac{< M \neq N, st_2 > \to 5 \neq 5 = false}{\mathrm{A}_{\mathrm{Var}}\ \dfrac{< M, st_2 > \to 5}{},\ \mathrm{A}_{\mathrm{Var}}\ \dfrac{< N, st_2 > \to 5}{}}},$$

де результуючий стан $st = st_2 = [M \mapsto 5, N \mapsto 5]$.

Завдання 4.3. Нехай *Sum_fac*(*N*) – програма обчислення суми факторіалів до вхідного $n \geq 0$ (включно) через множення та додавання одним циклом.

Обчислити за натуральною семантикою результат застосування програми *Sum_fac*(*N*) до даного $[N \mapsto 3]$.

Розв'язання.

Текст програми:

```
Sum_fac(N) =
begin
        I := 0;
        F := 1;
        R := 1;
        while I ≠ N do
        begin
            I := I + 1;
            F := F * I;
            R := R + F
        end
end
```

Спочатку введемо скорочення для фрагментів програми:

$P_1 = I := 0;\ F := 1;\ R := 1;$ while $I \neq N$ do
begin $I := I + 1;\ F := F * I;\ R := R + F$ end

$P_2 = F := 1;\ R := 1;$ while $I \neq N$ do
begin $I := I + 1;\ F := F * I;\ R := R + F$ end

$P_3 = R := 1;$ while $I \neq N$ do
begin $I := I + 1;\ F := F * I;\ R := R + F$ end

$P_4 =$ while $I \neq N$ do begin $I := I + 1;\ F := F * I;\ R := R + F$ end

$P_5 = \text{begin } I := I + 1;\ F := F * I;\ R := R + F \text{ end}$
$P_6 = I := I + 1$
$P_7 = F := F * I$
$P_8 = R := R + F$

Нехай $st_0 = [N \mapsto 3]$, тоді обчислимо результуючий стан st застосування програми *Sum_fac*(*N*) до даного st_0:
Sem_P_{Nat} (*Sum_fac*(*N*)) $(st_0) = st$.

$$\text{BE}\frac{\text{SEQ}\dfrac{\text{AS}\dfrac{\text{A}_{\text{Num}}\dfrac{}{< 0, st_0 > \to 0}}{< I := 0, st_0 > \to st_1}, \dfrac{}{< F := 1; P_3, st_1 > \to st}\ (1)\,\text{SEQ}}{< I := 0; P_2, st_0 > \to st}}{< begin\ P_1\ end, st_0 > \to st},$$

де $st_1 = st_0 \nabla [I \mapsto 0] = [N \mapsto 3, I \mapsto 0]$,

$$(1)\,\text{SEQ}\frac{\text{AS}\dfrac{\text{A}_{\text{Num}}\dfrac{}{< 1, st_1 > \to 1}}{< F := 1, st_1 > \to st_2}, \dfrac{}{< R := 1; P_4, st_2 > \to st}\ (2)\,\text{SEQ}}{< F := 1; P_3, st_1 > \to st},$$

де $st_2 = st_1 \nabla [F \mapsto 1] = [N \mapsto 3, I \mapsto 0, F \mapsto 1]$,

$$(2)\,\text{SEQ}\frac{\text{AS}\dfrac{\text{A}_{\text{Num}}\dfrac{}{< 1, st_2 > \to 1}}{< R := 1, st_2 > \to st_3}, \dfrac{}{< P_4, st_3 > \to st}\ (3)\,\text{WH}_{\text{true}}}{< R := 1; P_4, st_2 > \to st},$$

де $st_3 = st_2 \nabla [R \mapsto 1] = [N \mapsto 3, I \mapsto 0, F \mapsto 1, R \mapsto 1]$,

$$(3)\,\text{WH}_{\text{true}}\frac{\dfrac{}{< I \neq N, st_3 > \to true}\ (4)\,\text{B}_{\neq}, \dfrac{}{< P_5, st_3 > \to st_4}\ (5)\,\text{BE}, \dfrac{}{< P_4, st_4 > \to st}\ (10)\,\text{WH}_{\text{true}}}{< while\ I \neq N\ do\ P_5, st_3 > \to st},$$

$$(4)\,\mathrm{B}_{\neq}\frac{< I \neq N, st_3 > \to 0 \neq 3 = true}{\mathrm{A}_{\mathrm{Var}}\dfrac{< I, st_3 > \to 0}{},\ \mathrm{A}_{\mathrm{Var}}\dfrac{< N, st_3 > \to 3}{}},$$

$$(5)\,\mathrm{BE}\frac{< begin\ P_6; P_7; P_8\ end, st_3 > \to st_4}{\mathrm{SEQ}\dfrac{< I := I+1; P_7; P_8, st_3 > \to st_4}{\dfrac{< I := I+1, st_3 > \to st_5}{(6)\,\mathrm{AS}},\ \dfrac{< P_7; P_8, st_5 > \to st_4}{(7)\ \mathrm{SEQ}}}},$$

$$(6)\,\mathrm{AS}\frac{< I := I+1, st_3 > \to st_5}{\mathrm{A}_{+}\dfrac{< I+1, st_3 > \to 0+1=1}{\mathrm{A}_{\mathrm{Var}}\dfrac{< I, st_3 > \to 0}{},\ \mathrm{A}_{\mathrm{Num}}\dfrac{< 1, st_3 > \to 1}{}}},$$

