

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Л.Л. ОМЕЛЬЧУК

**ФОРМАЛЬНІ МЕТОДИ СПЕЦИФІКАЦІЇ
ПРОГРАМ**

Навчальний посібник

Рецензенти:

к-т фіз.-мат. наук В.В. Зубенко,

к-т фіз.-мат. наук С.С. Шкільняк

*Рекомендовано до друку вченою радою факультету кібернетики (протокол
№ 5 від 21 грудня 2009 року)*

Омельчук Л.Л. Формальні методи специфікації програм. – К.: УкрІНТЕІ,
2010. - 78 с.

Викладено матеріали до дисципліни за блоком «Формальні методи специфікації програм». Розглянуто поняття: специфікація, методи та мови специфікацій та їх класифікація, мова Z та її місце в класифікації, основні поняття мови Z. Наведено вправи для самостійної роботи.

Для студентів четвертого курсу кафедри теорії та технології програмування факультету кібернетики.

© Л.Л. Омельчук, 2010

Частина 1

МОВИ ТА МЕТОДИ СПЕЦИФІКАЦІЇ ПРОГРАМ

Постійне розширення сфери застосування обчислювальної техніки та необхідність побудови все більш складних програмних систем загострює проблему швидкого та економічного конструювання надійного програмного забезпечення. Сучасні програмні системи характеризуються великою кількістю елементів даних, функцій та підсистем, кожна з яких розв'язує свої локальні задачі і має свої функції. Разом з тим, ускладнення програмних комплексів, а також збільшення залежності людей від правильного функціонування систем, викликає зростання вимог до їх надійності. В багатьох випадках “ручні” методи розробки програмного забезпечення стають незадовільними. Постала задача автоматизації процесів розробки програмного забезпечення (ПЗ).

1.1. Достовірність тестування ПЗ

Сьогодні надзвичайно важливою є задача розробки надійного програмного забезпечення.

Тестування програм – фактично єдиний засіб, що традиційно використовується для перевірки правильності програм. Проте, тестування може лише знайти помилку, а не довести, що ПЗ не має помилок.

Від помилок незастрахована жодна програма, в тому числі і ПЗ, критичне до безпеки. Але помилки в такому ПЗ можуть викликати надзвичайні наслідки. Оборона, авіація, космос, медицина, технологічні процеси на сучасних ядерних, хімічних та інших виробництвах є неповним списком предметних областей де низька якість ПЗ веде до людських жертв та великих матеріальних витрат.

Вичерпне тестування складних ПС в принципі неможливе: можливо перевірити лише невелику кількість з усього простору можливих станів системи. В результаті, тестування може продемонструвати наявність помилок, а не надати гарантію їх відсутності.

Що-ж стосується використання математичних методів для верифікації ПЗ в плані його відповідності специфікації, то воно поки що не увійшло в практику в значному масштабі.

1.2. Методи формальної розробки, мови специфікацій

Формальна специфікація – завершений опис моделі системи та вимог до її поведінки в термінах того чи іншого формального методу.

Формальний метод (метод формальної розробки ПС) – це набір методів та інструментальних засобів, що базуються на математичних засадах (моделювання, математична логіка, теорія множин, теорія скінченних автоматів, алгебра, тощо) та використовуються для формальної специфікації, верифікації, аналізу вимог до ПС та предметних областей. Такі інструментальні засоби іноді ще називають системою формальних міркувань. Як правило, крім системи формальних міркувань формальні методи включають стандартизовані мови (мови специфікацій, формальні нотації). Приклади формальних методів: CSP[4], CCS[5], OBJ[6], VDM[7, 8, 9], В-метод[7], Z-метод [7, 10, 11, 12], RAISE[7, 13, 14, 15, 16].

Формальна мова (мова специфікацій, формальна нотація) – це мова з точно визначеним синтаксисом та семантикою. Мови формальних специфікацій використовуються для написання специфікації, тобто для опису властивостей певної предметної області. Це, наприклад, мови Z [7, 10, 11, 12], В [7], CLEAR [17], LARCH [18, 19, 20].

Формальна розробка – систематичне перетворення специфікації у виконуваний код з використанням певних формалізованих правил (певних формальних методів).

Методи формальної розробки базуються на використанні мови специфікацій та засобів формальних міркувань для побудови ПС. При цьому засоби формальних міркувань базуються на мовах формальних специфікацій, математичних засадах та використовуються для аналізу моделі системи та самої системи. Мови формальних специфікацій в свою чергу базуються на математичному апараті (наприклад, числення предикатів, алгебра, теорія скінченних автоматів) та використовуються для побудови моделі системи.

Існує низка підходів до методів формальної розробки:

- операційний підхід – розробка системи базується на описі моделі, яка має властивості розроблюваної системи;
- аксіоматичний підхід – розробка базується на аксіоматичному описі поведінки системи;
- алгебричний підхід – розробка базується на алгебричному описі властивостей дій;
- гібридний підхід об'єднує операційний підхід з аксіоматичним чи алгебричним підходами.

Кроки при формальній розробці об'єднані та супроводжуються аналізом, який базується на перевірці властивостей системи. Така перевірка здійснюється засобами формальних міркувань за допомогою двох основних підходів.

- Автоматичне доведення теорем (theorem proving) – доведення теорем (логічних тверджень про властивості системи) за допомогою програмного забезпечення. В основі автоматичного доведення теорем лежить апарат математичної логіки.
- Перевірка моделі (model checking) – процес перевірки, чи є побудована структура моделлю заданої предметної області.

1.3. Досвід використання формальних методів

Існує ряд прикладів використання формальних методів (ФМ), та зокрема методу *Z* для специфікації апаратних систем.

IBM використовувала метод *Z* для ре-специфікації великої (більше півмільйона рядків коду) системи обробки транзакцій CICS, в результаті чого покращила її підтримуваність. Крім того, для проекту CICS, незалежний аудит підтвердив економію коштів у розмірі 9% завдяки застосуванню специфікацій на *Z*, що зменшило кількість помилок та підвищило якість коду.

Британська компанія Plessey розробила специфікації для комп'ютерної архітектури надійної мультиобробки інформації Type Environment. Модуль роботи з числами з плаваючою комою для трансп'ютера був специфікований на *Z*, при цьому були виявлені помилки в реалізації.

Tektronix використала метод *Z* для специфікації функціональності осцилографу.

Dansk Datamatik Center вже багато років розробляє індустріальні компілятори, використовуючи формальні методи.

Прикладами використання формальних методів є Європейський стандарт для програмної інженерії Portable Common Tools Environment та специфікації програмно-інженерних середовищ на основі баз даних.

Компанія Rolls-Royce, використовувала формальні методи для специфікації програми керування ядерним реактором.

Є ряд стандартів та класів систем, де застосування формальних методів для розробки регламентується та вимагається. Зокрема:

Міжнародна електротехнічна комісія (International Electrotechnical Commission, IEC – міжнародна організація з стандартизації в галузі електричних, електронних та суміжних технологій) вказує ряд формальних методів (CCS, CSP, HOL, LOTOS, OBJ, VDM, *Z*, Temporal Logic) для розробки критичних до безпеки систем.

Європейська космічна агенція рекомендує використання VDM або *Z*, доповнених описами на природних мовах, для специфікації вимог до критичних

систем, а також доведення коректності та застосування формальних виведень перед тестуванням.

Міністерство оборони Великобританії поширює використання формальних методів у своїх стандартах та вимагає: використання формальної нотації в специфікації критичних до безпеки компонент, аналіз таких компонент з точки зору несуперечливості та повноти, все критичне до безпеки програмне забезпечення має бути верифіковане та валідоване (що включає формальне доведення та строге, але неформальне, доведення коректності, статичний та динамічний аналіз), програмні та електронні компоненти обладнання для оборонних цілей мають бути класифіковані відповідно до аналізу ризиків.

Наглядова рада з атомної енергетики Канади також вимагає застосування формальних методів для розробки програмного забезпечення безпечних систем атомних електростанцій.

1.4. Використання специфікацій програм на різних етапах життєвого циклу програми

Досвід показує, що використання математичних методів для специфікацій та моделювання програмного забезпечення може сприяти розв'язанню проблеми побудови надійного програмного забезпечення. Математичні методи, які-б ефективно використовувалися у виробництві програмного забезпечення мають задовольняти деяким основним вимогам. По-перше нотації (записи), які застосовуватимуться в специфікаціях мають бути стандартизовані, а отже виробництво багатьох проектів може провадитись групами розробників без непорозумінь. Це дає можливість використовувати вже розроблені специфікації. По-друге, програмні засоби потребують автоматизації маніпулювання з формалізованим текстом. Обидві ці передумови вимагають від таких нотацій стабільності та ясності.

Традиційно виділяють наступні фази життєвого циклу програми:

- аналіз вимог;
- специфікація;
- проектування;
- реалізація;
- тестування та відлагодження;
- експлуатація та супроводження.

Специфікації можуть використовуватися на різних фазах життєвого циклу ПЗ, зокрема:

- при аналізі вимог специфікації можуть використовуватися для уточнення вимог, узгоджень їх із замовником, для побудови прототипів;

- при проектуванні – для контролю правильності проекту;
- при реалізації – для формулювання завдань розробникам та створення документації;
- при тестуванні – для перевірки виконання вимог;
- при супроводженні – для уточнення змін, підтримки узгодженості документації з системою, та інше.

Однією з важливих переваг використання специфікацій є збільшення глибини розуміння системи яка уточнюється. В процесі створення специфікації розробники мають більше можливостей для виявлення недоліків, непослідовностей, неоднозначностей та неповноти проекту. Специфікація є корисним засобом зв'язку між замовником та проектувальником, між проектувальником та розробником, а також між розробником та тестувальником. Вона часто виступає як супутня документація до програмного коду системи, але має високий рівень опису. Однією з важливих теоретичних причин використання формальних методів¹ (ФМ) специфікацій програм, є те, що самі програми є математичними об'єктами, так як вони мають формальну семантику та виражені на формальній мові, а отже, можуть бути оброблені за допомогою математичних засобів. Таким чином, виникає можливість доведення певних властивостей програм, зокрема властивості правильності, за допомогою математичних методів.

Останнім часом серед практичних методів проектування програмних систем (ПС) найбільшого розповсюдження набуло об'єктно-орієнтоване проектування (ООП). Визначилась технологічна база розробки різних видів програмного забезпечення методами ООП, зокрема, мова UML (Unified Modeling Language) [2] та її інструментальна підтримка (Rational Rose, Ms Visio, Rational Software та інші). ООП стало базисом генеруючого (породжуючого) [3] програмування, яке включає в себе і інші методи програмування. Зазначені методи підтримують розробку ПС на усіх етапах життєвого циклу.

Результатом є напівформальна специфікація, тобто специфікація, яка включає формальні, неофіційні та графічні елементи, але не володіє формальною семантикою. Разом з тим напівформальна специфікація, яка не має точної семантики, може привести до неоднозначного тлумачення; тому напівформальні специфікації не можуть використовуватися для формальних міркувань.

¹ Тут і далі під формальними методами розуміється «розділ інформатики, пов'язаний з застосуванням математичних методів до виробництва програмного забезпечення.» [4]. Формальні методи базуються на математичній логіці. Ці методи володіють ефективними механізмами для моделювання, синтезу та аналізу.

Відмітимо, що UML та подібні їй мови специфікацій (наприклад, MSC, SDL) безумовно є достатньо зручними засобами проектування, але, як правило, непридатні для доведення властивостей програм, зокрема, правильності, на що робиться акцент в математичних (формальних) методах, базованих на апараті математичної логіки. Деякі дослідження присвячені формалізації напівформальних методів, зважаючи на що, далі ці засоби будуть розглядатися поряд із формальними методами.

Однією з важливих теоретичних причин використання математичних (формальних) методів специфікацій програм є те, що самі програми є математичними об'єктами, так як вони мають формальну семантику та виражені на формальній мові, а отже можуть бути оброблені за допомогою математичних засобів. Таким чином, виникає можливість доведення певних властивостей програм, зокрема, властивості правильності за допомогою математичних методів.

1.5. Класифікація методів та мов специфікацій програм

В основу класифікації мов та методів специфікацій програм можна покласти різні критерії: призначення, вид, повнота та форма представлення тощо. Таким чином отримуємо різні класифікації мов та методів специфікацій предметних областей (програмних систем).

У відповідності до повноти опису специфікації бувають **виконуваними** (Executable). Такі специфікації визначають поведінку системи достатньою мірою для забезпечення можливості їх програмної інтерпретації (анімації). Приклади мов специфікацій, які дозволяють задавати виконувані специфікації: Z [7, 10, 11, 12], RSL [7, 13, 14, 15, 16], VDM [7, 8, 9], B [7].

У відповідності до об'єкту специфікації виділяють наступні типи специфікацій:

- **поведінкові** (behavioral) специфікації – описують обмеження на поведінку об'єкту специфікації, а саме на функціональні можливості, безпеку та виконання;
- **структурні** (structural) специфікації – описують обмеження на внутрішній склад об'єкту специфікації, а саме на використання та склад, відношення залежності;
- специфікації **взаємодії** (interaction) – описують обмеження на взаємодію між двома чи більшою кількістю об'єктів специфікації, а саме на відповідність інтерфейсів та протоколів.

Методи специфікацій традиційно класифікують за підходом до представлення моделі. Зокрема, виділяють наступні.

- **Моделе-орієнтовані** підходи до специфікації. Метою таких специфікацій є побудова абстрактної моделі специфікованої ПС. Такі методи специфікацій базуються на описі станів (state-based). Приклади: VDM [7, 8, 9], Z [7, 10, 11, 12], RAISE (RSL) [7, 13, 14, 15, 16], B [7].
- Методи, **орієнтовані на властивості** (за іншою термінологією **алгебричні** методи). Метою таких специфікацій є опис системи в термінах бажаних властивостей без побудови явної моделі. Такі методи специфікацій базуються на описі дій (action-based). Приклади: OBJ [6], LARCH [18, 19, 20], Anna [21], Clear [17], StateCharts [22], CSP [4], CCS [5].

Відмінність між моделе-орієнтованими та алгебричними методами не настільки чіткі, як може спочатку здатися. На практиці моделе-орієнтовані специфікації часто описують аспекти специфікуємої системи за допомогою аксіом, що властиві специфікаціям, орієнтованим на властивості. А алгебричні специфікації часто описують набори базових типів даних при моделюванні властивостей, але при цьому використовують побудову моделі специфікуємої системи. Методи, які є одночасно моделе-орієнтованими та орієнтованими на властивості, називаються гібридними. Це, зокрема, VDM [7, 8, 9], Anna [21], Z [7, 10, 11, 12], RSL [7, 13, 14, 15, 16], UML [2].

При класифікації мов специфікацій за призначенням виділяють **універсальні** мови специфікацій, які застосовуються для опису широкого класу задач (наприклад, Z [7, 10, 11, 12], B [7], VDM [7, 8, 9], Clear [17], RSL [7, 13, 14, 15, 16]) та **спеціалізовані** мови специфікацій, що розроблені для певних предметних областей та є практично незастосовними для інших предметних областей. Так, CCS [5] та CSP [4] є спеціалізованими методами формальних специфікацій, що призначені для опису взаємодії між паралельно працюючими процесами.

Можна класифікувати мови за видом представлення специфікації. З цієї точки зору виділяються **візуальні** (графічні) та **текстові** специфікації. У візуальних специфікаціях поведінка та структура системи представляється графічним способом (у вигляді графів). Такими є UML/OCL [2], StateCharts [22].

Крім того, мови специфікацій класифікують у відповідності до типу представлення специфікації. Мови специфікацій розподіляються на наступні базові типи:

- **транзиційні** специфікації (специфікації переходів станів);
- **темпоральні** (часові) логічні специфікації;
- **паралельні** специфікації;

- специфікації **абстрактної моделі**;
- **алгебричні** специфікації;
- **аксіоматичні** специфікації.

Одна мова специфікацій може підтримувати кілька типів.

1.5.1. Транзиційні специфікації

Специфікації цього типу описують поведінку системи набором станів та визначають операції як переходи між станами; поведінка може також задаватися відношеннями між станами.

Специфікація включає стани та переходи.

Особливостями таких специфікацій є загальний простір станів, текстові та графічні нотатки, модульність, адекватне застосування до систем керування.

Специфікації станів та переходів між ними підтримують такі мови: StateCharts [22], CSP [4], CCS [5], UML/OCL (діаграми скінченних автоматів (станів), діяльності) [2].