де $st_5 = st_3 \nabla [I \mapsto 1] = [N \mapsto 3, I \mapsto 1, F \mapsto 1, R \mapsto 1]$,

$$(7)\ \mathrm{SEQ}\frac{< F := F * I; P_8, st_5 > \to st_4}{\dfrac{< F := F * I, st_5 > \to st_6}{(8)\ \mathrm{AS}},\ \dfrac{< P_8, st_6 > \to st_4}{(9)\ \mathrm{AS}}},$$

$$(8)\ \mathrm{AS}\frac{< F := F * I, st_5 > \to st_6}{\mathrm{A}_{*}\dfrac{< F * I, st_5 > \to 1*1=1}{\mathrm{A}_{\mathrm{Var}}\dfrac{< F, st_5 > \to 1}{},\ \mathrm{A}_{\mathrm{Var}}\dfrac{< I, st_5 > \to 1}{}}},$$

де $st_6 = st_5 \nabla [F \mapsto 1] = [N \mapsto 3, I \mapsto 1, F \mapsto 1, R \mapsto 1]$,

$$(9)\ \mathrm{AS}\frac{< R := R + F, st_6 > \to st_4}{\mathrm{A}_{+}\dfrac{< R + F, st_6 > \to 1+1=2}{\mathrm{A}_{\mathrm{Var}}\dfrac{< R, st_6 > \to 1}{},\ \mathrm{A}_{\mathrm{Var}}\dfrac{< F, st_6 > \to 1}{}}},$$

де $st_4 = st_6 \nabla [R \mapsto 2] = [N \mapsto 3, I \mapsto 1, F \mapsto 1, R \mapsto 2]$;

далі:

$$(10)\ \mathrm{WH}_{\mathrm{true}}\frac{< while\, I \neq N\, do\, P_5, st_4 > \rightarrow st}{\underset{(11)\,\mathrm{B}_{\neq}}{< I \neq N, st_4 > \rightarrow true}},\ \underset{(12)\,\mathrm{BE}}{< P_5, st_4 > \rightarrow st_7},\ \underset{(17)\,\mathrm{WH}_{\mathrm{true}}}{< P_4, st_7 > \rightarrow st}},$$

$$(11)\ \mathrm{B}_{\neq}\frac{< I \neq N, st_4 > \rightarrow 1 \neq 3 = true}{\mathrm{A}_{\mathrm{Var}}\underline{< I, st_4 > \rightarrow 1},\ \mathrm{A}_{\mathrm{Var}}\underline{< N, st_4 > \rightarrow 3}},$$

$$(12)\ \mathrm{BE}\frac{< begin\, P_6; P_7; P_8\, end, st_4 > \rightarrow st_7}{\mathrm{SEQ}\dfrac{< I := I+1; P_7; P_8, st_4 > \rightarrow st_7}{\underset{(13)\,\mathrm{AS}}{< I := I+1, st_4 > \rightarrow st_8},\ \underset{(14)\ \mathrm{SEQ}}{< P_7; P_8, st_8 > \rightarrow st_7}}},$$

$$(13)\ \mathrm{AS}\frac{< I := I+1, st_4 > \rightarrow st_8}{\mathrm{A}_{+}\dfrac{< I+1, st_4 > \rightarrow 1+1=2}{\mathrm{A}_{\mathrm{Var}}\underline{< I, st_4 > \rightarrow 1},\ \mathrm{A}_{\mathrm{Num}}\underline{< 1, st_4 > \rightarrow 1}}},$$

де $st_8 = st_4 \nabla [I \mapsto 2] = [N \mapsto 3, I \mapsto 2, F \mapsto 1, R \mapsto 2]$,

$$(14)\ \mathrm{SEQ}\frac{< F := F * I; P_8, st_8 > \rightarrow st_7}{\underset{(15)\ \mathrm{AS}}{< F := F * I, st_8 > \rightarrow st_9},\ \underset{(16)\ \mathrm{AS}}{< P_8, st_9 > \rightarrow st_7}},$$

$$(15)\ \mathrm{AS}\frac{< F := F * I, st_8 > \rightarrow st_9}{\mathrm{A}_{*}\dfrac{< F * I, st_8 > \rightarrow 1 * 2 = 2}{\mathrm{A}_{\mathrm{Var}}\underline{< F, st_8 > \rightarrow 1},\ \mathrm{A}_{\mathrm{Var}}\underline{< I, st_8 > \rightarrow 1}}},$$

де $st_9 = st_8 \nabla [F \mapsto 2] = [N \mapsto 3, I \mapsto 2, F \mapsto 2, R \mapsto 2]$,

$$(16)\ \mathrm{AS}\frac{< R := R + F, st_9 > \rightarrow st_7}{\mathrm{A}_{+}\dfrac{< R + F, st_9 > \rightarrow 2 + 2 = 4}{\mathrm{A}_{\mathrm{Var}}\underline{< R, st_9 > \rightarrow 2},\ \mathrm{A}_{\mathrm{Var}}\underline{< F, st_9 > \rightarrow 2}}},$$

де $st_7 = st_9 \nabla [R \mapsto 4] = [N \mapsto 3, I \mapsto 2, F \mapsto 2, R \mapsto 4]$;

далі:

$$(17)\ \mathrm{WH}_{\mathrm{true}}\frac{< while\ I \neq N\ do\ P_5, st_7 > \rightarrow st}{\underset{(18)\,\mathrm{B}_{\neq}}{< I \neq N, st_7 > \rightarrow true}},\ \underset{(19)\,\mathrm{BE}}{< P_5, st_7 > \rightarrow st_{10}},\ \underset{(24)\,\mathrm{WH}_{\mathrm{false}}}{< P_4, st_{10} > \rightarrow st}},$$

$$(18)\ \mathrm{B}_{\neq}\frac{< I \neq N, st_7 > \rightarrow 2 \neq 3 = true}{\mathrm{A}_{\mathrm{Var}}\dfrac{< I, st_7 > \rightarrow 2}{}},\ \mathrm{A}_{\mathrm{Var}}\frac{< N, st_7 > \rightarrow 3}{},$$

$$(19)\ \mathrm{BE}\frac{< begin\ P_6; P_7; P_8\ end, st_7 > \rightarrow st_{10}}{\mathrm{SEQ}\dfrac{< I := I + 1; P_7; P_8, st_7 > \rightarrow st_{10}}{\underset{(20)\,\mathrm{AS}}{< I := I + 1, st_7 > \rightarrow st_{11}}},\ \underset{(21)\ \mathrm{SEQ}}{< P_7; P_8, st_{11} > \rightarrow st_{10}}}},$$

$$(20)\ \mathrm{AS}\frac{< I := I + 1, st_7 > \rightarrow st_{11}}{\mathrm{A}_{+}\dfrac{< I + 1, st_7 > \rightarrow 2 + 1 = 3}{\mathrm{A}_{\mathrm{Var}}\dfrac{< I, st_7 > \rightarrow 2}{}},\ \mathrm{A}_{\mathrm{Num}}\dfrac{< 1, st_7 > \rightarrow 1}{}}},$$

де $st_{11} = st_7 \nabla [I \mapsto 3] = [N \mapsto 3, I \mapsto 3, F \mapsto 2, R \mapsto 4]$,

$$(21)\ \mathrm{SEQ}\frac{< F := F * I; P_8, st_{11} > \rightarrow st_{10}}{\underset{(22)\ \mathrm{AS}}{< F := F * I, st_{11} > \rightarrow st_{12}}},\ \underset{(23)\ \mathrm{AS}}{< P_8, st_{12} > \rightarrow st_{10}}},$$

$$(22)\ \mathrm{AS}\frac{< F := F * I, st_{11} > \rightarrow st_{12}}{\mathrm{A}_{*}\dfrac{< F * I, st_{11} > \rightarrow 2 * 3 = 6}{\mathrm{A}_{\mathrm{Var}}\dfrac{< F, st_{11} > \rightarrow 2}{}},\ \mathrm{A}_{\mathrm{Var}}\dfrac{< I, st_{11} > \rightarrow 3}{}}},$$

де $st_{12} = st_{11} \nabla [F \mapsto 6] = [N \mapsto 3, I \mapsto 3, F \mapsto 6, R \mapsto 4]$,

$$(23)\ \mathrm{AS}\frac{< R := R + F, st_{12} > \rightarrow st_{10}}{\mathrm{A}_{+}\dfrac{< R + F, st_{12} > \rightarrow 4 + 6 = 10}{\mathrm{A}_{\mathrm{Var}}\dfrac{< R, st_{12} > \rightarrow 4}{}},\ \mathrm{A}_{\mathrm{Var}}\dfrac{< F, st_{12} > \rightarrow 6}{}}},$$

де $st_{10} = st_{12} \nabla [R \mapsto 10] = [N \mapsto 3, I \mapsto 3, F \mapsto 6, R \mapsto 10]$;

нарешті,

$$(24)\ \mathrm{WH}_{\mathrm{false}}\frac{< \mathit{while}\, I \neq N\, \mathit{do}\, P_5, st_{10} > \to st}{\mathrm{B}_{\neq}\dfrac{< I \neq N, st_{10} > \to 3 \neq 3 = \mathit{false}}{\mathrm{A}_{\mathrm{Var}}\dfrac{< I, st_{10} > \to 3}{},\ \mathrm{A}_{\mathrm{Var}}\dfrac{< N, st_{10} > \to 3}{}}}.$$

Отримали результат:

$st = st_{10} = [N \mapsto 3, I \mapsto 3, F \mapsto 6, R \mapsto 10]$.

Завдання для самоконтролю

Для задач, наведених у завданнях для самоконтролю в розд. 3, зробити таке:

1) написати SIPL-програму для задачі;

2) протестувати побудовану програму в операційній семантиці для вказаних вхідних даних;

3) довести правильність побудованої програми.

Приклади завдань для контрольної роботи

Варіант 1

I. Визначення операційної семантики.

II. Для функції обчислення F_n (n-го елемента ряду Фібоначчі, $n>0$) зробити таке:

1) написати SIPL-програму;

2) протестувати побудовану програму в операційній семантиці для вхідних даних $n = 3$;

3) довести правильність побудованої програми.

Варіант 2

I. Доведення теореми про еквівалентність операційної та композиційної семантик.

II. Для функції обчислення C_x^y (x, y>0) зробити таке:

1) написати SIPL-програму;

2) протестувати побудовану програму в операційній семантиці на вхідних даних $x = 4$, $y = 3$;

3) довести правильність побудованої програми.

Варіант 3

I. Доведення коректності програм для операційної семантики.

II. Для функції обчислення 3^x (x>0) зробити таке:

1) написати SIPL-програму, використовуючи функції ∗,+, –;

2) протестувати побудовану програму в операційній семантиці апроксимацій на вхідних даних $x = 2$;

3) довести правильність побудованої програми.