1.5.2. Темпоральні (часові) та паралельні специфікації

Вказані специфікації явно визначають поведінку у відповідності з описами системних станів і наборів подій в термінах порядку виконання та синхронізації. Паралельні специфікації визначають дії в термінах паралельних подій.

Специфікація включає стани, переходи, події, впорядкування та синхронізацію.

Особливостями таких специфікацій є потужний механізм специфікації, рівень синхронізації специфікації.

Часові логічні специфікації підтримують такі мови: Temporal Logic Specification (TLS) [23, 24, 25], Petri nets [26], StateCharts [22], GIL [27], CSP [4], UML/OCL (діаграми скінченних автоматів (станів), діяльності, комунікації, взаємодії, послідовності, синхронізації) [2].

1.5.3. Специфікації абстрактної моделі

Такі специфікації явно описують поведінку в термінах моделі. Специфікація використовує чітко визначені типи (множини, послідовності, відношення, функції) та визначає операції на моделях.

Специфікація включає модель типу, інваріант, перед- та пост- умови.

Особливостями таких специфікацій є явне моделювання станів в моделях, можливості виконання та побудови об'єктів в ієрархічному порядку.

Специфікації абстрактної моделі підтримуються такими мовами: VDM [7, 8, 9], Z [7, 10, 11, 12], RAISE (RSL) [7, 13, 14, 15, 16], UML/OCL [2].

1.5.4. Алгебричні специфікації

Ці специфікації визначають поведінку сукупністю відношень еквівалентності, що описують властивості об'єктів та операції над ними.

Специфікація включає функції та відношення.

Особливостями таких специфікацій є наявність описів функцій, підтримка абстракції даних; ці специфікації особливо застосовні для абстрактних типів даних.

Алгебричні специфікації підтримують такі мови: OBJ [6], Larch [18, 19, 20], Anna [21], Clear [17].

1.5.5. Аксиоматичні специфікації

Ці специфікації визначають поведінку за допомогою логічних формул, а також задають вхідні, проміжні та вихідні твердження.

Специфікація включає операції зв'язку між вхідними та вихідними параметрами, аксіоми з перед- та пост- умовами.

Особливостями аксіоматичних специфікацій є:

- простота розуміння;
- широкі можливості застосування;
- можливість розширення системи;
- використання методів доведення.

Аксиоматичні специфікації підтримують такі мови: VDM [7, 8, 9], Anna [21] та Z [7, 10, 11, 12].

1.6. Рівні застосування формальних методів (ФМ)

Навіть найбільші прихильники формальних методів (ФМ) змушені визнати, що існують області, у яких ФМ виявляються гіршими за більш загальноприйняті методи. Так, вважається, що вони погано підходять для проектування інтерфейсу користувача, при якому важливу роль грають неформальні розуміння; утім, у декількох програмах, що мали визначений успіх, формальна специфікація інтерфейсу все-таки була проведена [19].

Застосовувати ФМ до всіх компонентів системи є надлишковим та дорогим. Навіть у системі CICS – спільному проекті обчислювальної лабораторії Оксфордського університету і корпорації IBM, відзначеному Королівської премії за досягнення в техніку, – за допомогою ФМ розроблено лише біля десятої частини. Ця десята частина вилилася в сотні тисяч рядків коду і тисячі сторінок специфікацій, дозволила знизити витрати на 9% у порівнянні з розробкою

традиційними методами (підтверджено незалежною перевіркою) і часто згадується в якості одного із самих значних прикладів застосування ФМ.

Визначивши, що ФМ вам дійсно потрібні, вибравши придатну нотацію і зрозумівши, які компоненти системи виграють від формального підходу, необхідно подумати про рівень, до якого будуть застосовуватися ФМ. Будемо вважати, що таких рівнів три:

- формальна специфікація;
- формальна розробка та перевірка;
- доведення.

1.6.1. Формальна специфікація

Використання формальної специфікації вигідно в багатьох випадках. Формальна мова допомагає зробити специфікації більш повними й однозначними, полегшує їхнє обмірковування, навіть на неформальному рівні. Він дозволяє краще орієнтуватися в рівнях абстрагування і відкладати складності до більш придатного моменту.

ФМ особливо важливі для проникнення в суть проекрованої системи, дозволу неоднозначностей і структурування як підходу до проблеми, так і реалізації, що виходить у результаті.

Ця техніка дуже добре зарекомендувала себе при розробці програмної архітектури для сімейства осциллографів і в таких різноманітних видах діяльності, як формальний опис алгоритму для системи голосування, опис структури документів і виявлення внутрішніх протиріч у побудові WWW.

1.6.2. Формальна розробка і перевірка

Повна формальна розробка в даний час проводиться досить рідко. Вона передбачає формальну специфікацію системи доведення, що система має необхідні властивості і не має небажаними, і нарешті, застосування уточнюючих розрахунків, що перетворюють абстрактну специфікацію в усі більш і більш конкретні представлення, останнім з яких є код, що виконується.

У цьому випадку допускаються як формальні, так і неформальні, але строгі доведення.

1.6.3. Доведення

З появою інструментів підтримки, зокрема, програм для доведення теорем і перевірки доведень, стала можлива механічна перевірка несуперечності й обґрунтованості доведень.

Для деяких класів систем машинна перевірка доведень дійсно дуже важлива. Природно включити сюди системи з підвищеними вимогами до безпеки і захисту. Фактично за таку перевірку ратують у своїх стандартах багато організацій. Так, Європейське космічне агентство наполягає на застосуванні формальних виведень (перед тестуванням) скрізь, де це можливо, і рекомендує незалежну перевірку доведень як спосіб зменшити імовірність помилки людини.

Деякі ФМ включають системи виведення теорем у якості однієї з компонентів. Це в першу чергу HOL, Larch (з компонентом LP - Larch Prover), OBJ і PVS. Крім того, існують системи доведення і середовища підтримки, що включають подібні системи, для таких методів як B (B toolkit фірми B-Core), CSP (FDR фірми Formal Systems (Europe) Ltd.), RAISE (CRI), VDM (VDM Toolbox фірми IFAD) і Z (Balzac/Zola фірми Imperial Software Technology, ProofPower фірми ICL).

Цікава ідея розробки систем доведення для використання з визначеним методом. Так, програми доведення теорем для Z були створені в середовищах EVES, HOL і OBJ.

Кожен із трьох перерахованих рівнів корисний сам по собі. Але перед тим, як стати на шлях цілком формальної розробки з машинною перевіркою доведень, варто оцінити, чи окупляться додаткові витрати часу, сил, людської праці, грошей на інструментальні програми і т.д. Для систем, що вимагають найвищої надійності – таких, де помилка загрожує втратою людських життів, значним фінансовим чи матеріальним збитком – вони окупаються і, більш того, необхідні.

Контрольні запитання та вправи

1. Що таке формальна специфікація?
2. Що таке метод формальної розробки ПС (формальний метод)?
3. Наведіть визначення формальної нотації (формальної мови, мови специфікації).
4. Появніть зв'язок та відмінності між мовами та методами формальної специфікації.
5. Які підходи до методів формальної розробки Ви знаєте?
6. На яких етапах життєвого циклу програми та як саме можуть використовуватися специфікації програм?
7. Як поділяють специфікації у відповідності до повноти опису?
8. Які специфікації виділяють у відповідності до об'єкту специфікації?
9. Які методи специфікацій можна виділити у відповідності до відходу представлення моделі? Їх відмінності та спільні риси.

10. Які мови специфікацій можна виділити у відповідності до призначення, а які за видом?
11. Як класифікують мови специфікацій програм у відповідності до типу представлення специфікації?
12. Які основні властивості транзиційних специфікацій Ви знаєте та які транзиційні специфікації Ви можете назвати?
13. Які основні властивості темпоральних (часових) специфікацій Ви знаєте та які темпоральні (часові) специфікації Ви можете назвати?
14. Які основні властивості специфікацій абстрактної моделі Ви знаєте та які специфікації абстрактної моделі Ви можете назвати?
15. Які основні властивості алгебричних специфікацій Ви знаєте та які алгебричні специфікації Ви можете назвати?
16. Які основні властивості аксіоматичних специфікацій Ви знаєте та які аксіоматичні специфікації Ви можете назвати?
17. Які рівні застосування формальних методів Ви знаєте?

Частина 2

Основні поняття мови Z

2.1. Типи даних мови Z

2.1.1. Об'єкти та типи

Тип – це вираз обмеження: він є назвою набору, або складним типом, що створений з більш простих типів за допомогою конструкторів типів.

Кожен вираз, який з'являється в Z специфікації пов'язане з унікальним типом та, якщо вираз визначений, то значення виразу є членом його типу. Кожна змінна має тип, який може бути виведений з її декларації, існують правила для отримання типу складного виразу будь-якого вигляду від типів його підвиразів.

Типи важливі, так як можливо автоматично обчислити типи всіх виразів в специфікації та перевірити, що вони мають зміст.

Наприклад у виразі: $(0, 1) = \{1, 2, 3\}$, ліва сторона – задана пара, але права – множина, таким чином (згідно з Z) вираз не має змісту. Помилки такого вигляду можуть бути знайдені в ході перевірки типів. Звичайно, немає ніякої гарантії, що специфікація, вільна від помилок типу може бути здійснена, і ще менше гарантії, що вона відповідає вимогам замовника. Можливість автоматичної перевірки типів – сильна прагматична причина для наявності типів в Z, існують для цього також і теоретичні причини, пов'язані з тим, що кожен вираз в специфікації виражається як множина, та уникнення парадоксів теорії множин Рассела та інших.

Кожна Z специфікація починається з деяких об'єктів, що відіграють певну роль в специфікації, але не мають цікавої внутрішньої структури. Ці атомні об'єкти є членами основних типів, або задають множини специфікації. В багатьох специфікаціях атомними об'єктами є цілі числа, і вони є базовим типом Z, але існують і інші базові типи, наприклад специфікація ЗАПИСНА КНИЖКА, може мати як атомний об'єкт ім'я, який належить до базового типу FNAME.

Стартуючи з атомних об'єктів, більш складні об'єкти можуть бути отримані різними способами. Ці складні об'єкти – змінні складних типів, утворених за допомогою конструкторів типів Z. Є три види складних типів: типи множин (set types), декартові добутки (Cartesian product types) та типи схеми (schema types). Для отримання складнішої структури конструктори типу можуть застосовуватись неодноразово.

2.1.1.1. Множини та множина типів

Множиною називається невпорядкований набір різних значень одного і того-ж типу. Наприклад: $\{1,3,5\}$, $\{\text{"Mary"},\text{"John"},\text{"Peter"}\}$.

При цьому, як видно з визначення множини, записи $\{1,3,5\}$, $\{5,3,1\}$ та $\{3,5,1,1\}$ визначають одну і ту ж множину.

Множина в Z може бути описана, як список елементів, наприклад: $\{1,3,5\}$. Описаний тип $\mathbf{P Z}$ включає перші 5 натуральних числа, що кратні 2.

Множину можна також задати, задаючи властивість елементів множини, наприклад: множина $\{p : \mathbf{PERSON} \mid \text{age}(p) > 16\}$ має тип $\mathbf{P PERSON}$.

Множини елементів одного типу рівні, якщо вони містять однакові дані.

2.1.1.2. Кортежі та декартові добутки

Декартовим добутком називається впорядкований скінчений набір значень, можливо, різних типів, наприклад: $(1,2)$, $(1,\text{true},\text{"John"})$.

В декартовому добутку суттєвим є порядок слідування елементів, тобто $(1,2)$ та $(2,1)$ є різними значеннями.

Кортежі (a, b, c) , $(a, (b, c))$ та $((a, b), c)$ мають різні типи.

Більш загально, якщо $x_1, \dots, x_n - n$, де $n \geq 2$ об'єктів типів t_1, \dots, t_n відповідно, то n -кортеж (x_1, \dots, x_n) – об'єкт типу $t_1 \times \dots \times t_n$. Якщо (y_1, \dots, y_n) інший об'єкт того-ж типу $t_1 \times \dots \times t_n$, то $(x_1, \dots, x_n) = (y_1, \dots, y_n)$, тоді і тільки тоді, коли $x_i = y_i$ для кожного $1 \leq i \leq n$.

В Z специфікаціях не існує кортежів, без компонент, або з однією компонентою.

2.1.1.3. Відношення та функції

Поки що представлені два види об'єктів – множини та кортежі – єдині, які є фундаментальними для Z . Інші математичні об'єкти можуть бути промодельовані за допомогою комбінації цих трьох основних конструкцій.

Серед найважливіших математичних об'єктів – бінарні відношення та функції, обидва вони промодельовані в Z їх графами. Граф бінарних відношень – набір впорядкованих пар. Наприклад, граф відношення \leq на цілих числах містить пари $(0, 1)$, $(0, 2)$, $(1, 2)$, $(-37, 42)$, і так далі, але не містить $(3, 3)$ чи $(45, 34)$. Ототожнення між бінарним відношенням та його графом в Z настільки сильна, що можна про них говорити як про один об'єкт. Запис $X \leftrightarrow Y$, означає множину бінарних відношень між множинами X та Y , визначене далі, це є синонімом множини $\mathbf{P}(X \times Y)$ підмножин множини $X \times Y$ впорядкованих пар.

Математичні функції – спеціальний вид відношення, що пов'язують кожен об'єкт зліва щонайбільше з одним об'єктом вправа. Далі буде описано запис $X \mapsto Y$, що є синонімом множини відношень з такою властивістю. Вони

називаються частковими функціями. Множина $X \rightarrow Y$, містить всі тотальні функції, тобто відношення, що пов'язують кожен об'єкт зліва рівно з одним об'єктом з права. Запис $f(x)$ може використовуватись, якщо f – функція: значенням цього виразу є унікальний елемент з множини Y , з яким пов'язаний елемент x відношенням f . Функції з кількома аргументами можуть бути промодельовані вважаючи множину зліва від стрілки декартовим добутком. \mathbf{Z} розглядає функцію як статичне відношення між аргументами та результатами. Дві функції рівні, якщо вони містять однакові впорядковані пари.

2.1.2. Властивості та схеми

Сигнатура – набір змінних з типами. Для прикладу декларація $x, y : \mathbf{Z}$ створює сигнатуру з двома змінними x та y обидві типу \mathbf{Z} . В цій сигнатурі, предикат $x < y$ виражає властивість, що значення x менше значення y .

Кожна сигнатура природнім чином пов'язана з схемою типу. Наприклад, сигнатура, створена декларацією змінних $x, y : \mathbf{Z}$ асоціюється з схемою типу $\langle |x, y : \mathbf{Z} | \rangle$. Значення в цьому типі – зв'язування, в яких змінні приймають різні значення свого типу. Наприклад, властивість, виражена предикатом $x < y$ істинна при зв'язуванні типу $\langle x \Rightarrow 3, y \Rightarrow 5 \rangle$, та фальш при $\langle x \Rightarrow 5, y \Rightarrow 3 \rangle$. Предикат $y > x$ виражає ту ж саму властивість.

Предикат істинний при зв'язуванні, якщо властивість, яку він виражає істинна при цьому зв'язуванні. Т.ч. зв'язування задовольняє властивості, або предикату, який виражає цю властивість, якщо властивість істинна при цьому зв'язуванні. Було показано, що існує більше ніж один шлях вираження властивості через предикат, тоді говорять, що предикати логічно еквівалентні.

Схема A з сигнатурою та властивістю в нашому прикладі може виглядати наступним чином:

A:
$x, y : \mathbf{Z}$
$x < y$

x, y при цьому називаються компонентами (змінними) A .

2.1.2.1. Об'єднання властивостей

Найпростіші предикати істинні, якщо властивість, яку вони виражають, істинна при всіх зв'язуваннях, та фальш, якщо властивість, яку вони виражають не є істинною ні при якому зв'язуванні.

Рівняння $E_1 = E_2$ виражає властивість рівності значень E_1 та E_2 . Предикат $E_1 \in E_2$ виражає властивість, що E_1 належить множині E_2 .

Базові предикати можуть бути об'єднані різними способами. Наприклад, предикат $P_1 \vee P_2$, $P_1 \wedge P_2$, виражають властивості, що хоча б один з предикатів P_1 або P_2 істинний, та обидва предикати P_1 та P_2 істинні. Також для побудови складних предикатів можна використовувати і інші зв'язки логічного числення такі як \neg , \Rightarrow та \Leftrightarrow .

Якщо x – натуральне число виразимо предикат з квантором узагальнення наступним чином: $\forall z : N \bullet x < z$. Аналогічно можна використовувати квантор існування (\exists).

Більш загальна форма використання квантора узагальнення для побудови предикатів: $\forall D \mid P \bullet Q$, де D – опис змінних, а P та Q – предикати.

2.1.2.2. Оформлення та перейменування

Якщо S – схема, тоді S' та ж сама схема S за виключенням того, що всі імена, що входять до її складу були замінені на ті ж імена з суфіксом $'$. Тобто сигнатура S' містить компоненту x' для кожної компоненти x з S , при чому їх типи співпадають.

Нехай A – схема, визначена вище, тоді зв'язування для неї мають тип $\langle \langle x, y : Z \rangle \rangle$, та зв'язування для A' мають тип $\langle \langle x', y' : Z \rangle \rangle$. Зв'язування $z = \langle x' \Rightarrow 3, y' \Rightarrow 5 \rangle$, для A' отримуємо зв'язування $z_0 = \langle x \Rightarrow 3, y \Rightarrow 5 \rangle$, для A . При чому, якщо A' на z приймає значення істина, то A на z_0 теж істина.

Є три стандартних оформлення, що використовуються в описі даних: $'$ для позначення заключного стану, $?$ для позначення вхідних змінних та $!$ для позначення вихідних.

Інша операція на схемах перейменування. Якщо S схема, то $S [y_1 / x_1, \dots, y_n / x_n]$ схема, отримана заміною кожної компоненти x_i відповідною y_i . Для того, щоб запис мав зміст ідентифікатори x_i повинні бути всі різні, при цьому компоненти y_i не повинні бути різними, чи відрізнятися від компонент схеми S . При цьому потрібно враховувати узгодженість типів після перейменування (будь-які дві компоненти з однаковими іменами повинні мати один тип).

Наприклад схема $A [y / x]$ має єдину компоненту y , тому що компонента x перейменована в y та зливається з оригінальним y . Це можливо, бо y та x мають один тип. Тоді зв'язування $z = \langle y \Rightarrow 5 \rangle$ має коректний тип $\langle \langle y : Z \rangle \rangle$, але зв'язування $z_0 = \langle x \Rightarrow 5, y \Rightarrow 5 \rangle$, для A не задовольняє властивості схеми A , тому z не задовольняє властивості $A [y / x]$.

2.1.2.3. Об'єднання схем

Дві сигнатури вважають *сумісними за типами*, якщо кожна змінна, яка присутня в обох, має однаковий тип в обох сигнатурах.

Наприклад, наступні дві сигнатури є *сумісними за типом*, так як їх єдина спільна змінна b має однаковий тип.

$$a : \mathbf{P} X; b : X \times Y$$

та

$$b : X \times Y; c : Z$$

Ці дві сигнатури можуть бути об'єднані, та утворити сигнатуру

$$a : \mathbf{P} X; b : X \times Y; c : Z.$$

Нова сигнатура містить змінні кожної з попередніх сигнатур з тими-ж типами, тому говорять, що попередні сигнатури є *підсигнатурами*.

Якщо одна сигнатура є підсигнатурою іншої, то зв'язування z_1 для першої може бути отримане із будь-якого зв'язування z для другої, ігноруванням додаткових змінних. В такому випадку зв'язування z_1 називають *обмеженням* зв'язування z , а зв'язування z називають *розширенням (продовженням)* зв'язування z_1 до більшої сигнатури.

Для сумісності за типами сигнатури мають мати однакові типи для спільних змінних, але це не означає, що ці змінні повинні бути оголошені однаково, оскільки декларація може нести більше інформації, ніж тип змінної. Для прикладу, два бінарних відношення між множинами X та Y мають однаковий тип $\mathbf{P} (X \times Y)$, тому дві сигнатури тип-сумісні, навіть, якщо в одній сигнатурі декларується $f : X \leftrightarrow Y$, а в іншій $f : X \rightarrow Y$.

Дві схеми S та T з тип-сумісними сигнатурами можуть бути об'єднані *оператором об'єднання схем*, та утворити нову схему $S \wedge T$. При умові, що жодна компонента в S не має такого-ж імені як глобальна змінна в T , та навпаки. Сигнатурою цієї нової схеми є об'єднання сигнатур схем схеми S та T , а її властивість об'єднання властивостей схеми S та T : істинна, якщо при будь-якому зв'язуванні z , обмеження z до сигнатур S задовольняє властивості S , та обмеження z до сигнатур T задовольняє властивості T .

Приклад. Нехай маємо описану вище схему A :

$A:$
$x, y : \mathbf{Z}$
$x < y$

та схему

G:
$y : \mathbf{Z}$ $z : 1..10$
$y = z * z$

Тоді схема $A \wedge G$ матиме вигляд:

$A \wedge G$:
$x, y : \mathbf{Z}$ $z : 1..10$
$x < y \wedge y = z * z$

Іншими логічними зв'язками схем (при умові тип-сумісності) є \vee , \Rightarrow та \Leftrightarrow при цьому сигнатури об'єднуються, а до властивостей схем застосовується відповідна логічна зв'язка. Можна також застосовувати \neg до схеми, при цьому сигнатура зберігається, а властивість є запереченням відповідної властивості попередньої схеми. Так, запереченням схеми G буде наступна схема:

$\neg G$:
$y, z : \mathbf{Z}$
$z < 1 \vee z > 10 \vee y \neq z * z$

Це зроблено, завдяки тому, що частина декларації початкової схеми стала її властивістю, тобто маємо:

G:
$y, z : \mathbf{Z}$
$1 \leq z \leq 10 \wedge y = z * z$

Якщо D – декларація, P – предикат, а S - схема, тоді $\forall D \mid P \bullet S$ є схемою. Тоді схема S повинна містити як компоненти всі представлені змінні D і вони повинні мати ті-ж типи. Сигнатура результату містить всі компоненти S крім представлених в D , та вони мають той-же тип як в S . Властивість результату, отримана наступним чином: для будь-якого зв'язування z з сигнатури результату розглядаються всі розширення z_1 до сигнатури S . Якщо кожне таке розширення z_1 , що задовольняє і обмеження D і предикат P також задовольняє і властивості S , тоді зв'язування задовольняє властивість $\forall D \mid P \bullet S$.

Схема $\exists D \mid P \bullet S$ має ту ж сигнатуру, що і $\forall D \mid P \bullet S$, але її властивість істинна при зв'язуванні z , якщо принаймні одне з розширень z задовольняє обмеження D , предикат P та властивість S .

Наприклад, вираз $\forall z : \mathbf{Z} \mid z > 5 \bullet G$ може бути записане наступним чином:

$y : \mathbf{Z}$
$\forall z : \mathbf{Z} \mid z > 5 \bullet z \in 1..10 \wedge y = z * z$

2.1.3. Змінні та область їх дії

Специфікації можуть містити глобальні змінні, компоненти схем та локальні змінні. Область дії в \mathbf{Z} визначається декларацією, де ім'я відноситься до усього свого пункту.

Як і в багатьох мовах програмування (Pascal, Algol 60) та в багатьох формальних системах (λ -числення, логіки першого порядку) в \mathbf{Z} є система вкладених областей дії. Кожна змінна, представлена в декларації має область її дії в специфікації, що називається областю дії декларації, де назва змінної відноситься до цієї декларації.

З специфікації можуть також містити глобальні змінні, що оголошені поза будь-якою схемою, і ці змінні, можуть використовуватися у визначеннях схем. Математична бібліотека \mathbf{Z} оголошує багато таких змінних (+, <, >, ...) які є глобальними змінними від бібліотеки. Крім глобальних змінних, оголошених як частина математичних бібліотек, специфікація часто представляє і свої особисті глобальні змінні.

– Визначення глобальної множини імен

Приклад множина імен файлів інформація в файловій системі:

given FILENAME, BLOCK.

Звичайно ці імена вводяться в квадратних дужках: [FILENAME, BLOCK].

– Визначення глобальних змінних

Приклад

let

$f : \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$

|

$\forall n, m : \mathbf{N} \bullet f(n, m) = 10 * n + m$

end.

let

$encode : \mathbf{CHAR} \rightarrow \mathbf{N};$

$decore : \mathbf{N} \leftrightarrow \text{CHAR}$

|

$decode = encode^{-1}$

end.

В звичайному синтаксисі опис глобальних змінних має вигляд:

$f : \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$
$\forall n, m : \mathbf{N} \bullet f(n, m) = 10 * n + m.$

– Визначення схем

Кожне з цих введень є новою схемою, яка має певні параметри.

Приклад.

let POOL[RESOURCE] = **schema**

$inuse, avail : \mathbf{P} \text{ RESOURCE}$

|

$inuse \cap avail = \emptyset$

end.

let LOOKUP = (LOOKUP1 \wedge Ok) \vee ErrNotFound.

В звичайному синтаксисі описи глобальних змінних мають вигляд:

$inuse, avail : \mathbf{P} \text{ RESOURCE}$
$inuse \cap avail = \emptyset$

Визначення однієї схеми в термінах інших використовують символ \triangleq :

LOOKUP \triangleq (LOOKUP1 \wedge Ok) \vee ErrNotFound.

2. 2. Огляд синтаксису мови Z

2.2.1. Короткий огляд синтаксису мови Z

S, ..., S буде список з одного, чи більше випадків класу S, та S₁;...; S_n буде один, чи більше випадків S.

Фрази зазначені в квадратних дужках є необов'язковими.

Можна ігнорувати крапки з комою, що відділяють пункти.

Пункти відокремлені випадками класу Sep, який тут може бути крапка з комою, чи нова лінія (NL).

Specification ::= Paragraph NL... NL Paragraph

Paragraph ::= [Ident, ..., Ident]

| Axiomatic-Box

		Schema-Box
		Generic-Box
		Schema-Name [Gen-Formals] $\hat{=}$ Schema-Exp
		Def-Lhs = Expression
		Ident ::= Branch ... Branch
		Predicate
Axiomatic-Box	::=	$\left[\begin{array}{l} \text{Decl-Part} \\ \hline \text{Axiom-Part} \end{array} \right]$
Schema-Box	::=	$\frac{\text{Schema-Name [Gen-Formals]}}{\left[\begin{array}{l} \text{Decl-Part} \\ \hline \text{Axiom-Part} \end{array} \right]}$
Generic-Box	::=	$\frac{\text{Gen-Formals}}{\left[\begin{array}{l} \text{Decl-Part} \\ \hline \text{Axiom-Part} \end{array} \right]}$
Decl_Part	::=	Basic_Decl Sep ... Sep Basic_Decl
Axiom_Part	::=	Predicate Sep ... Sep Predicate
Sep	::=	; NL
Def-Lhs	::=	Var-Name [Gen-Formals] Pre-Gen Decoration Ident Ident In-Gen Decoration Ident
Branch	::=	Ident Var-Nema <<Expression>>
Schema-Exp	::=	\forall Schema-Text • Schema-Exp \exists Schema-Text • Schema-Exp \exists_1 Schema-Text • Schema-Exp Schema-Exp-1

Schema-Exp-1	::=	[Schema-Text] Schema-Ref \neg Schema-Exp-1 pre Schema-Exp-1 Schema-Exp-1 \vee Schema-Exp-1 Schema-Exp-1 \wedge Schema-Exp-1 Schema-Exp-1 \Rightarrow Schema-Exp-1 Schema-Exp-1 \Leftrightarrow Schema-Exp-1 Schema-Exp-1 $\{$ Schema-Exp-1 Schema-Exp-1 \setminus (Decl-Name, ..., Decl-Name) Schema-Exp-1 ; Schema-Exp-1 Schema-Exp-1 \gg Schema-Exp-1 (Schema-Exp)
Schema-Text	::=	Declaration / Predicate /
Schema-Ref	::=	Schema-Name Decoration / Gen-Actuals / / Renaming /
Renaming	::=	[Decl-Name/Decl-name, ..., Decl-Name/Decl-Name]
Declaration	::=	Basic-Decl; ...; Basic-Decl
Basic-Decl	::=	Decl-Name, ..., Decl-Name : Expression Schema-Ref
Predicate	::=	\forall Schema-Text • Predicate \exists Schema-Text • Predicate \exists_1 Schema-Text • Predicate let Let-Def; ...; Let-Def • Predicate Predicate-1
Predicate-1	::=	Expression Rel Expression Rel ... Rel Expression Pre-Rel Decoration Expression Schema-Ref pre Schema-Ref <i>true</i>

Rel	::=	= \in In-Rel Decoration
Let-Def	::=	Var-Name == Expression
Expression-0	::=	λ Schema-Text • Expression μ Schema-Text / • Expression / let Let-Def; ...; Let-Def • Expression Expression
Expression	::=	if Predicate then Expression else Expression Expression-1
Expression-1	::=	Expression-1 In-Gen Decoration Expression-1 Expression-2 \times Expression-2 \times ... \times Expression-2 Expression-2
Expression-2	::=	Expression-2 In-Fun Decoration Expression-2 P Expression-4 Pre-Gen Decoration Expression-4 - Decoration Expression-4 Expression-3
Expression-3	::=	Expression-3 Expression-4 Expression-4
Expression-4	::=	Var-Name / Gen-Actuals / Number Schema-Ref Set-Exp \langle / Expression, ..., Expression / \rangle [/ Expression, ..., Expression /] (Expression, Expression, ..., Expression) θ Schema-Name Decoration / Renaming / Expression-4 . Var-Name Expression-4 Post-Fun Decoration Expression-4 ^{Expression} (Expression-0)

Set-Exp	::=	{ [Expression, ..., Expression] }
		{ Schema-Text [• Expression] }
Ident	::=	Word Decoration
Decl-Name	::=	Ident Op-Name
Var-Name	::=	Ident (Op-Name)
Op-Name	::=	_ In-Sym Decoration _
		Pre-Sym Decoration _
		_ Post-Sym Decoration
		_ Decoration
In-Sym	::=	In-Fun In-Gen In-Rel
Pre-Sym	::=	Pre-Gen Pre-Rel
Post-Sym	::=	Post-Fun
Decoration	::=	[Stroke ... Stroke]
Gen-Formals	::=	[Ident, ..., Ident]
Gen-Actuals	::=	[Expression, ... Expression]

Нижче зазначено список термінальних символів, що використовуються в граматиці:

Word	назва, чи спеціальний символ;
Stroke	одиничний символ: ‘, ?, ! чи цифра нижнього індексу;
Schema-Name	те саме, що Word, але використовується як назва схеми;
In-Fun	інфіксний функціональний символ;
In-Rel	інфіксний символ відношення;
In-Gen	інфіксний видовий символ;
Pre-Rel	префіксний символ відношення;
Pre-Gen	префіксний видовий символ;
Post-Fun	префіксний функціональний символ;

2.2.2. Word та ідентифікатори

Word – найпростіший вид назви в Z специфікації: це непорожня послідовність великих та малих латинських літер, цифр, підкреслювань, чи спеціальних символів. Word використовуються як назви схем. Ідентифікатор (Ident) слово (word) що доповнюється описом:

Ident:: = Word Decoration

Якщо word використовується в специфікації як назва схеми, то називається *schema name*. Якщо A є назва схеми, то A' є назвою схеми, яка є копією схеми A, де всі імена компонент позначені символом '.

Деяким словам дають спеціальний статус символів операцій. Вони класифікуються як функціональні символи (In-Fun, Post-Fun), символи відношень (Pre-Rel або In-Rel) чи видові символи (Pre-Gen або In-Gen).

2.2.3. Символи операцій

Функціональні символи бувають двох видів: інфіксні, які з'являються між двома аргументами та постфіксні, які слідують за єдиним аргументом.

Іноді, необхідно назвати символ без того, щоб застосовувати його до аргументів, це може бути зроблено заміною аргументів символом $_$, наприклад $_+$, $_ \rightarrow _$. Для запобігання непорозумінь такі вирази в тексті завжди потрібно поміщати в круглі дужки. Задамо деякі скорочення:

$x+y$ скорочення для $(_+_)$ (x, y)

$x \geq y$ скорочення для $(_\geq_)$ (x, y)

$x \rightarrow y$ скорочення для $(_\rightarrow_)$ (x, y)

disjoint x скорочення для $x \in (\text{disjoint } _)$ (розбирати на частини)

F X скорочення для $(\mathbf{F} _)[X]$

R* скорочення для $(_*)\mathbf{R}$.