Варіант 4

I. Доведення вдастивостей програм для операційної семантики.

II. Для функції $[\log_2 n]$ (n>0) зробити таке:

1) написати SIPL-програму, використовуючи функції *div*, *mod*, +, –;

2) протестувати побудовану програму в операційній семантиці на вхідних даних $n = 3$;

3) довести правильність побудованої програми.

РОЗДІЛ 5
АКСІОМАТИЧНА СЕМАНТИКА

Логіка Хоара є численням, формули якого пов'язують програми (оператори) з імперативною мовою програмування з парами тверджень – логічними формулами першого порядку, які називають *передумовою* та *післяумовою* (*постумовою*) і записують у вигляді {передумова} <оператор> {післяумова}. Твердження інтерпретуються як предикати на станах (вільним змінним відповідають змінні програми в даному стані) і задають відповідно множини станів, на яких вони істинні (формула $y>0$ задає множину станів, у яких змінна y додатна). Якщо перед виконанням оператора істинна передумова, то після його виконання буде істинною післяумова – таким чином виконання коду змінює стан програми – дане. Логіка Хоара надає аксіоми та правила виведення для всіх базових операторів мови програмування, а за допомогою виведень у логіці можна доводити властивості як окремих операторів, так і всієї програми, наприклад часткову коректність. Тому логіка Хоара виступає інструментом формальної верифікації програм. Якщо відома формальна специфікація (тобто в першу чергу – післяумова) програми, то можна, рухаючись від післяумови до початку програми, знайти найслабшу передумову (weakest precondition) для кожного оператора, а отже, і для всієї програми в цілому – тобто умову до вхідного стану, за якої програма працюватиме коректно.

Аксіоми та правила виводу аксіоматичної семантики Хоара:

$$\text{SKIP}\ \frac{}{\{P\}\ skip\ \{P\}}$$

$$\text{BE}\ \frac{\{P\}\ S\ \{Q\}}{\{P\}\ begin\, S\, end\ \{Q\}}$$

$$\text{AS}\frac{}{\{P[e/x]\}\ X := E\ \{P\}}, \text{ де } e = sem\ (E),\ x = X \Rightarrow$$

$$\text{SEQ}\frac{\{P\}\ S\ \{Q\}, \{Q\}\ T\ \{R\}}{\{P\}\ S;T\ \{R\}}$$

$$\text{IF}\frac{\{b \wedge P\}\ S\ \{Q\}, \{\neg b \wedge P\}\ T\ \{Q\}}{\{P\}\ if\ B\ then\ S\ else\ T\ \{Q\}}, \text{ де } b = sem\ (B)$$

$$\text{WH}\frac{\{P \wedge b\}\ S\ \{P\}}{\{P\}\ while\ B\ do\ S\ \{\neg b \wedge P\}}, \text{ де } b = sem\ (B)$$

$$\text{CONS}\frac{P' \to P, \{P\}\ S\ \{Q\}, Q \to Q'}{\{P'\}\ S\ \{Q'\}}$$

Зауваження: для зручності нотації правила часто записують при побудові дерева в перегорнутому вигляді (міняють місцями чисельник і знаменник), щоб контролювати ріст незбалансованого дерева. Так і будемо робити далі.

Як завжди, для зручності назви змінних позначимо великими літерами, їх значення – відповідними маленькими.

Завдання 5.1. Довести часткову коректність програми *DIV*(*A*, *D*) (ділення в цілих числах, див. завд. 1.4) за допомогою логіки Хоара (в аксіоматичній семантиці).

Розв'язання.

Текст програми:

DIV(*A*, *D*) =

```
begin
    Q := 0;
    R := A;
    while R ≥ D do
    begin
            Q := Q + 1;
            R := R – D
    end
end
```

Спочатку введемо скорочення для фрагментів програми:

$S_1 = R := A$; while $R \geq D$ do begin $Q := Q + 1$; $R := R - D$ end

$S_2 =$ while $R \geq D$ do begin $Q := Q + 1$; $R := R - D$ end

$S_3 =$ begin $Q := Q + 1$; $R := R - D$ end

$S_4 = Q := Q + 1$

$S_5 = R := R - D$

Необхідно перевірити трійку Хоара на істинність:

$\{a \geq 0 \wedge d > 0\}\, DIV(A, D)\, \{q = [a / d] \wedge r = a - q * d\}$,

де в післяумові відображено, що q – ціла частина від ділення a на d, при цьому r – залишок від ділення a на d.

$[\, a / d\,]$ – позначення операції цілочисельного ділення.

Ураховуючи, що $DIV(A, D) =$ begin $Q := 0$; S_1 end, маємо таке виведення в логіці Хоара:

$$\text{BE}\frac{\{P_0\}\, DIV(A,D)\, \{P_1\}}{\text{SEQ}\dfrac{\{P_0\}\, Q := 0; S_1\, \{P_1\}}{\text{AS}\dfrac{\{P_0\}\, Q := 0\, \{P_2\}}{}, \dfrac{\{P_2\}\, R := A; S_2\, \{P_1\}}{\text{AS}\dfrac{\{P_2\}\, R := A\, \{P_3\}}{}, \dfrac{\{P_3\}\, S_2\, \{P_1\}}{\text{(1) CONS}}}}},$$

$$\text{(1) CONS}\frac{\{P_3\}\, while\, R \geq D\, do\, S_3\, \{P_1\}}{P_3 \to P_3, \text{WH}\dfrac{\{P_3\}\, while\, R \geq D\, do\, S_3\, \{P_6\}}{\{P_5\}\, begin\, S_4; S_5\, end\, \{P_3\}}, P_6 \to P_1},$$

$$\text{(2) BE}$$

$$\text{(2) BE}\frac{\{P_5\}\, begin\, S_4; S_5\, end\, \{P_3\}}{\text{SEQ}\dfrac{\{P_5\}\, S_4; S_5\, \{P_3\}}{\text{AS}\dfrac{\{P_5\}\, Q := Q + 1\, \{P_4\}}{}, \text{AS}\dfrac{\{P_4\}\, R := R - D\, \{P_3\}}{}}}.$$

Доведення виконуємо з кінця, відштовхуючись від предиката-післяумови всієї програми й рухаючись до її початку, щоб вивести предикат-передумову. При цьому предикати набувають вигляду: інваріант циклу – $P = (q * d + r = a \wedge d > 0 \wedge r \geq 0)$; далі

$P_1 = (q = [a/d] \wedge r = a - q * d)$ – предикат-післяумова (див. вище),

$P_6 = P \wedge \neg(r \geq d) = (q * d + r = a \wedge 0 \leq r \wedge r < d \wedge d > 0)$.

Ураховуючи, що q – ціле число, $P_6 \Rightarrow q = (a - r)/d$, з обмежень на r маємо: $0 \leq r < d \Rightarrow a/d - 1 < (a - r)/d \leq a/d \Rightarrow$

$\Rightarrow a/d - 1 \leq q < a/d \Rightarrow q = [a/d]$, отже, $P_6 \rightarrow P_1 = true$.

Згідно з правилом WH: $P_3 = P = (q * d + r = a \wedge d > 0 \wedge r \geq 0)$,

$P_5 = P \wedge (r \geq d) = (q * d + r = a \wedge r \geq d \wedge d > 0)$, тоді

$P_4 = P_3[r - d/r] = (q * d + r = a \wedge d > 0 \wedge r \geq 0)[r - d/r] =$

$= (q * d + r - d = a \wedge d > 0 \wedge r \geq d)$.

Тепер $P_4[q + 1/q] = (q * d + r - d = a \wedge d > 0 \wedge r \geq d)[q + 1/q] =$

$= ((q + 1) * d + r - d = a \wedge d > 0 \wedge r \geq d) =$

$= (q * d + r = a \wedge d > 0 \wedge r \geq d) = P_5$.

Отже, правило WH застосоване коректно. Залишилось зробити такі перетворення:

$P_2 = P_3[a/r] = (q * d + a = a \wedge d > 0 \wedge r \geq 0) = (q = 0 \wedge d > 0 \wedge a \geq 0)$,

$P_0 = P_2[0/q] = (0 = 0 \wedge d > 0 \wedge a \geq 0) = (d > 0 \wedge a \geq 0)$.

Таким чином, передумовою коректної роботи програми буде предикат $P_0 = d > 0 \wedge a \geq 0$, що і треба було довести.

Завдання 5.2. Довести часткову коректність програми *GCD*(*M*, *N*) (обчислення найбільшого спільного дільника за алгоритмом Евкліда, див. завд. 1.1) за допомогою логіки Хоара (в аксіоматичній семантиці).

Розв'язання.

Текст програми із завд. 1.1 необхідно скорегувати, адже для доведень в аксіоматичній семантиці Хоара вимагається, щоб вхідні змінні не змінювали значень протягом виконання програми:

$GCD(M, N) =$
begin
    $A := M$;
    $B := N$;
    while $A \neq B$ do
        if $A > B$ then
            $A := A - B$
        else
            $B := B - A$
end

Спочатку введемо скорочення для фрагментів програми:

$S_1 = B := N$; while $A \neq B$ do
if $A > B$ then $A := A - B$ else $B := B - A$ end
$S_2 =$ while $A \neq B$ do if $A > B$ then $A := A - B$ else $B := B - A$ end
$S_3 =$ if $A > B$ then $A := A - B$ else $B := B - A$
$S_4 = A := A - B$
$S_5 = B := B - A$

Необхідно перевірити таку трійку Хоара на істинність:

$$\{m > 0 \wedge n > 0\}\, GCD(M,\ N)\, \{a = \gcd(m, n) \wedge b = \gcd(m, n)\},$$

де в післяумові відображено, що значення змінних A та B дорівнюють найбільшому спільному дільнику (gcd) початкових значень змінних M та N.

Ураховуючи, що $GCD(M, N) =$ begin $A := M$; S_1 end, маємо таке виведення в логіці Хоара:

$$\text{CONS}\frac{\dfrac{\{P_0\}\, GCD(M, N)\, \{P_1\}}{P_0 \to P_0,\ \text{BE}\dfrac{\{P_0\}\, GCD(M, N)\, \{P_7\}}{\{P_0\}\, A := M; S_1\, \{P_7\}},\ P_7 \to P_1}}{\text{(1) SEQ}},$$

$$(1)\ \text{SEQ}\frac{\{P_0\}\,A := M;S_1\,\{P_7\}}{\text{AS}\dfrac{\{P_0\}\,A := M\,\{P_2\}}{},\ \text{SEQ}\dfrac{\{P_2\}\,B := N;S_2\,\{P_7\}}{\text{AS}\dfrac{\{P_2\}\,B := N\,\{P_3\}}{},\ \dfrac{\{P_3\}\,S_2\,\{P_7\}}{(2)\ \text{WH}}}},$$

$$(2)\ \text{WH}\frac{\{P_3\}\,while\ A \neq B\ do\ S_3\,\{P_7\}}{\text{IF}\dfrac{\{P_4\}\,if\ A > B\ then\ S_4\ else\ S_5\,\{P_3\}}{\text{AS}\dfrac{\{P_5\}\,A := A - B\,\{P_3\}}{},\ \text{AS}\dfrac{\{P_6\}\,B := B - A\,\{P_3\}}{}}}.$$