Назви також використовуються в деклараціях, наприклад

$_+_ : \mathbf{Z} \times \mathbf{Z} \rightarrow \mathbf{Z}$

$_\geq_ : \mathbf{Z} \leftrightarrow \mathbf{Z}$

є деклараціями + та \geq .

Деякі символи операцій, які не є стандартними задаватимуться по мірі необхідності.

Деякі інші символи операцій є стандартними:

Інфіксні функціональні символи (In-Fun)

Пріоритет 1: \mapsto

Пріоритет 2: ..

Пріоритет 3: +, -, ∪, \, ∩, ⊕, ⊖

Пріоритет 4: *, div, mod, ∩, 1, †, ;, °, ⊗

Пріоритет 5: ⊕, #

Пріоритет 6: ◁, ▷, ◀, ▶

Постфіксні функціональні символи (Post-Fun)

$_ \sim _ * _ +$

Інфіксні символи відношень (In-Rel)

$\neq \notin \subseteq \subset < \leq \geq >$ **prefix suffix in E** \sqsubseteq **partition**

Префіксні символи відношень (Pre-Rel)

disjoint

Інфіксні видові символи (In-Gen)

$\leftrightarrow \rightrightarrows \rightarrow \rightsquigarrow \rightharpoonup \dashrightarrow \twoheadrightarrow \Rightarrow \Longrightarrow$

Префіксні видові символи (Pre-Gen)

P₁ id F F₁ seq seq₁ iseq bag

2.2.4. Специфікації

Z специфікація складається поєднанням різних підходів: формального математичного тексту та неофіційного пояснення. Формальний текст складається з послідовності параграфів, що представляють схеми, глобальні змінні, базові типи специфікацій, кожен параграф базується на попередніх. Крім випадків визначень вільного типу, рекурсія не дозволена.

Кожен параграф може визначати один чи більше імен схем, основних типів, глобальних змінних та глобальних констант. Він може використовувати імена, визначені в попередніх параграфах. Ця поступова побудова словника називається *принципом визначення перед використанням*. Область дії кожного глобального імені від його визначення до кінця специфікації.

Існує кілька видів параграфів. Основні визначення типу, аксіоматичні описи, обмеження, визначення схеми, визначення скорочень, видова схема, постійні визначення та вільні визначення типу.

2.2.5. Визначення базових типів

Paragraph ::= [Ident, ..., Ident]

Визначення базових типів представляє один, чи кілька основних типів. Імена, що використовуються не повинні мати попередньої глобальної декларації. Область

їх дії простягається від визначення до кінця специфікації, їх імена стають частиною глобального словника основних типів.

Прикладом такого визначення типу є PERSON, ADDRESS з ADDRESS_BOOK (див далі): [PERSON, ADDRESS]

2.2.6. Аксиоматичні описи

$$\text{Paragraph} ::= \left[\begin{array}{l} \text{Declaration} \\ \hline \text{Predicate; } \dots \text{ ; Predicate} \end{array} \right]$$

Аксиоматичний опис представляє змінні та визначає їх значення. Ці змінні не повинні мати попередньої глобальної декларації. Область їх дії простягається від визначення до кінця специфікації, змінні стають частиною глобального сигнатури специфікації. Предикати зв'язують значення нових змінних між собою та до значень змінних, що були оголошені раніше.

Дужки [...] вказують на те, що роздільна лінія та список предикатів під нею є необов'язковими, якщо частина предиката відсутня, то вважаємо, що предикат всюди істинний.

Приклад:

$$\left[\begin{array}{l} \text{square} : N \rightarrow N \\ \hline \forall n \in N \bullet \text{square}(n) = n * n \end{array} \right]$$

2.2.7. Обмеження

Paragraph ::= Predicate

Предикат може з'явитися як параграф. Він визначає обмеження на значення глобальних змінних, що були оголошені попередньо. За цією це подібно до того, ніби обмеження було задане при оголошенні цих змінних, як частини їх аксиоматичного опису.

Прикладом обмеження є наступний предикат, що стверджує, що змінна n має значення більше 7:

$n > 7$.

2.2.8. Визначення схеми

$$\text{Paragraph} ::= \left[\begin{array}{l} \text{Schema-Name} \\ \text{Declaration} \\ \hline \text{Predicate; } \dots \text{ ; Predicate} \end{array} \right] \quad (1)$$

Paragraph ::= Schema-Name \cong Schema-Exp (2)

Ці форми представляють нову назву схеми. Слово, що поверх таблиці (в даному випадку Schema-Name), чи яке з'являється зліва від визначення надалі пов'язується з цією схемою. Ця назва не повинна з'являтися попередньо в специфікації і від місця оголошення до кінця специфікації є назвою схеми. Якщо частина предикату відсутня, то предикат вважається істинним.

Назвемо (1) вертикальною формою визначення схеми, а (2) горизонтальною. Тоді, якщо маємо наступну вертикальну форму:

S $D_1; \dots ; D_m$
$P_1; \dots ; P_n$

то еквівалентною їй горизонтальною формою є наступна форма:

$S \cong [D_1; \dots ; D_m | P_1; \dots ; P_n]$.

Задамо приклад визначення схеми, прийнятого для специфікації адресної книги, де стан системи, описаний в Z виглядає наступним чином:

$AddressBook$ $person: \mathbf{P}$ PERSON $address: PERSON \rightarrow ADDRESS$
$person = \mathbf{dom}$ address

Це визначення задане в вертикальній формі, воно може бути записане в горизонтальній наступним чином:

$AddressBook \cong$

$[person: \mathbf{P}$ PERSON; $address: PERSON \rightarrow ADDRESS$ |
 $person = \mathbf{dom}$ address].

Наступне визначення додавання нової адреси $AddAddress_1$ використовує більш складне вираження схеми:

$AddAddress_1 \cong (AddAddress \wedge OK) \vee Known.$

2.2.9. Скорочені визначення

Paragraph ::= Ident == Expression

Скорочені визначення представляє нову глобальну константу. Ідентифікатор (Ident) стає глобальною константою; його значення задається виразом справа (Expression), а його тип той-же, що і тип виразу. Область дії – від визначення до закінчення специфікації.

Приклад скороченого визначення є наступне визначення:

DATABASE == *ADDRESS* → *PERSON*.

2.3. Зсилки схеми

Коли схемі задана назва, як в підчастині “**Визначення схеми**”, воно може бути використовуватись для зсилки до схеми. Зсилки схеми може використовуватися як декларація, вираз чи предикат.

Schema-Ref ::= Schema-Name Decoration /Renaming/

Renaming ::= [Ident/Ident, ..., Ident/Ident]

Зсилка схеми складається з назви схеми, з описом (який може бути порожнім), та необов’язкового списку перейменувань. Будується копія названої схеми, яка була змінена, застосовуючи декорування до всіх компонент.

В списку перейменувань $[y_1 / x_1, \dots, y_n / x_n]$ всі x_i -ті повинні бути різні ідентифікатори, y_i не обов’язково повинні бути відмінними одна від одної, чи від компонент перейменовуваної схеми, але якщо дві компоненти схеми співпадають після перейменування, то їх типи повинні узгоджуватися (див. “**Оформлення та перейменування**”).

2.4. Декларація змінних

Змінні представлені та пов’язані з типами у відповідності до декларацій. Як пояснено в пункті “**Властивості та схеми**” декларація може також вимагати, щоб значення змінних задовольняло деяким властивостям, які називаються обмеженнями декларації. В **Z** існує два види декларацій:

Basic-Decl ::= Ident, ..., Ident : Expression
| Schema-Ref

Перший вид являє собою список змінних, що внесені до списку явно. Наприклад $x, y : 1 .. 10$.

Другий вид декларації – зсилка схеми; представляє компоненти схеми, як змінні, з тими-ж типами які вони мають в схемі, та обмеженнями значень, щоб задовольнити їх властивості. Наприклад, якщо **A** – декларація:

$x, y : \mathbf{Z}$
$x > y$

Тоді декларація представляє змінні x , y типу \mathbf{Z} з властивістю $x > y$.

Можлива також послідовність декларацій:

Declaration ::= Basic-Decl; ...; Basic-Decl.

2.5. Текст схеми

Текст схеми складається з декларації та необов'язкового списку предикатів.

Тексти схеми з'являються в вертикальній формі в аксіоматичних описах та визначеннях схеми, але вони мають також і горизонтальну форму:

Schema-Text ::= Declaration [| Predicate; ...; Predicate].

Ця форма використовується після кванторів \forall , \exists та \exists_1 та в виразах сформованих з використанням λ , μ та $\{\}$.

2.6. Вирази

Найпростішим видом виразів є ідентифікатори та натуральні числа. Круглі дужки можуть використовуватися для групування в виразах:

Expression ::= Ident
 | Number
 | (Expression)

Назва виразу:

(...) – кортеж

{...} – множина

Синтаксис:

Expression ::= (Expression, ..., Expression)
 | { [Expression, ..., Expression] }

Потрібно зауважити, що кортеж повинен містити принаймні два вирази.

Правила типу:

У виразі (E_1, \dots, E_n) , якщо аргументи E_i мають тип t_i , то вираз має тип $t_1 \times \dots \times t_n$.

У виразі $\{E_1, \dots, E_n\}$ кожен підвираз E_i має тип t . Тип всього виразу $\mathbf{P} t$.

Опис:

Вираз (x_1, \dots, x_n) означає n -ку з компонентами x_1, \dots, x_n .

Множина $\{x_1, \dots, x_n\}$ включає лише об'єкти x_1, \dots, x_n . Тут порядок несуттєвий. Якщо деякі елементи дублюються, то копія ігнорується.

Закони:

$(x_1, \dots, x_n) = (y_1, \dots, y_n) \Leftrightarrow x_1 = y_1 \wedge \dots \wedge x_n = y_n$

$$y \in \{x_1, \dots, x_n\} \Leftrightarrow y = x_1 \vee \dots \vee y = x_n$$

Назва виразу:

P – потужність множини

\times – кортеж

Синтаксис:

Expression ::= **P** Expression
 | Expression \times ... \times Expression

Правила типу:

У виразі **P** E, аргумент E повинен мати тип **P** t. Тип виразу тоді буде **P** (**P** t). Для прикладу, якщо E множина цілих чисел має тип **P** Z, тоді **P** E має тип **P** (**P** Z) – множина множин цілих чисел.

У виразі $E_1 \times \dots \times E_n$ кожен аргумент E_i має тип множини **P** t_i . Тип всього виразу тоді **P** ($t_1 \times \dots \times t_n$). Для прикладу, якщо E_1 має тип **P** Z, а E_2 має тип **P** CHAR, то $E_1 \times E_2$ має тип **P** (Z \times CHAR) – множина пар, цілих чисел та символів.

Опис:

Якщо S множина всіх підмножин S.

Якщо S_1, \dots, S_n множини, тоді $S_1 \times \dots \times S_n$ множина всіх n-ок вигляду (x_1, \dots, x_n) , де $x_i \in S_i$ для всіх $1 \leq i \leq n$.

Закони:

$$S_1 \times \dots \times S_n = \{x_1 : S_1; \dots; x_n : S_n \bullet (x_1, \dots, x_n)\}$$

Назва виразу:

{ | • } – множина включення

Синтаксис:

Expression ::= { Schema-Text [• Expression] }

Зауваження: якщо частина виразу відсутня, то опис в схемі тексту (див. вище).

Правила області дії:

У виразі { S • E } схема тексту S вносить локальні змінні, їх область дії вираз E.

Правила типу:

У виразі { S • E }, якщо тип підвиразу E є t, то типом виразу є **P** t.

Опис:

Елементами множини { S • E } є значення, взяті виразом E, коли змінні представлені S приймають всі можливі значення, які задовольняють властивості S.

Закони:

$$y \in \{ S \bullet E \} \Leftrightarrow \{ \exists S \bullet y = E \}$$

передбачається, що y не є змінною з S .

Назва виразу:

λ – λ - вираз

μ – μ - вираз

Синтаксис:

Expression ::= (λ Schema-Text \bullet Expression)

 ::= (μ Schema-Text [\bullet Expression])

У μ -виразі, якщо частина виразу відсутня, то описано в схемі тексту (див. вище).

Правила області дії:

У виразах ($\lambda S \bullet E$) та ($\mu S \bullet E$) схема тексту S представляє локальні змінні, їх область дії вираз E .

Правила типу:

У виразі ($\lambda S \bullet E$), якщо типом $E \in t$, та типом $S \in t'$, то тип усього виразу є $\mathbf{P}(t' \times t)$.

У виразі ($\mu S \bullet E$), якщо типом $E \in t$, тоді тип виразу є також t . Якщо вираз E відсутній, то тип виразу є типом характеристики кортежу декларації в S .

Опис:

Вираз ($\lambda S \bullet E$) означає функцію, яка бере аргументи з S , та як результат видає властивості E . Це еквівалентно, множині $\{ S \bullet (T, E) \}$, де T – характеристика кортежу S .

Вираз ($\mu S \bullet E$) визначений тальки, якщо існує унікальний шлях представлення значень змінних, представлених S , який задовольняє властивість S . Якщо це так, то значенням є значення E , при яких змінні, представлені S приймають ці значення.

λ - та μ - вирази повинні завжди міститися в круглих дужках.

Назва виразу:

let – локальне визначення

Синтаксис:

Expression ::= (**let** Let-Def; ...; Let-Def \bullet Expression)

Let-Def ::= Ident = Expression

Правила області дії:

У **let**-виразі (**let** $x_1 == E_1; \dots; x_n == E_n \bullet E$), змінні x_1, \dots, x_n локальні, їх область дії вираз E .

Правила типу:

У **let**-виразі ($\mathbf{let} \ x_1 == E_1; \dots; x_n == E_n \bullet E$), кожна локальна змінна x_i має той-же тип, що і відповідний вираз E_i . Тип цілого виразу є тип E .

Опис:

Значенням **let**-виразу є значення, яке прийняло його тіло, коли локальні змінні, приймають значення, задані правими сторонами визначень. Вираз ($\mathbf{let} \ x_1 == E_1; \dots; x_n == E_n \bullet E$) визначений, якщо всі праві сторони E_1, \dots, E_n визначені, та E також визначене в закріпленні де x_1, \dots, x_n приймають значення E_1, \dots, E_n відповідно. Значення виразу є значення E в закріпленні.

Для запобігання неоднозначностей **let**-вираз завжди записується в дужках.

Закони:

$(\mathbf{let} \ x_1 == E_1; \dots; x_n == E_n \bullet E)$

$= (\mu \ x_1 : t_1; \dots; x_n : t_n \mid x_1 = E_1; \dots; x_n = E_n \bullet E),$

якщо всі E_i -ті визначені, та немає колізій з іменами змінних.

Назва виразу:

Application – застосування функції

Синтаксис:

Expression ::= Expression Expression

Правила типу:

У виразі $E_1 E_2$, підвираз E_1 повинен мати тип $\mathbf{P} (t_1 \times t_2)$, та E_2 повинен мати тип t_1 , для деяких типів t_1 та t_2 . Типом всього виразу є t_2 .

Опис:

Вираз $f \ x$ означає застосування функції f до аргументу x . Строго кажучи, f не повинне бути функцією, але вираз $f(x)$ визначений, тоді і тільки тоді, коли існує унікальну значення y , таке, що $(x, y) \in f$, та значення виразу є значення y .

Закони:

$(\exists_1 \ y : Y \bullet (x, y) \in f) \Rightarrow$

$(x, f(x)) \in f \wedge$

$f(x) = (\mu \ y : Y \mid (x, y) \in f)$

Назва виразу:

\cdot – вибір

Синтаксис:

Expression ::= Expression . Ident

Правила типу:

У виразі $E.y$ підвираз E повинен мати схему типу у формі $\langle \mid x_1 : t_1; \dots; x_n : t_n \mid \rangle$, та ідентифікатор y має бути ідентичний з одним із імен компонент x_i для деякого $1 \leq i \leq n$. Типом виразу є відповідно тип t_i .

Опис:

Цей запис для вибору компоненти із зв'язування. Якщо $a \in$ зв'язування $\langle x_1 \Rightarrow v_1, \dots, x_n \Rightarrow v_n \rangle$ та $y \in$ ідентичним деякому x_i , тоді значенням $a . y \in v_i$.

Закони:

$$a \in S \Rightarrow a . y = (\lambda S \bullet y)(a)$$

Якщо a та b мають тип $\langle x_1 : t_1; \dots; x_n : t_n \rangle$, то
 $a = b \Leftrightarrow a . x_1 = b . x_1 \wedge \dots \wedge a . x_n = b . x_n$.

Назва виразу:

θ – обов'язкове форматування

Синтаксис:

Expression ::= θ Schema-Name Decoration [Renaming]

Renaming ::= [Ident / Ident, ..., Ident / Ident]

Правила типу:

У виразі $\theta S'$, в якому символ $'$ встановлений для (можливо порожнього) декорування, нехай компонентами $S \in x_1, \dots, x_n$. Змінні x'_1, \dots, x'_n повинні бути в межах: нехай їхні типи t_1, \dots, t_n . Типом виразу є схема типу $\langle x_1 : t_1; \dots; x_n : t_n \rangle$. Потрібно звернути увагу на те, що компоненти в цьому типі проіменовані без декорування ($'$): це означає, що $\theta S'$ має той-же тип, що і θS .

Якщо є перейменування $[q_1 / p_1, \dots, q_k / p_k]$, відбувається заміна кожного q_j на відповідний p_j . Тип виразу, сформованого з типів цих змінних, як і до цього, його компоненти не декоровані (символом $'$) та незміннені імена x_1, \dots, x_n .

Опис:

Значення виразу $\theta S'$, при зв'язуванні $u \in$ саме зв'язування z з схемою типу показаною вище; для всіх $i, 1 \leq i \leq n$, компонента $z.x_i \in$ значенням змінної x'_i у зв'язуванні u , тому $z \in$ зв'язуванням $\langle x_1 \Rightarrow u.x'_1, \dots, x_n \Rightarrow u.x'_n \rangle$.

Якщо нова схема T визначена наступним чином: $T \triangleq S'$, тоді θT буде мати тип $\langle x'_1 : t_1; \dots; x'_n : t_n \rangle$, де назви компонент декоровані. Це відрізняється від типу $\theta S'$.

Форма θ -виразу, що містить перейменування, дозволяється, головним чином, для надання характеристики кортежу для зсилок схеми, які використовують перейменування. В цій формі, значення компонент після декорування та перейменування, використовуються для формування значення виразу.

Закони:

$$(\theta S') . x_i = x'_i$$

$$\theta S' = \theta S \Leftrightarrow x'_1 = x_1 \wedge \dots \wedge x'_n = x_n$$

Назва виразу:

Schema-Ref

Синтаксис:

Expression ::= Schema-Ref

Тут частини схеми декорування та перейменування та зсилки порожні.

Правила типу:

Тип схеми зсилається на S використовує вираз $P \langle x_1 : t_1; \dots; x_n : t_n \rangle$, де S має компоненти x_1, \dots, x_n з типами t_1, \dots, t_n відповідно.

Опис:

Схема зсилки може використовуватися як вираз якщо: його значення є множина значення компонент в якій закріплені властивостями схеми. Зсилка схеми S еквівалентна множині $\{S \bullet \theta S\}$.

Закони:

$$S [E_1, \dots, E_n] = \{S [E_1, \dots, E_n] \bullet \theta S\}$$

Назва виразу:

If then else – умовний вираз

Синтаксис:

Expression ::= **if** Predicate **then** Expression **else** Expression

Правила типу:

У виразі **if P then E₁ else E₂**, типи виразів E₁ та E₂ повинні бути однакові. Їх спільний тип є типом усього виразу.

Опис:

Якщо предикат P істинний, то значенням виразу **if P then E₁ else E₂** є вираз E₁, інакше значенням виразу є E₂.

Закони:

$$P \Rightarrow \text{if } true \text{ then } E_1 \text{ else } E_2 = E_1$$

$$\neg P \Rightarrow \text{if } false \text{ then } E_1 \text{ else } E_2 = E_2$$

$$\text{if } P \text{ then } E \text{ else } E = E$$

Назва виразу:

Operators – правила для інфіксних та постфіксних функціональних символів

Синтаксис:

Expression ::= Expression **In-Fun** Expression
| Expression **Post-Fun**

Правила типу:

У виразі $E_1 \omega E_2$, де ω є інфіксним функціональним символом, тип ω має бути $\mathbf{P}((t_1 \times t_2) \times t_3)$, та типи підвиразів E_1 та E_2 повинні бути t_1 та t_2 відповідно, для деяких типів t_1 , t_2 та t_3 . Типом всього виразу є t_3 . х спільний тип є типом усього виразу.

У виразі $E \omega$, де ω є постфіксним функціональним символом, тип ω має бути $\mathbf{P}(t_1 \times t_2)$, та тип підвиразу E повинен бути t_1 , для деяких типів t_1 та t_2 . Типом всього виразу є t_2 .

Опис:

Вираз $E_1 \omega E_2$ є скороченням для виразу $(_ \omega _)(E_1, E_2)$, застосування ω до пари (E_1, E_2) . Відповідно правила застосування функції визначене, коли існує унікальне y таке, що $((E_1, E_2), y) \in (_ \omega _)$, та його значенням є y .

Вираз $E \omega$ є скороченням для виразу $(_ \omega)E$, застосування ω до E . Воно визначене, коли існує унікальне y таке, що $(E, y) \in (_ \omega)$, та його значенням є y .

Закони:

$$E_1 \omega E_2 = (_ \omega _)(E_1, E_2)$$

$$E \omega = (_ \omega)E$$

Назва виразу:

$\langle \dots \rangle$ – відображення послідовності

$[\dots]$ – відображення множини

Синтаксис:

$$\begin{aligned} \text{Expression} ::= & \langle / \text{Expression}, \dots, \text{Expression} / \rangle \\ & | [/ \text{Expression}, \dots, \text{Expression} /] \end{aligned}$$

Правила типу:

У виразі $\langle E_1, \dots, E_n \rangle$, всі підвирази E_i повинні мати однаковий тип t . Типом всього виразу є $\mathbf{P}(\mathbf{Z} \times t)$.

У виразі $[E_1, \dots, E_n]$, всі підвирази E_i повинні мати т однаковий тип t . Типом всього виразу є $\mathbf{P}(t \times \mathbf{Z})$.

Опис:

Вирази визначені, лише коли всі підвирази E_i визначені.

Закони:

$$\langle x_1, x_2, \dots, x_n \rangle = \{ 1 \mapsto x_1, 2 \mapsto x_2, \dots, n \mapsto x_n \}$$

$$[x_1, x_2, \dots, x_n] = \{ x_1 \mapsto k_1, x_2 \mapsto k_2, \dots, x_n \mapsto k_n \}$$

тут для кожного i , елементу x_i з'являються k_i для списку x_1, x_2, \dots, x_n .

2.7. Предикати

Назва:

= – еквівалентність

∈ – включення

Синтаксис:

Predicate ::= Expression = Expression

| Expression ∈ Expression

Правила типу:

У предикаті $E_1 = E_2$, вирази E_1 та E_2 повинні мати однаковий тип. У предикаті $E_1 \in E_2$, якщо E_1 має тип t , то E_2 повинен мати тип $\mathbf{P} t$.

Опис:

Предикат $x = y$ істинний, якщо x та y один і той-же об'єкт. Предикат $x \in y$ істинний, якщо об'єкт x належить множині y .

Закони:

$$x = x$$

$$x = y \Rightarrow y = x$$

$$x = y \wedge y = z \Rightarrow x = z$$

Якщо S та T є підмножинами X ,

$$(\forall x : X \bullet x \in S \Leftrightarrow x \in T) \Leftrightarrow S = T.$$

Якщо x та y є підмножинами X ,

$$(\forall S : \mathbf{P} X \bullet x \in S \Leftrightarrow y \in S) \Leftrightarrow x = y.$$

Назва:

¬ – заперечення

∧ – кон'юнкція

∨ – диз'юнкція

⇒ – імплікація

↔ – еквівалентність

Синтаксис:

Predicate ::= ¬ Predicate

| Predicate ∧ Predicate

| Predicate ∨ Predicate

| Predicate ⇒ Predicate

| Predicate ↔ Predicate

Ці зв'язки пропозиційної логіки задані в порядку їх пріоритету.

Опис:

Це є зв'язки пропозиційної логіки. Вони дозволяють будувати складні предикати з більш простих. Таблиці істинності задані звичним чином:

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>

p	$\neg p$
<i>true</i>	<i>false</i>
<i>false</i>	<i>true</i>

Назва:

\forall – універсальний квантор

\exists – квантор існування

\exists_1 – унікальний квантор

Синтаксис:

Predicate ::= \forall Schema-Text • Predicate
| \exists Schema-Text • Predicate
| \exists_1 Schema-Text • Predicate

Межі типу:

У предикатах $\forall S \bullet P$, $\exists S \bullet P$ та $\exists_1 S \bullet P$ текст схеми S включає локальні змінні, область дії яких включає предикат P .

Опис:

Ці квантори є кванторами логіки предикатів, з відповідним визначенням істинності.

Закони:

$$(\forall D \mid P \bullet Q) \Leftrightarrow (\forall D \bullet P \Rightarrow Q)$$

$$(\exists D \mid P \bullet Q) \Leftrightarrow (\exists D \bullet P \wedge Q)$$

$$(\exists_1 D \mid P \bullet Q) \Leftrightarrow (\exists_1 D \bullet P \wedge Q)$$

$$(\exists S \bullet P) \Leftrightarrow \neg (\forall S \bullet \neg P)$$

$$(\exists_1 x : A \bullet \dots x \dots) \Leftrightarrow (\exists x : A \bullet \dots x \dots \wedge (\forall y : A \bullet \dots y \dots \bullet y=x)).$$

Назва:

let – локальне визначення

Синтаксис:

Predicate ::= (**let** Let-Def; ...; Let-Def • Predicate)

Let-Def ::= Ident = = Expression

Межі та правила типу:

У предикаті (**let** $x_1 = E_1; \dots; x_n = E_n \bullet P$), змінні x_1, \dots, x_n є локальними змінними, область дії яких включає предикат P , але не вирази E_1, \dots, E_n . Кожна локальна змінна x_i має той же тип, що і відповідний йому вираз E_i .

Опис:

Предикаті (**let** $x_1 = E_1; \dots; x_n = E_n \bullet P$) істинний, при зв'язуванні z , тоді і тільки тоді, коли предикат P істинний при зв'язуванні, отриманому із зв'язування z , розширяючи його так, щоб кожна змінна x_i приймала значення відповідного їй виразу E_i . Якщо деякий E_i невизначений, то і весь предикат невизначений.

Закони:

$$\begin{aligned} &(\text{let } x_1 = E_1; \dots; x_n = E_n \bullet P) \\ &\Leftrightarrow (\exists_1 x_1 : t_1; \dots; x_n : t_n \mid x_1 = E_1; \dots; x_n = E_n \bullet P), \end{aligned}$$

Якщо всі E_i визначені та немає ніякого “захоплення” змінних.

Назва:

Schema-Ref – схема зсилок на предикати

Синтаксис:

Predicate ::= **Schema-Ref**
| pre **Schema-Ref**

Межі та правила типу:

У предикаті S' , де S ім'я схеми, всі компоненти декорованої схеми S' повинні бути в області дії та мати ті самі типи, що і в сигнатурі схеми.

У предикаті ‘pre S ’ де S схема, всі компоненти схеми S , за винятком декорованих (\prime) або (!) повинні бути в області дії та мати ті ж типи, що і в сигнатурі схеми.

Опис:

Посилання схеми S' може використовуватися як предикат, який істинний в точності на тих зв'язуваннях з урахуванням сигнатури, де виконується властивість схеми. Це є еквівалентом предикату $\theta S' \in S$, тут S означає $\{S \bullet \theta S\}$.

Предикат ‘pre S ’ використовується, коли S є схемою, що описує дію. Це еквівалентно предикату $PreS$, де $PreS$ є схемою, що визначена наступним чином:

$$PreS \triangleq \text{pre } S.$$

Закони:

$$S' [E_1, \dots, E_n] \Leftrightarrow \theta S' \in \{S [E_1, \dots, E_n] \bullet \theta S\}$$

Назва:

Relations – правила для символів відношень

Синтаксис:

Predicate ::= Expression Rel Expression Rel ... Rel Expression
 | Pre-Rel Expression

Правила типу:

У предикаті $E_1 R E_2$, де R інфіксний символ відношення, R повинно мати тип $\mathbf{P}(t_1 \times t_2)$, та E_1 та E_2 повинні мати типи t_1 та t_2 відповідно, для деяких типів t_1 та t_2 .

У предикаті $R E$, де R префіксний символ відношення, R повинно мати тип $\mathbf{P} t$, та E повинно мати типи t , для деякого типу t .

Опис:

Як пояснено в (Символи операцій) низка відношень

$$E_1 R_1 E_2 R_2 E_3 \dots E_{n-1} R_{n-1} E_n$$

еквівалентна з'єднанню окремих відношень:

$$E_1 R_1 E_2 \wedge E_2 R_2 E_3 \wedge \dots \wedge E_{n-1} R_{n-1} E_n.$$

Рівність та включення $=$ та \in можуть також бути присутні в такому ланцюжку.

Це правило пояснює довільні ланцюжки відношень в термінах прости відношень $E_1 R E_2$, для якого правила типу пояснені вище. Такі відношення можна записувати як $(E_1, E_2) \in (_ R _)$.

Предикат $R E$, де $R \in$ префіксним реляційном символом, є скороченням для $E \in (R _)$, цей предикат істинний, якщо значення E належить множині R .

2.8. Схема виразів

Хоча ті-ж самі символи зв'язок та квантори використовуються для формування предикатів (див. Предикати) та як оператори в схемі виразів, що формується, синтаксис \mathbf{Z} завжди дозволяє їй бути введеною з контексту, де оператор знаходиться по праву сторону від $\underline{\wedge}$.

Найпростішим видом схеми виразу є схема посилань (Зсилки схеми) та схема текстів (Тексти схеми). Круглі дужки можуть використовуватися для групування в схемі виразів:

Schema-Exp ::= Schema-Ref
 | [Schema-Text]
 | (Schema-Exp)

Назва:

\neg – схема заперечення

\wedge – схема кон'юнкції

\vee – схема диз'юнкції

\Rightarrow – схема імплікації

\Leftrightarrow – схема еквівалентності

Синтаксис:

Schema-Exp ::= \neg Schema-Exp
| Schema-Exp \wedge Schema-Exp
| Schema-Exp \vee Schema-Exp
| Schema-Exp \Rightarrow Schema-Exp
| Schema-Exp \Leftrightarrow Schema-Exp

Опис:

Ці логічні операції над схемами були представлені в частині про об'єднання схем (**Об'єднання схем**). Таблиці істинності задані звичним чином:

p	q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>false</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>

p	$\neg p$
<i>true</i>	<i>false</i>
<i>false</i>	<i>true</i>

Назва:

\forall – універсальний квантор схеми

\exists – квантор існування схеми

\exists_1 – унікальний квантор схеми

\setminus – приховування схеми

\lceil – проектування схеми

Синтаксис:

Schema-Exp ::= \forall Schema-Text • Schema-Exp
| \exists Schema-Text • Schema-Exp
| \exists_1 Schema-Text • Schema-Exp
| Schema-Exp \setminus (Ident, ..., Ident)
| Schema-Exp \lceil Schema-Exp

Опис:

Ці дії згруповані разом, так як всі вони приховують деякі з компонент своїх схем аргументів.

Для вираження схеми $\forall D \mid P \bullet S$, щоб бути дозволеною всі змінні представлені текстом схеми “ $D \mid P$ ” повинні бути серед компонент схеми S та мати ті-ж типи (**Об’єднання схем**). Подібні правила стосуються і кванторів \exists та \exists_1 .

Операція $S \setminus (x_1, \dots, x_n)$ видаляє зі схеми S компоненти x_1, \dots, x_n явно задані списком, при цьому ці компоненти повинні існувати.

Операція проектування схем $S \upharpoonright T$ приховує всі компоненти схеми S , за винятком тих, що є також компонентами схеми T . Схеми S та T повинні бути тип-сумісними, але T може містити компоненти, яких немає в S , сигнатура результату є сигнатурою T (**Об’єднання схем**).

Закони:

$S \setminus (x_1, \dots, x_n)$ те саме, що $(\exists x_1 : t_1; \dots; x_n : t_n \bullet S)$, де x_1, \dots, x_n мають типи t_1, \dots, t_n в S .

$S \upharpoonright T$ еквівалентно $(S \wedge T) \setminus (x_1, \dots, x_n)$, де x_1, \dots, x_n є компонентами S не спільні з T .

Назва:

pre – передумова

Синтаксис:

Schema-Exp ::= **pre** Schema-Exp

Опис:

Ця операція задає передумову дії, описаної схемою.