Перш ніж перейти безпосередньо до доведення, нагадаємо властивості функції gcd(*a*, *b*): $\gcd(a,b) = \gcd(a-b,b)$, якщо $a > b$ та $\gcd(a,b) = \gcd(a,b-a)$, якщо $a < b$. Обґрунтування цього факту таке: кожний спільний дільник *d* чисел $a = k*d$ та $b = n*d$ є також дільником $a-b = (k-n)*d$ (якщо $a > b$) або $b-a = (n-k)*d$ (якщо $a < b$). Оскільки це справедливо для всіх дільників *a* та *b*, то всі спільні дільники пар чисел (a,b) та $(a-b,b)$ (або $(b-a,a)$) збігаються, а отже, найбільший спільний дільник у цих пар чисел однаковий. Також $\gcd(a,a) = a$.

Доведення виконуємо з кінця, відштовхуючись від предиката-післяумови всієї програми й рухаючись до її початку, щоб вивести предикат-передумову. При цьому предикати набувають вигляду:

інваріант циклу $P = (\gcd(a,b) = \gcd(m,n) \wedge a > 0 \wedge b > 0)$;

$P_1 = (a = \gcd(m,\ n) \wedge b = \gcd(m,\ n))$ – предикат-післяумова (див. вище);

предикат-післяумова циклу WH:

$P_7 = \neg(a \neq b) \wedge P = (a = b) \wedge P,$ при цьому:

$$P_7 \to P_1 = (a = b \wedge P) \to (a = \gcd(m,n) \wedge b = \gcd(m,n)) =$$
$$= (a = b \wedge \gcd(a,b) = \gcd(m,n) \wedge a > 0 \wedge b > 0) \to$$
$$\to (a = \gcd(m,n) \wedge b = \gcd(m,n)) =$$
$$= \neg(a = b \wedge \gcd(a,b) = \gcd(m,n) \wedge a > 0 \wedge b > 0) \vee$$
$$\vee (a = b \wedge \gcd(a,b) = \gcd(m,n)) =$$
$$= \neg(a = b \wedge \gcd(a,b) = \gcd(m,n)) \vee \neg(a > 0 \wedge b > 0) \vee$$
$$\vee (a = b \wedge \gcd(a,b) = \gcd(m,n)) = true,$$

адже $\neg Q \vee T \vee Q = true$,

$P_3 = P = (\gcd(a,b) = \gcd(m,n) \wedge a > 0 \wedge b > 0)$ як інваріант циклу,

$P_4 = (a \neq b) \wedge P = (a \neq b \wedge \gcd(a,b) = \gcd(m,n) \wedge a > 0 \wedge b > 0)$

як передумова тіла циклу при застосуванні правила WH,

$P_5 = P_3[a - b / a] = (\gcd(a-b,b) = \gcd(m,n) \wedge a - b > 0 \wedge b > 0);$

потім, згідно з властивістю gcd, отримуємо

$P_5 = (\gcd(a,b) = \gcd(m,n) \wedge (a > b) \wedge b > 0) = a > b \wedge P = a > b \wedge P_4,$

$P_6 = P_3[b - a / b] = (\gcd(a,b-a) = \gcd(m,n) \wedge a > 0 \wedge b - a > 0);$

далі, згідно з властивістю gcd,

$P_6 = (\gcd(a,b) = \gcd(m,n) \wedge (b > a) \wedge a > 0) = a < b \wedge P =$

$= (a \leq b \wedge a \neq b) \wedge P = \neg(a > b) \wedge ((a \neq b) \wedge P) = \neg(a > b) \wedge P_4$

згідно з правилом IF.

Тепер

$P_2 = P_3[n / b] = (\gcd(a,n) = \gcd(m,n) \wedge a > 0 \wedge n > 0),$

$P_0 = P_2[m / a] = (\gcd(m,n) = \gcd(m,n) \wedge m > 0 \wedge n > 0) =$

$= (m > 0 \wedge n > 0).$

Отже, передумовою коректної роботи програми буде предикат $P_0 = m > 0 \wedge n > 0$, що і треба було довести.

Завдання 5.3. Довести часткову коректність програми *Sum_fac*(*N*) обчислення суми факторіалів до вхідного $n \geq 0$ (включно) через множення й додавання (одним циклом, див. завд. 4.3) за допомогою логіки Хоара (в аксіоматичній семантиці).

Розв'язання.