Якщо S це схема, та x'_1, \dots, x'_m компоненти S , які декоровані ($'$), та $y!_1, \dots, y!_n$ компоненти, декоровані ($!$), тоді схема “pre S ” є результатом приховування цих компонент в схемі S :

$S \setminus (x'_1, \dots, x'_m, y!_1, \dots, y!_n)$.

Містить лише компоненти S що передають стани перед операцією та її вхідні дані.

Назва:

; – послідовна композиція

>> – зсув

Синтаксис:

Schema-Exp ::= Schema-Exp ; Schema-Exp
| Schema-Exp >> Schema-Exp

Опис:

Для того, щоб композиція $S ; T$ була визначена, для кожного слова x , якщо x' є компонентою S , то саме x є компонентою T , при цьому тип цих компонент

повинен бути одним. Тоді x називається відповідною змінною стану. Крім того, типи всіх інших компонент (в тому числі вхідні, вихідні та відповідні змінні стану) повинні бути однаковими.

Схема $S ; T$ має всі компоненти S та T , за винятком компонент x' в S та x в T , де x відповідна змінна стану. Якщо $State$ – це схема, що містить лише відповідні змінні стану, тоді $S ; T$ визначена як

$\exists State'' \bullet$

$(\exists State' \bullet [S; State'' \mid \theta State' = \theta State'']) \wedge$

$(\exists State \bullet [T; State'' \mid \theta State = \theta State''])$.

У цьому визначенні, $State''$ є прихованим станом, в якому S завершується та починається T . Визначення приймає, що компоненти $State''$ не перетинаються з компонентами S та T , інакше використовується деяке інше декорування змінних.

Для того, щоб композиція зсуву $S \gg T$ була визначена потрібно, щоб для кожного слова x , такого, що S має вихідний параметр $x!$ та T має вхідний параметр $x?$, типи цих компонент повинні бути однаковими. Тоді x називається *riped* змінною. Будь-які інші відповідні компоненти (в тому числі початкові та заключні змінні стану) в S та T повинні мати однаковий тип в обох.

Схема $S \gg T$ має всі компоненти S та T , за винятком вихідних $x!$ в S та вхідних $x?$ в T , де x *riped* змінна. Якщо $Pipe$ – це схема, що містить лише відповідні *riped* змінні, тоді $S ; T$ визначена як

$\exists Pipe!?! \bullet$

$(\exists Pipe! \bullet [S; Pipe!?! \mid \theta Pipe! = \theta Pipe!?!]) \wedge$

$(\exists Pipe? \bullet [T; Pipe!?! \mid \theta Pipe? = \theta Pipe!?!])$.

У цьому визначенні, $Pipe!?!$ не перетинаються з компонентами S та T .

2.9. Види

2.9.1 Видові схеми

Якщо потрібно використати таку саму структуру для змінних, але з іншими типами, потрібно використовувати видові схеми: схема з одним, або більше формальними параметрами.

Видові визначаються подібно до інших схем, але з видовими параметрами:

$$Paragraph ::= [\begin{array}{l} \text{Schema-Name}[Ident, \dots, Ident] \\ \text{Declaration} \\ \text{Predicate; } \dots \text{ ; Predicate } \end{array}]$$

$Paragraph ::= \text{Schema-Name}[Ident, \dots, Ident] \cong \text{Schema-Exp}$

В правій стороні визначення, сукупність основних типів з формальними видовими параметрами. Кожне використання імені, окрім θ -виразів, повинне подаватися з зазначенням фактичних видових параметрів:

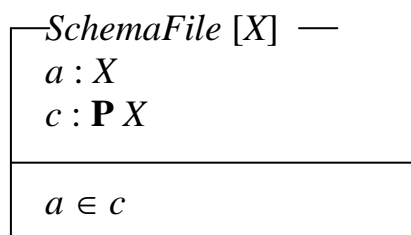
Schema-Ref ::=

Schema-Name Decoration [[Expression, ..., Expression]] [Renaming]

Renaming ::= [Ident / Ident, ..., Ident / Ident]

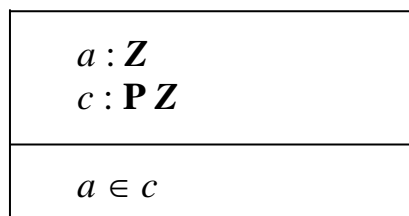
Сигнатура отриманої схеми, отримується при застосуванні оформлення, заміни типів фактичних параметрів для формальних.

Приклад.



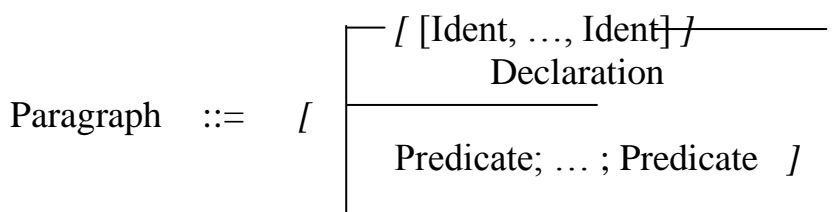
Типи a та c параметризовані формальним параметром X .

Якщо замість параметра підставити Z , т.б. $SchemaFile [Z]$, ми отримаємо наступну схему:



2.9.2 Видові константи

В описі видових констант вказуються формальні видові параметри [Ident, ..., Ident]:



Формальні видові параметри є локальними для визначення. Кожна змінна, представлена у відповідності з декларацією стає глобальною видовою константою. Ці ідентифікатори повинні бути попередньо визначені як глобальні змінні, чи видові константи. Область їх дії від визначення до закінчення

специфікації. Предикати повинні визначати значення констант унікально для кожного значення формальних параметрів. Видові константи можуть також бути представлені у скороченому вигляді:

Paragraph ::= Def-Lhs == Expression
 Def-Lhs ::= Ident [[Ident, ..., Ident]]
 | Ident In-Gen Ident
 | Pre-Gen Ident

Приклад 1. Видовий конструктор множини не порожніх множин, може виглядати наступним чином:

$\mathbf{P}_1 : \mathbf{P} (\mathbf{P} X)$
$\mathbf{P}_1 = \{ s : \mathbf{P} X \mid s \neq \emptyset \}$

При застосуванні дужки навколо видових параметрів є необов'язковою: т.б. форми $\mathbf{P}_1[s]$ та \mathbf{P}_1 є однаково прийнятними.

Приклад 2. Видова функція *first* (перший) може бути визначена наступним чином:

$first : X \times Y \rightarrow X$
$\forall x : X; y : Y \bullet first(x, y) = x$

У прикладі *first*(3, 4), видова константа *first* має тип

$\mathbf{P} ((\alpha \times \beta) \times \alpha)$, де α та β є типами підставленими замість двох видових параметрів X та Y . Аргумент (3, 4) мають тип $\mathbf{Z} \times \mathbf{Z}$, тому α та β обидва мають тип \mathbf{Z} , та тип усього виразу *first*(3, 4) є \mathbf{Z} :

$$first(3, 4) = first[\mathbf{Z}, \mathbf{Z}](3, 4).$$

Приклад 3. Задамо видове визначення символу підмножини:

$_ \subseteq _ : \mathbf{P} X \leftrightarrow \mathbf{P} X$
$\forall s, t : \mathbf{P} X \bullet$ $s \subseteq t \Leftrightarrow \forall x : X \bullet x \in s \Rightarrow x \in t$

Символ \subseteq задає відношення між двома множинами з однаковим типом $\mathbf{P} X$. При застосуванні $\{2, 3\} \subseteq \{1, 2, 3, 4\}$.

Тут $\{2, 3\} \subseteq \{1, 2, 3, 4\}$ має тип $\mathbf{P} \alpha \leftrightarrow \mathbf{P} \alpha$, де α є типом підставленим замість видового параметру X . При аргументах $\{2, 3\}$, $\{1, 2, 3, 4\}$ α має тип \mathbf{Z} , тому тип усього відношення $\{2, 3\} \subseteq \{1, 2, 3, 4\}$ є $\mathbf{P} \mathbf{Z} \leftrightarrow \mathbf{P} \mathbf{Z}$.

2.10. Математичні засоби

Важливою частиною мови Z є стандартна бібліотека, чи комплект інструментів математичних визначень.

2.10.1. Множини (Sets)

2.10.1.1 Способи завдання множин

Перерахуванням:

$$S = = \{a, b, c\}$$

Приклад 1.

Oceans = = {Atlantic, Arctic, Indian, Pacific}.

Приклад 2.

Якщо маємо наступні визначення множин: $s = = \{2, 2, 5, 5, 3\}$ та $t = = \{2, 3, 5\}$, тоді $s = t$.

Деякі множини мають спеціальні назви, наприклад $\mathbf{N} = \{0, 1, 2, 3, \dots\}$.

Включенням (comprehended):

$$\{x : s \mid p\}$$

тут x елемент множини s , такий, що істинним є предикат p .

Приклад 3. $\{x : \mathbf{N} \mid x < 10\}$.

Якщо нас цікавить деякий вираз, сформований із значень, що задовольняють предикату, а не значення безпосередньо, тоді ми записуємо:

$$\{x : s \mid p \bullet e\}$$

для того, щоб позначити набір усіх виразів e , таких, що x отриманий із s та задовольняє предикату p .

Приклад 4.

Множину адрес людей, що водять червону машину можна задати наступним чином: $\{x : Person \mid x \text{ drives a red car} \bullet address(x)\}$.

Якщо у нас немає жодних обмежень на вибір значень, то ми можемо задати множину наступним чином: $\{x : s \bullet e\}$.

$$\text{При цьому } \{x : s \bullet e\} = \{x : s \mid true \bullet e\}$$

Приклад 5.

Множину адрес людей можна задати наступним чином:
 $\{x : Person \bullet address(x)\}$.

Визначення може мати і наступний вигляд: $\{x_1 : s_1; \dots; x_n : s_n \mid p \bullet e\}$.

Множина множин:

Якщо множина складається з усіх підмножин деякої множини, то її можна задати наступним чином: $\mathbf{P} T$. Тут T – множина.

2.10.1.2 Операції над множинами

Назва:

\neq – нерівність

\notin – не включення

Визначення:

$[X]$ $_ \neq _ : X \leftrightarrow X$ $_ \notin _ : X \leftrightarrow \mathbf{P} X$
$\forall x, y : X \bullet x \neq y \leftrightarrow \neg(x = y)$ $\forall x : X; S : \mathbf{P} X \bullet x \notin S \leftrightarrow \neg(x \in S)$

Закони:

$$x \neq y \Rightarrow y \neq x$$

Назва:

\emptyset – порожня множина

\subseteq – включення

\subset – строге включення

\mathbf{P}_1 – не порожня підмножина

Визначення:

$$\emptyset[X] = = \{x : X \mid false\}$$

$[X]$ $_ \subseteq _, _ \subset _ : \mathbf{P} X \leftrightarrow \mathbf{P} X$
$\forall S, T : \mathbf{P} X \bullet (S \subseteq T \leftrightarrow (\forall x : X \bullet x \in S \Rightarrow x \in T)) \wedge$ $(S \subset T \leftrightarrow S \subseteq T \wedge S \neq T)$ $\mathbf{P}_1 X = = \{S : \mathbf{P} X \mid S \neq \emptyset\}$

Закони:

$$x \notin \emptyset$$

$$S \subseteq T \leftrightarrow S \in \mathbf{P} T$$

$$S \subseteq S$$

$$\neg(S \subset T)$$

$$S \subseteq T \wedge T \subseteq S \Leftrightarrow S = T$$

$$\neg (S \subset T \wedge T \subset S)$$

$$S \subseteq T \wedge T \subseteq V \Rightarrow S \subseteq V$$

$$S \subset T \wedge T \subset V \Rightarrow S \subset V$$

$$\emptyset \subseteq S$$

$$\emptyset \subset S \Leftrightarrow S \neq \emptyset$$

$$\mathbf{P}_1 X = \emptyset \Leftrightarrow X = \emptyset$$

$$X \neq \emptyset \Leftrightarrow X \in \mathbf{P}_1 X$$

Назва:

\cup – об'єднання множин

\cap – перетин множин

\setminus – різниця множин

Визначення:

$$\begin{array}{|l} [X] \\ \hline \cup, \cap, \setminus : \mathbf{P} X \times \mathbf{P} X \rightarrow \mathbf{P} X \\ \hline \forall S, T : \mathbf{P} X \bullet \\ S \cup T = \{x : X \mid x \in S \vee x \in T\} \wedge \\ S \cap T = \{x : X \mid x \in S \wedge x \in T\} \wedge \\ S \setminus T = \{x : X \mid x \in S \wedge x \notin T\} \end{array}$$

Закони:

$$S \cup S = S \cap S = S \cup \emptyset = S \setminus \emptyset = S$$

$$S \cap \emptyset = S \setminus S = \emptyset \setminus S = \emptyset$$

$$S \cup T = T \cup S$$

$$S \cap T = T \cap S$$

$$S \cup (T \cup V) = (S \cup T) \cup V$$

$$S \cap (T \cap V) = (S \cap T) \cap V$$

$$S \cup (T \cap V) = (S \cup T) \cap (S \cup V)$$

$$S \cap (T \cup V) = (S \cap T) \cup (S \cap V)$$

$$(S \cap T) \cup (S \setminus T) = S$$

$$S \cup (T \setminus V) = (S \cup T) \setminus (V \setminus S)$$

$$(S \setminus T) \cap T = \emptyset$$

$$S \cap (T \setminus V) = (S \cap T) \setminus V$$

$$S \setminus (T \setminus V) = (S \setminus T) \cup (S \cap V)$$

$$(S \cup T) \setminus V = (S \setminus V) \cup (T \setminus V)$$

$$(S \setminus T) \setminus V = S \setminus (T \cup V)$$

$$S \setminus (T \cap V) = (S \setminus T) \cup (S \setminus V)$$

Назва:

\cup – узагальнене об'єднання

\cap – узагальнений перетин

Визначення:

$[X]$ $\cup, \cap, : \mathbf{P}(\mathbf{P} X) \rightarrow \mathbf{P} X$
$\forall A : \mathbf{P}(\mathbf{P} X) \bullet$ $\cup A = \{x : X \mid (\exists S : A \bullet x \in S)\} \wedge$ $\cap A = \{x : X \mid (\forall S : A \bullet x \in S)\}$

Закони:

$$\begin{aligned} \cup(A \cup B) &= (\cup A) \cup (\cup B) & \cap(A \cup B) &= (\cap A) \cap (\cap B) \\ \cup[X] \emptyset &= \emptyset & \cap[X] \emptyset &= X \\ S \cap (\cup A) &= \cup \{T : A \bullet S \cap T\} & S \cup (\cap A) &= \cap \{T : A \bullet S \cup T\} \\ (\cup A) \setminus S &= \cup \{T : A \bullet T \setminus S\} & S \setminus (\cap A) &= \cup \{T : A \bullet S \setminus T\} \\ A \subseteq B &\Rightarrow \cup A \subseteq \cup B & A \subseteq B &\Rightarrow \cap B \subseteq \cap A \\ A \neq \emptyset &\Rightarrow S \setminus (\cup A) = \cap \{T : A \bullet S \setminus T\} \\ A \neq \emptyset &\Rightarrow (\cup A) \setminus S = \cap \{T : A \bullet T \setminus S\} \end{aligned}$$

Назва:

first, second – проєкція впорядкованих пар

Визначення:

$[X]$ $first : X \times Y \rightarrow X$ $second : X \times Y \rightarrow Y$
$\forall x : X; y : Y \bullet$ $first(x, y) = x \wedge second(x, y) = y$

Закони:

$$(first\ p, second\ p) = p$$

2.10.2. Відношення

Назва:

\leftrightarrow – бінарне відношення

\mapsto – відображення

Визначення:

$$X \leftrightarrow Y = = \mathbf{P}(X \times Y)$$

[X, Y]	_____
_ ↦ _ : X × Y → X × Y	
∃ x : X; y : Y •	_____
x ↦ y = (x, y)	

Назва:

dom, ran – область визначення та множина значень відношення

Визначення:

[X, Y]	_____
dom : (X ↔ Y) → P(X)	
ran : (X ↔ Y) → P(Y)	
∃ R : X ↔ Y •	_____
dom R = { x : X; y : Y x <u>R</u> y • x } ∧	
ran R = { x : X; y : Y x <u>R</u> y • y }	

Закони:

- $x \in \text{dom } R \Leftrightarrow (\exists y : Y \bullet x \underline{R} y)$
- $x \in \text{ran } R \Leftrightarrow (\exists x : X \bullet x \underline{R} y)$
- $\text{dom } \{x_1 \mapsto y_1, \dots, x_n \mapsto y_n\} = \{x_1, \dots, x_n\}$
- $\text{ran } \{x_1 \mapsto y_1, \dots, x_n \mapsto y_n\} = \{y_1, \dots, y_n\}$
- $\text{dom } (Q \cup R) = (\text{dom } Q) \cup (\text{dom } R)$
- $\text{ran } (Q \cup R) = (\text{ran } Q) \cup (\text{ran } R)$
- $\text{dom } (Q \cap R) \subseteq (\text{dom } Q) \cap (\text{dom } R)$
- $\text{ran } (Q \cap R) \subseteq (\text{ran } Q) \cap (\text{ran } R)$
- $\text{dom } \emptyset = \emptyset$
- $\text{ran } \emptyset = \emptyset$

Назва:

- id– відношення ідентичності
- ; – відношення композиції
- – обернене відношення композиції

Визначення:

$$\text{id } X = \{ x : X \bullet x \mapsto x \}$$

$\begin{array}{l} [X, Y, Z] \\ \underline{_}; \underline{_} : (X \leftrightarrow Y) \times (Y \leftrightarrow Z) \rightarrow (X \leftrightarrow Z) \\ \underline{_} \circ \underline{_} : (Y \leftrightarrow Z) \times (X \leftrightarrow Y) \rightarrow (X \leftrightarrow Z) \end{array}$
$\begin{array}{l} \forall Q : X \leftrightarrow Y; R : Y \leftrightarrow Z \bullet \\ Q; R = R \circ Q \{ x : X; y : Y; z : Z \mid \\ x \underline{Q} y \wedge y \underline{R} z \bullet x \mapsto z \} \end{array}$

Закони:

$$(x \mapsto x') \in \text{id } X \Leftrightarrow x = x' \in X$$

$$(x \mapsto z) \in P; Q \Leftrightarrow (\exists y : Y \bullet x \underline{P} y \wedge y \underline{Q} z)$$

$$P; (Q; R) = (P; Q); R$$

$$\text{id } X; P = P$$

$$P; \text{id } Y = P$$

$$\text{id } V; \text{id } W = \text{id } (V \cap W)$$

$$(f \circ g)(x) = f(g(x))$$

Назва:

- \triangleleft – область обмежень
- \triangleright – діапазон обмежень

Визначення:

$\begin{array}{l} [X, Y] \\ \underline{_} \triangleleft \underline{_} : \mathbf{P} X \times (X \leftrightarrow Y) \rightarrow (X \leftrightarrow Y) \\ \underline{_} \triangleright \underline{_} : (X \leftrightarrow Y) \times \mathbf{P} Y \rightarrow (X \leftrightarrow Y) \end{array}$
$\begin{array}{l} \forall S : \mathbf{P} X; R : X \leftrightarrow Y \bullet \\ S \triangleleft R = \{ x : X; y : Y \mid x \in S \wedge x \underline{R} y \bullet x \mapsto y \} \\ \forall R : X \leftrightarrow Y; T : \mathbf{P} Y \bullet \\ R \triangleright T = \{ x : X; y : Y \mid x \underline{R} y \wedge y \in T \bullet x \mapsto y \} \end{array}$

Закони:

$$S \triangleleft R = \text{id } S; R = (S \times Y) \cap R$$

$$R \triangleright T = R; \text{id } T = R \cap (X \times T)$$

$$\text{dom } (S \triangleleft R) = S \cap (\text{dom } R)$$

$$\text{ran } (R \triangleright T) = (\text{ran } R) \cap T$$

$$S \triangleleft R \subseteq R$$

$$R \triangleright T \subseteq R$$

$$(S \triangleleft R) \triangleright T = S \triangleleft (R \triangleright T)$$

$$S \triangleleft (V \triangleleft R) = (S \cap V) \triangleleft R$$

$$(R \triangleright T) \triangleright W = R \triangleright (T \cap W)$$

Назва:

\triangleleft – область анти-обмежень

\triangleright – діапазон анти-обмежень

Визначення:

$[X, Y]$ $_ \triangleleft _ : \mathbf{P} X \times (X \leftrightarrow Y) \rightarrow (X \leftrightarrow Y)$ $_ \triangleright _ : (X \leftrightarrow Y) \times \mathbf{P} Y \rightarrow (X \leftrightarrow Y)$
$\forall S : \mathbf{P} X; R : X \leftrightarrow Y \bullet$ $S \triangleleft R = \{ x : X; y : Y \mid x \notin S \wedge x \underline{R} y \bullet x \mapsto y \}$ $\forall R : X \leftrightarrow Y; T : \mathbf{P} Y \bullet R \triangleright T = \{ x : X; y : Y \mid x \underline{R} y \wedge y \notin T \bullet x \mapsto y \}$

Закони:

$$S \triangleleft R = (X \setminus S) \triangleleft R$$

$$R \triangleright T = R \triangleright (Y \setminus T)$$

$$(S \triangleleft R) \cup (S \triangleleft R) = R$$

$$(R \triangleright T) \cup (R \triangleright T) = R$$

Назва:

\sim – відношення інверсії

Визначення:

$[X, Y]$ $_ \sim _ : (X \leftrightarrow Y) \rightarrow (Y \leftrightarrow X)$
$\forall R : X \leftrightarrow Y \bullet$ $R \sim = \{ x : X; y : Y \mid x \underline{R} y \bullet y \mapsto x \}$

Закони:

$$(y \mapsto x) \in R^{\sim} \Leftrightarrow (x \mapsto y) \in R$$

$$(R^{\sim})^{\sim} = R$$

$$(Q; R)^{\sim} = R^{\sim}; Q^{\sim}$$

$$(\text{id } V)^{\sim} = \text{id } V$$

$$\text{dom } (R)^{\sim} = \text{ran } R$$

$$\text{ran } (R)^{\sim} = \text{dom } R$$

$$\text{id } (\text{dom } R) \subseteq R; R^{\sim}$$

$$\text{id } (\text{ran } R) \subseteq R^{\sim}; R$$

Назва:

$_(|_)|$ – образ відношення

Визначення:

$$\begin{array}{|l} [X, Y] \\ \hline _(|_)| : (X \leftrightarrow Y) \times \mathbf{P} X \rightarrow \mathbf{P} Y \\ \hline \forall R : X \leftrightarrow Y; S : \mathbf{P} X \bullet \\ R(|S|) = \{ x : X; y : Y \mid x \in S \wedge x \underline{R} y \bullet y \} \end{array}$$

Закони:

$$x \in R(|S|) \Leftrightarrow (\exists x : X \bullet x \in S \wedge x \underline{R} y)$$

$$R(|S|) = \text{ran } (S \triangleleft R)$$

$$\text{dom } (Q; R) = Q^{\sim}(|\text{dom } R|)$$

$$\text{ran } (Q; R) = R^{\sim}(|\text{ran } Q|)$$

$$R(|S \cup T|) = R(|S|) \cup R(|T|)$$

$$R(|S \cap T|) = R(|S|) \cap R(|T|)$$

$$R(|\text{dom } R|) = \text{ran } R$$

$$\text{dom } R = \textit{first } (|R|)$$

$$\text{ran } R = \textit{second } (|R|)$$

Назва:

$_ \oplus _$ – виключення

Визначення:

$[X, Y]$ $_ \oplus _ : (X \leftrightarrow Y) \times (X \leftrightarrow Y) \rightarrow (X \leftrightarrow Y)$
$\forall Q, R : X \leftrightarrow Y \bullet$ $Q \oplus R = ((\text{dom } R) \not\leftarrow Q) \cup R$

Закони:

$$R \oplus R = R$$

$$P \oplus (Q \oplus R) = (P \oplus Q) \oplus R$$

$$\emptyset \oplus R = R \oplus \emptyset = R$$

Назва:

- $_ \oplus _$ – транзитивне замикання
- $_ \oplus _$ – рефлексивно-транзитивне замикання

Визначення:

$[X]$ $_ \oplus _ : (X \leftrightarrow X) \rightarrow (X \leftrightarrow X)$
$\forall R : X \leftrightarrow X \bullet$ $R^+ = \cap \{Q : X \leftrightarrow X \mid R \subseteq Q \wedge Q \subseteq Q\}$ $R^* = \cap \{Q : X \leftrightarrow X \mid \text{id } X \subseteq Q \wedge R \subseteq Q \wedge Q \subseteq Q\}$

Закони:

$$R \subseteq R^+$$

$$R^+; R^+ \subseteq R^+$$

$$R \subseteq Q \wedge Q \subseteq Q \Rightarrow R^+ \subseteq Q$$

$$\text{id } X \subseteq R^*$$

$$R \subseteq R^*$$

$$R^*; R^* \subseteq R^*$$

$$\text{id } X \subseteq Q \wedge R \subseteq Q \wedge Q \subseteq Q \Rightarrow R^* \subseteq Q$$

2.10.3. Функції

Назва:

- \rightarrow Часткова однозначна функція
- \rightarrow Тотальна функція
- \rightarrow Часткова ін'єктивна
- \rightarrow Тотальна ін'єктивна
- \rightarrow Часткова сюр'єктивна
- \rightarrow Тотальна сюр'єктивна
- \rightarrow Бієкція

Визначення:

$$X \twoheadrightarrow Y = \{f: X \twoheadrightarrow Y \mid \text{ran } f = Y\}$$

$$X \leftrightarrow Y = \{f: X \leftrightarrow Y \mid (\forall x: X; y_1, y_2: Y \bullet \\ (x \mapsto y_1) \in f \wedge (x \mapsto y_2) \in f \Rightarrow y_1 = y_2)\}$$

$$X \rightarrow Y = \{f: X \twoheadrightarrow Y \mid \text{dom } f = X\}$$

$$X \rightsquigarrow Y = \{f: X \twoheadrightarrow Y \mid (\forall x_1, x_2: \text{dom } f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2)\}$$

$$X \twoheadrightarrow Y = (X \rightsquigarrow Y) \cap (X \rightarrow Y)$$

$$X \twoheadrightarrow Y = (X \twoheadrightarrow Y) \cap (X \rightarrow Y)$$

$$X \rightsquigarrow Y = (X \twoheadrightarrow Y) \cap (X \rightsquigarrow Y)$$

2.10.4. Числа та обмеження

Назва:

\mathbb{N} – натуральні числа

\mathbb{Z} – цілі числа

$+$, $-$, $*$, div , mod – арифметичні операції

$<$, \leq , \geq , $>$ – числові порівняння

Визначення:

$[Z]$
$_{+}, _{-}, _{*} : Z \times Z \rightarrow Z$
$_{\text{div}}, _{\text{mod}} : Z \times (Z \setminus \{0\}) \rightarrow Z$
$_{-} : Z \rightarrow Z$
$_{<}, _{\leq}, _{\geq}, _{>} : Z \leftrightarrow Z$
описати самостійно

$$N = \{n: Z \mid n \geq 0\}$$

Назва:

\mathbb{N}_1 – строго додатні цілі

succ – функція наступного

.. – область чисел

Визначення:

$succ : N \rightarrow N$ $.._ : Z \times Z \rightarrow \mathbf{P} Z$
$\forall n : N \bullet succ(n) = n + 1$ $\forall a, b : Z \bullet$ $a .. b = \{ k : Z \mid a \leq k \leq b \}$

$$N_1 = N \setminus \{0\}$$

Назва:

R^k – повтор

Визначення:

$[X]$ $iter : Z \rightarrow (X \leftrightarrow X) \rightarrow (X \leftrightarrow X)$
$\forall R : X \leftrightarrow X \bullet$ $iter\ 0\ R = id\ X \wedge (\forall k : N \bullet iter\ (k + 1)\ R = R ; (iter\ k\ R)) \wedge$ $(\forall k : N \bullet iter\ (-k)\ R = iter\ k\ (R \sim))$

Як правило $iter\ k\ R$ записують R^k .

Назва:

- \mathbf{F} – скінченні множини
- \mathbf{F}_1 – непорожні скінченні множини
- $\#$ – кількість елементів множини

Визначення:

$$\mathbf{F} X = \{ S : \mathbf{P} X \mid \exists n : N \bullet \exists f : 1..n \rightarrow S \bullet \text{ran } f = S \}$$

$$\mathbf{F}_1 X = \mathbf{F} X \setminus \{\emptyset\}$$

$[X]$ $\# : \mathbf{F} X \rightarrow N$
$\forall S : \mathbf{F} X \bullet$ $\# S = (\mu n : N \mid (\exists f : 1..n \twoheadrightarrow S \bullet \text{ran } f = S))$

Назва:

- \twoheadrightarrow – скінченна часткова
- \succrightarrow – скінчення часткова

$$X \twoheadrightarrow Y == \{f : X \rightarrow Y \mid \text{dom } f \in \mathbb{F} X\}$$

$$X \rightsquigarrow Y == (X \twoheadrightarrow Y) \cap (X \rightarrow Y)$$

Назва:

min, max – мінімум та максимум множини чисел

Визначення:

$min : \mathbf{P}_1 \mathbf{Z} \rightarrow \mathbf{Z}$ $max : \mathbf{P}_1 \mathbf{Z} \rightarrow \mathbf{Z}$
$min : \{S : \mathbf{P}_1 \mathbf{Z}; m : \mathbf{Z} \mid$ $m \in S \wedge (\forall n : S \bullet m \leq n) \bullet S \mapsto m\} max : \{S : \mathbf{P}_1 \mathbf{Z}; m : \mathbf{Z} \mid m \in S \wedge (\forall n : S \bullet m \geq n) \bullet S \mapsto m\} $

2.10.5. Послідовності

Назва:

seq – скінченні послідовності

seq₁ – непорожні скінченні послідовності

iseq – ін’єктивні послідовності

Визначення:

$$seq X == \{f : \twoheadrightarrow X \mid \text{dom } f = 1 .. \#f\}$$

$$seq_1 X == \{f : seq X \mid \#f > 0\}$$

$$iseq X == seq X \cap \rightsquigarrow X$$

Назва:

$\hat{\ }^$ – конкатенація

rev – обернення

Визначення:

$[X]$ $\hat{\ }^ : seq X \times seq X \rightarrow seq X$ $rev : seq X \rightarrow seq X$
$\forall s, t : seq X \bullet$ $s \hat{\ }^ t = s \cup \{n : \text{dom } t \bullet n + \#s \mapsto t(n)\} \forall s : seq X \bullet rev s = (\lambda n : \text{dom } s \bullet s (\#s - n + 1)) $

Назва:

head, last, tail, front – розкладення послідовності

Визначення:

[X]
$head, last : seq_1 X \rightarrow X$ $tail, front : seq_1 X \rightarrow seq X$
$\forall s : seq_1 X \bullet$ $head\ s = s(1) \wedge$ $last\ s = s(\#s) \wedge$ $tail\ s = (\lambda n : 1 .. \#s - 1 \bullet s(n+1)) \wedge$ $front\ s = (1 .. \#s - 1) \triangleleft s$

Назва:

prefix – префікс послідовності

suffix – суфікс послідовності

in – частина послідовності

Визначення:

[X]
$_prefix_ , _suffix_ , _in_ : seq X \leftrightarrow seq X$
$\forall s, t : seq X \bullet$ $s\ prefix\ t \Leftrightarrow (\exists v : seq X \bullet s \hat{=} v = t) \wedge$ $s\ suffix\ t \Leftrightarrow (\exists u : seq X \bullet u \hat{=} s = t) \wedge$ $s\ in\ t \Leftrightarrow (\exists u, v : seq X \bullet u \hat{=} s \hat{=} v = t)$

2.11. Вільні типи

В процесі побудови специфікації потрібно визначати різноманітні рекурсивні структури даних: списки, масиви, дерева. Ці структури можуть бути змодельовані з використанням комбінацій множин та відношень, але результат визначення може виявитися занадто багатослівним. Більш зручно та коротко задати ці типи з використанням так званих *вільних типів*: множини з структурованою інформацією.

Синтаксис визначення вільного типу наступний:

Paragraph ::= Ident ::= Branch | ... | Branch

Branch ::= Ident [«Expression»]

Приклад 1.

Факультет: *faculty* ::= dean | professor | reader | student.

Приклад 2.

Множина *nat*: *nat* ::= zero | succ« *nat* ».