Текст програми:

Sum_fac(*N*) =

= begin

$I := 0;$

$F := 1;$

$R := 1;$

while $I \neq N$ do

```
    begin
        I := I + 1;
        F := F * I;
        R := R + F
    end
end
```

Спочатку введемо скорочення для фрагментів програми:
$S_1 = I := 0;\ F := 1;\ R := 1;$ while $I \neq N$ do
begin $I := I + 1;\ F := F * I;\ R := R + F$ end
$S_2 = F := 1;\ R := 1;$ while $I \neq N$ do
begin $I := I + 1;\ F := F * I;\ R := R + F$ end
$S_3 = R := 1;$ while $I \neq N$ do
begin $I := I + 1;\ F := F * I;\ R := R + F$ end
$S_4 =$ while $I \neq N$ do begin $I := I + 1;\ F := F * I;\ R := R + F$ end
$S_5 =$ begin $I := I + 1;\ F := F * I;\ R := R + F$ end
$S_6 = I := I + 1$
$S_7 = F := F * I$
$S_8 = R := R + F$

Необхідно перевірити таку трійку Хоара на істинність:

$$\{n \geq 0\}\, Sum\_fac(N)\, \{r = \sum_{j=0}^{n} j!\},$$

де в післяумові відображено, що значення змінної R дорівнює сумі факторіалів усіх чисел від 0 до n (початкове значення змінної N).

Ураховуючи, що $Sum\_fac(N) =$ begin S_1 end, маємо таке виведення в логіці Хоара:

$$\text{CONS}\frac{\{P_0\}\, Sum\_fac(N)\, \{P_8\}}{P_0 \to P_0,\ \text{BE}\dfrac{\{P_0\}\, begin\, S_1\, end\, \{P_1\}}{\dfrac{\{P_0\}\, I := 0; S_2\, \{P_1\}}{\text{(1) SEQ}}},\ P_1 \to P_8},$$

$$\text{(1) SEQ}\frac{\{P_0\}\, I := 0; S_2\, \{P_1\}}{\text{AS}\dfrac{\{P_0\}\, I := 0\, \{P_2\}}{},\ \dfrac{\{P_2\}\, F := 1; S_3\, \{P_1\}}{\text{(2) SEQ}}},$$

$$(2)\ \text{SEQ}\frac{\{P_2\}F:=1;S_3\{P_1\}}{\text{AS}\dfrac{\{P_2\}F:=1\{P_3\}}{},\ \text{SEQ}\dfrac{\{P_3\}R:=1;S_4\{P_1\}}{\text{AS}\dfrac{\{P_3\}R:=1\{P_4\}}{},\ \dfrac{\{P_4\}S_4\{P_1\}}{(3)\ \text{WH}}}},$$

$$(3)\ \text{WH}\frac{\{P_4\}\,while\,I\neq N\,do\,S_5\{P_1\}}{\text{BE}\dfrac{\{P_5\}\,begin\,S_6;S_7;S_8\,end\,\{P_4\}}{\text{SEQ}\dfrac{\{P_5\}\ S_6;S_7;S_8\ \{P_4\}}{\dfrac{\{P_5\}I:=I+1\{P_6\}}{(4)\ \text{CONS}},\ \dfrac{\{P_6\}S_7;S_8\{P_4\}}{(5)\ \text{SEQ}}}}},$$

$$(4)\ \text{CONS}\frac{\{P_5\}I:=I+1\{P_6\}}{P_5\to P_9,\ \text{AS}\dfrac{\{P_9\}I:=I+1\{P_6\}}{},\ P_6\to P_6},$$

$$(5)\ \text{SEQ}\frac{\{P_6\}F:=F*I;S_8\{P_4\}}{\text{AS}\dfrac{\{P_6\}F:=F*I\{P_7\}}{},\ \text{AS}\dfrac{\{P_7\}R:=R+F\{P_4\}}{}}.$$

Доведення будуємо з кінця, відштовхуючись від предиката-післяумови всієї програми й рухаючись до її початку, щоб вивести предикат-передумову. При цьому предикати набувають вигляду:

інваріант циклу: $P=(r=\sum_{j=0}^{i}j!\wedge f=i!\wedge n\geq 0)$;

$P_8=(r=\sum_{j=0}^{n}j!)$ – предикат-післяумова (див. вище);

предикат-післяумова циклу WH:

$P_1=\neg(i\neq n)\wedge P=(i=n)\wedge P$, при цьому

$$P_1\to P_8=(i=n\wedge P)\to(r=\sum_{j=0}^{n}j!)=$$

$$=(i=n\wedge r=\sum_{j=0}^{i}j!\wedge f=i!\wedge n\geq 0)\to(r=\sum_{j=0}^{n}j!)=$$

$$=(r=\sum_{j=0}^{n}j!\wedge i=n\wedge f=i!\wedge n\geq 0)\to(r=\sum_{j=0}^{n}j!)=true,$$

адже $(Q \wedge T) \to Q = true.$

$P_4 = P = (r = \sum_{j=0}^{i} j! \wedge f = i! \wedge n \geq 0)$ як інваріант циклу,

$P_5 = (i \neq n) \wedge P = (i \neq n \wedge r = \sum_{j=0}^{i} j! \wedge f = i! \wedge n \geq 0)$ як передумова тіла циклу при застосуванні правила WH,

$$P_7 = P_4[r + f / r] = (r + f = \sum_{j=0}^{i} j! \wedge f = i! \wedge n \geq 0),$$

$$P_6 = P_7[f * i / f] = (r + f * i = \sum_{j=0}^{i} j! \wedge f * i = i! \wedge n \geq 0),$$

$$P_9 = P_6[i+1 / i] = (r + f * (i+1) = \sum_{j=0}^{i+1} j! \wedge f * (i+1) = (i+1)! \wedge n \geq 0) =$$

$$= (r + (i+1)! = \sum_{j=0}^{i} j! + (i+1)! \wedge f * (i+1) = i! * (i+1) \wedge n \geq 0) =$$

$$= (r = \sum_{j=0}^{i} j! \wedge f = i! \wedge n \geq 0).$$

Тепер

$$P_5 \to P_9 = (i \neq n \wedge r = \sum_{j=0}^{i} j! \wedge f = i! \wedge n \geq 0) \to$$

$$\to (r = \sum_{j=0}^{i} j! \wedge f = i! \wedge n \geq 0) = true,$$

оскільки $(T \wedge Q) \to Q = true.$

Звідси

$$P_3 = P_4[1 / r] = (1 = \sum_{j=0}^{i} j! \wedge f = i! \wedge n \geq 0),$$

$$P_2 = P_3[1/f] = (1 = \sum_{j=0}^{i} j! \wedge 1 = i! \wedge n \geq 0),$$

$$P_0 = P_2[0/i] = (1 = \sum_{j=0}^{0} j! \wedge 1 = 0! \wedge n \geq 0) = (n \geq 0).$$

Отже, передумовою коректної роботи програми буде предикат $P_0 = n \geq 0$, що і треба було довести.

Завдання для самоконтролю

Для задач, наведених у завданнях для самоконтролю в розд. 3, зробити таке:

1) написати SIPL-програму для задачі;

2) довести правильність побудованої програми в аксіоматичній програмі.

Приклади завдань для контрольної роботи

Варіант 1

I. Визначення операційної семантики

II. Для функції обчислення F_n (n-го елемента ряду Фібоначчі, $n>0$) зробити таке:

1) написати SIPL-програму;

2) довести правильність побудованої програми в аксіоматичній програмі.

Варіант 2

I. Доведення теореми про еквівалентність операційної та композиційної семантик.

II. Для функції обчислення C_x^y ($x, y>0$) зробити таке:

1) написати SIPL-програму;

2) довести правильність побудованої програми в аксіоматичній програмі.

Варіант 3

I. Доведення коректності програм для операційної семантики.

II. Для функції обчислення 3^x ($x>0$) зробити таке:

1) написати SIPL-програму, використовуючи функції ∗,+, –;

2) довести правильність побудованої програми в аксіоматичній програмі.

Варіант 4

I. Доведення вдастивостей програм для операційної семантики.

II. Для функції $[\log_2 n]$ ($n>0$) зробити таке:

1) написати SIPL-програму, використовуючи функції *div*, *mod*, +, –;

2) довести правильність побудованої програми в аксіоматичній програмі.

ЛІТЕРАТУРА

1. **Басараб И. А.** Композиционные базы данных / И. А. Басараб, Н. С. Никитченко, В. Н. Редько. – К.: Либідь, 1992.
2. **Никитченко Н. С.** Композиционная семантика языков программирования / Н. С. Никитченко // Программирование. – 1982. – № 6. – С. 9–18.
3. **Нікітченко М. С.** Структури даних в композиційних мовах програмування / М. С. Нікітченко, Т. В. Панченко // Вісн. Київ. ун-ту. Серія: фіз.-мат. науки. – 2004. – Вип. 2. – С. 316–325.
4. **Нікітченко М. С.** Теорія програмування / М. С. Нікітченко. – Ніжин, 2010. – Ч. 1.
5. **Панченко Т. В.** Моделювання структур даних та функцій над ними в композиційно-номінативній мові ACoN / Т. В. Панченко // Проблемы программирования. – 2004. – № 1-2. – С. 7–15.
6. **Панченко Т. В.** Композиційні методи специфікації та верифікації програмних систем : дис. канд. фіз.-мат. наук / Т. В. Панченко. – К., 2006.
7. **Редько В. Н.** Семантические структуры программ / В. Н. Редько // Программирование. – 1981. – № 1. – С. 3–19.
8. **Nielson H. R.** Semantics with Applications: A Formal Introduction / H. R. Nielson, F. Nielson. – Wiley Professional Computing, 1992.
9. **Nikitchenko N. A.** Composition Nominative Approach to Program Semantics : Technical Report IT-TR: 1998-020 / N. A. Nikitchenko. – Technical University of Denmark. – 1998.
10. **Winskel G.** The Formal Semantics of Programming Languages: An Introduction / G. Winskel. – London: MIT Press Foundations of Computing Series, 1993.

Навчальне видання

НІКІТЧЕНКО Микола Степанович,
ПАНЧЕНКО Тарас Володимирович,
ПОЛЯКОВ Сергій Анатолійович

ТЕОРІЯ ПРОГРАМУВАННЯ В ПРИКЛАДАХ І ЗАДАЧАХ

Навчальний посібник

Редактор *Н. Земляна*

Оригінал-макет виготовлено ВПЦ "Київський університет"
Виконавець *О. Бондаренко*

Формат 60x84$^{1/16}$. Ум. друк. арк. 11,1. Наклад 150. Зам. № 215-7404.
Гарнітура Times New Roman. Папір офсетний. Друк офсетний. Вид. № К5.
Підписано до друку 07.09.15

Видавець і виготовлювач
ВПЦ "Київський університет"
б-р Т. Шевченка 14, м. Київ, 01601
☎ (044) 239 32 22; (044) 239 31 72; тел./факс (044) 239 31 28
e-mail: vpc@univ.kiev.ua
http: vpc.univ.kiev.ua
Свідоцтво суб'єкта видавничої справи ДК № 1103 від 31.10.02