Кожен елемент *nat* є або zero, або наступник від натурального числа, zero не є наступником жодного числа, та кожен елемент з *nat* має лише одного наступника. Множина *nat* є містить наступні елементи: zero, succ zero, succ (succ zero), succ (succ (succ zero)), і так далі.

Приклад 3. Бінарні дерева

Ми можемо описати набір бінарних дерев, маркованих натуральними числами. Нехай константа *tip* є порожнім деревом, тоді якщо *n* є натуральним числом, *t*₁ та *t*₂ є деревами, то *fork*(*n*, *t*₁, *t*₂) є деревом:

TREE ::= *tip* | *fork* « **N** × *TREE* × *TREE* ».

Це вільне визначення типу еквівалентне наступному аксіоматичному опису:

[*TREE*]

tip : *TREE*

fork : **N** × *TREE* × *TREE* → *TREE*

disjoint <{*tip*}, ran *fork*>

∀ *W* : **P** *TREE* •

{*tip*} ∪ *fork*(**N** × *W* × *W*) ⊆ *W*

⇒ *TREE* ⊆ *W*

Нехай, ми хочемо довести, що предикат *P*(*t*) істинний для усіх дерев *t*. Принцип індукції стверджує, що достатньо довести наступні два факти:

1) *P*(*tip*) виконується

2) якщо *P*(*t*₁) та *P*(*t*₂) виконуються, то має місце *P*(*fork*(*n*, *t*₁, *t*₂)):

∀ *n* : **N**; *t*₁, *t*₂ : *TREE* •

P(*t*₁) ∧ *P*(*t*₂) ⇒ *P*(*fork*(*n*, *t*₁, *t*₂)).

Якщо ці факти мають місце, то принцип індукції дозволяє отримати наступне твердження: $\forall t : TREE \bullet P(t)$. Нехай, W це множина дерев, що задовольняють P : тоді маємо,

$$W = \{t : TREE \mid P(t)\}.$$

Твердження 1) говорить, що $\text{tip} \in W$, а твердження 2) говорить, що $\text{fork} \langle \mathbb{N} \times W \times W \rangle \subseteq W$, тому за принципом індукції, $TREE \subseteq W$. Це означає, що $\forall t : TREE \bullet t \in W$, що еквівалентне $\forall t : TREE \bullet P(t)$.

2.12. Δ та \exists домовленості

Дії на типах даних визначені схемами, які мають дві копії станів змінних серед своїх компонент: не декорована множина, що відповідає стану типу даних перед дією, та множиною з штрихом, що відповідає стану після дії. Для того, щоб більш зручно оголошувати ці змінні, є домовленість, що кожен раз, коли схема A введена як стан абстрактного типу даних, схема ΔA неявно визначена як комбінація A та A' , якщо різні визначення не зроблені явно:



З цим визначенням, кожна дія на типі даних може бути задана розширяючи ΔA з деклараціями вхідних та вихідних операцій та предикатів, що дають передумову та пост-умову.

Знак Δ є лише літерою в імені цієї схеми, та неявне визначення ΔA не більше ніж домовленість.

Видові схеми також можуть використовувати Δ . Якщо схема A має два видові параметри, тоді використовують ΔA , яка неявно визначається наступним чином:



Як раніше, специфікація є вільною для визначення ΔA іншими способами, та навіть з іншою кількістю видових параметрів.

Багато типів даних мають операції, які звертаються до інформації в стані взагалі без зміни стану. Цей факт може бути записаний, включенням рівняння θA

$= \theta A'$ в пост-умові операції, але зручно мати спеціальну схему ΞA , в якій ці дії доступу можуть бути побудовані. Подібно до ΔA , схема ΞA неявно визначається кожен раз, коли введена схема A , тоді маємо:

ΞA
A
A'
$\theta A = \theta A'$

Як і раніше, видові схеми також можуть використовувати Ξ . Якщо схема A має два видові параметри, тоді використовують ΞA , яка неявно визначається наступним чином:

$\Xi A[X, Y]$
$A[X, Y]$
$A'[X, Y]$
$\theta A = \theta A'$

2.13. Оцінювання Z специфікації

Як і при написанні програми створення специфікації не означає її правильність. Повнота та несуперечливість залишаються загальними проблемами. Повнота – відносно того, чи написали ми достатньо системних властивостей специфікуємої системи. Несуперечливість – відносно того чи не написано занадто багато системних властивостей так що вони не можуть бути одночасно істинними.

Animation

Animation розглядає наступні питання:

- перевірка синтаксису;
- перевірка типів;
- перевірка несуперечливості специфікації;
- перевірка виводимості властивостей із попередньо заданих умов.

Запуск (**animation**) специфікації, чи макетування, складається з прямого виконання специфікації для перевірки її функціональності. Система запуску - засіб, що бере специфікацію застосовує описані операції та спостерігає за результатами. Запуск специфікації є одним з підходів, що допомагає на ранніх стадіях виявити суперечливості специфікації. Проте, навіть після того, як ми

переконалися в повноті та несуперечливості специфікації питання про точність специфікування предметної області залишається відкритою. Це питання надзвичайно складне. Зазвичай вимоги до системи не мають ніякого іншого формального втілення, і ми не маємо на що спиратися для оцінювання формальної специфікації. Знову запуск, який дозволяє бути свідком системної поведінки – головний помічник і в цьому питанні. Проте, цей підхід має той же недолік, що і тестування програми – можна виявити помилки, але не гарантувати їх відсутність.

Виведення

Один природний спосіб використання формальної специфікації полягає в тому, щоб вивести з неї істинність критичних властивостей специфікуємої системи. Якщо це буде виконано, то ми отримаємо впевненість, що правильно описали наші наміри щодо системи. З цієї причини деякі інструментальні засоби для Z специфікацій містять prover. Це повинно полегшити механічну перевірку, що властивості зазначеної системи дійсно є логічними наслідками специфікації системи.

Перевірка моделі (Model Checking)

Альтернативний підхід до оцінки специфікації перевірка моделі (model checking). При цьому розглядаються інтерпретації та перевіряються на істинність формули на цих інтерпретаціях. Раніше відмічалось невідповідність такого підходу (він близький до тестування на прикладах). Взагалі кажучи існує необмежений діапазон інтерпретацій, які неможливо охопити. Тому при перевірці моделі метою є знаходження помилок. Як і при тестування програми незнаходження помилок не гарантує правильність, проте для ідентифікації помилок цей метод корисний.

2.14. Приклад застосування

Для прикладу опишемо наступну систему Адресна книга: в адресній книзі здійснюються записи про осіб та їхні адреси. Система підтримки адресної книги повинна забезпечувати можливість додавання/видалення особи та відповідно зміну адреси, а також видавати корисну довідкову інформацію (наприклад, про адресу певної особи, і т.і.).

Нехай максимальна кількість елементів в адресній книзі – $MaxSize$.

$MaxSize : N$

$MaxSize \leq 65535$

Можливі повідомлення:

MESSAGE ::= success

| *found*
 | *overflow*
 | *known*
 | *unknown*

Стан системи, описаний в Z виглядає наступним чином:

AddressBook::

person: **P** PERSON

address: PERSON \rightarrow ADDRESS

person = **dom** *address*

person \leq *MaxSize*

Тут і далі, стани представлені, як набір *person* імен осіб, адреси яких зберігаються в базі даних та часткової функції *address*, яка зіставляє адресу деякій особі. Інваріантне відношення між цими параметрами зазначає, що в базі даних зберігаються лише імена осіб, для яких зазначено адресу. При цьому одна особа може мати лише одну адресу, але за однією адресою може проживати кілька осіб. Наприклад, можливий наступний стан системи:

person = {Іванов, Петров, Сидоров}

address = {Іванов \mapsto Київ, вул. Леніна 1, кв. 1,
 Петров \mapsto Київ, вул. Леніна 1, кв. 10,
 Сидоров \mapsto Київ, вул. Леніна 1, кв. 1 }.

Операції додавання нового запису та знаходження особи можуть бути специфіковані наступним чином:

InsertOk

Δ *AddressBook*

name? : PERSON

addr? : ADDRESS

message! : MESSAGE

name? \notin *person*

person < *MaxSize*

address' = *address* \cup {*name?* \mapsto *addr?*}

message! = *success*

FindOk

\exists *AddressBook*

name? : PERSON

addr! : ADDRESS

message! : MESSAGE

name? ∈ *person*

addr! = *address(name?)*

message! = *found*

DeleteOk

Δ *AddressBook*

name? : PERSON

message! : MESSAGE

name? ∈ *person*

address' = {*name?*} ↯ *address*

message! = *success*

Компоненти станів, які відносяться до операції, разом з їх типами, зазначаються після оголошення схеми даних Δ *AddressBook*, так у випадку операції *InsertOk* маємо два вхідні параметри *name?* та *address?*, а у випадку операції *FindOk* маємо один вхідний параметр *name?*, та один вихідний(шуканий) – *address!*. Тобто вхідні параметри позначаються знаком “?” після імені параметру, а вихідні – “!”. В пост-умові нові значення станів компонент визначені їхнім іменем зі штрихом (напр. *замовлення'*). У випадку операції *FindOk* компонента стану системи *address* незмінна, призначена лише для читання, тобто її заключне значення рівне початковому, що і зазначено, як *address' = address*.

Опишемо ініціалізацію, тобто стан, який виникає перед першим використанням бази даних. При цьому “адресна книга” немає жодного запису.

DataBookInit

AddressBook

address = ∅

Специфікація описує, поки що просту базу даних “Адресна книга”, але некоректні вхідні дані можуть привести до помилок в описаних операціях. Числення схеми може використати додаткове розширення специфікації, для того, щоб вказати на зроблені при вводі даних помилки.

Почнемо з опису множини нових подій для бази даних. Передумова кожного випадку описує обставини, при який дія може зазнавати невдачі, та пост-умова визначає, що стан системи залишається незмінним.

Кожен з випадків помилки вводу матиме один вихідний параметр – *message!* для повідомлення про помилку. Параметр *message!* є рядком.

При виконанні операції *InsertAddress* виникне помилка, якщо *name? ∈ person*, тобто адреса для цієї особи вже відома, тому маємо:

InsertKnown

$\exists \textit{AddressBook}$
name? : PERSON
message! : MESSAGE

name? ∈ person
message! = known

Також, при виконанні операції *InsertAddress* може виникнути помилка переповнення, тому маємо:

InsertOverflow

$\exists \textit{AddressBook}$
name? : PERSON
message! : MESSAGE

person = MaxSize
message! = overflow

Тепер можна описати операцію *Insert*:

$\textit{Insert} \cong \textit{InsertOk} \vee \textit{InsertOverflow} \vee \textit{InsertKnown}$.

Аналогічно, при виконанні операції *FindOk* виникне помилка, якщо *name? ∉ person*, тобто адреса для цієї особи невідома, тому маємо:

FindUnknown

$\exists \textit{AddressBook}$
name? : PERSON
message! : MESSAGE

name? ∉ person
message! = unknown

Тоді операція *Find* матиме вигляд:

$\textit{Find} \cong \textit{FindOk} \vee \textit{FindUnknown}$.

Аналогічна помилка можк виникнути при виконанні операції *DeleteOk* тому маємо: $\textit{Delete} \cong \textit{DeleteOk} \vee \textit{FindUnknown}$.

Контрольні запитання та вправи

1. Що таке множина в мові Z ?
2. Як задати множину в мові Z ?
3. Що таке кортеж та декартовий добуток в мові Z та як їх задати?
4. Чи може існувати в Z -специфікації кортеж без компонент?
5. Чи може існувати в Z -специфікації кортеж з однією компонентою?
6. Чи може існувати в Z -специфікації кортеж з двома компонентами?
7. Що таке схема?
8. Які логічні операції можна виконувати над схемами?
9. Побудувати схеми $\neg A$, $A \vee G$, $A \Rightarrow G$, $A \Leftrightarrow G$ та $G \Rightarrow A$. Тут схеми A та G мають наступний вигляд:

A:

$x, y : Z$
$x < y$

G:

$y : Z$ $z : 1..10$
$y = z * z$

10. Якщо можливо побудувати наступні схеми (якщо ні пояснити чому):

- a) $A \vee B$;
- b) $B \Rightarrow G$;
- c) $\neg D$;
- d) $B \Leftrightarrow D$.

Тут схеми A та G задані у вправі 9, а схеми B та D мають вигляд:

B:

$x, y : Z$ $z : Z \rightarrow Z$
$x < y \wedge z = (x, y)$

D:

$z : Z \leftrightarrow Z$ $k : 1 .. 10$
$z = (k, k)$

11. Задати схеми:

a) $y : Z \mid y > 5 \bullet A$

b) $x : Z \mid x < 25 \bullet A$.

12. Як визначити глобальні змінні в мові Z?

13. Як визначити схему в мові Z?

14. Які множини задають наступні вирази:

a) $\{x : N, y : N \mid x > y \wedge x \text{ prime number} \wedge x + y < 100 \bullet y\}$.

b) $\{x : N, y : N \mid x > y \wedge x \text{ prime number} \wedge x + y < 100 \bullet x\}$.

c) $\{x : Person, y : Person \mid address(x) = address(y) \wedge x \text{ drives a red car} \wedge y \text{ studies at University} \bullet x\}$.

d) $\{x : Person, y : Person \mid address(x) = address(y) \wedge x \text{ drives a red car} \wedge y \text{ studies at University} \bullet y\}$.

15. Задайте множини:

a) множину натуральних чисел, кожне з яких менше за деяке просте число;

b) множину адрес водіїв машин червоного кольору за однією адресою з якими проживають діти віком молодші 10 років;

c) множину номерів машин червоного кольору водії яких проживають за однією адресою з студентом університету.

ВАРІАНТИ ЗАВДАНЬ ЧАСТИНИ 2

Для наступних завдань, у відповідності до обраного варіанту розробити Z-специфікацію.

1. Система обліку "автомобілі – власники – довіреності".

Система повинна забезпечувати наступні можливості: добавляти/видаляти нового власника та відповідно новий автомобіль для заданого власника, здійснювати аналогічні операції з довіреностями на автомобіль, видавати необхідну довідкову інформацію (наприклад, для заданого автомобіля визначати його власника, тощо), при цьому передбачається, що у кожного автомобіля може бути лише один власник, на один і той же автомобіль може бути зафіксовано кілька довіреностей.

2. Генеалогічне дерево.

Система підтримки генеалогічного дерева повинна надавати наступні можливості: добавляти в дерево нового члена сім'ї (дитину, чоловіка, жінку, ...), вносити зміни в вузли дерева (наприклад, фіксувати зміну прізвища чи дату смерті), здійснювати пошук корисної інформації за деревом (наприклад, для заданого члена сім'ї знаходити його дітей та навпаки).

3. Розклад руху поїздів на станції.

На станції є набір платформ та шляхів, на які можуть прибувати поїзда, відомі номери прибуваючих поїздів, час їх прибуття та відправлення, а також станції відправлення та призначення. Система підтримки розкладу станції повинна забезпечувати можливість формування розкладу, внесення в нього змін та видачу корисної інформації (наприклад, список поїздів до зазначеної станції призначення).

4. Система підтримки складання розкладу занять.

Система повинна забезпечувати можливість складання розкладу деякого навчального закладу, внесення в розклад змін та видачу корисної інформації (наприклад, за розкладом отримати розклад заданої групи на заданий день). В розкладі повинні фіксуватися час та місце проведення заняття, предмет та викладач, що проводить заняття, а також номер групи, для якої це заняття проводиться. Розклад не повинен містити колізій (наприклад, різні заняття не повинні перетинатися за місцем та часом їх проведення).

5. Замовлення/облік товарів в "бакалійній лавці".

В "бакалійній лавці" для кожного товару фіксується місце зберігання (певна полиця), кількість та постачальник цього товару. Система підтримки

замовлення/обліку товарів повинна забезпечувати можливість добавлення/видалення нового товару, зміни інформації про наявний товар (наприклад, при зміні кількості товару тощо) та видачі необхідної довідкової інформації (наприклад, список товарів, кількість яких необхідно поповнити).

6. Керування бібліотекою.

В бібліотеці здійснюється реєстрація всіх читачів та ведуться каталоги книжок, що надійшли до бібліотеки. Крім того фіксується інформація про те, які книжки у якого читача знаходяться на даний момент. Система підтримки керування бібліотекою повинна забезпечувати можливість добавлення/видалення читачів та відповідно книжок в каталогах, реєстрацію взятих та повернутих читачем книжок, а також видавати корисну довідкову інформацію (наприклад, про наявність в даний момент зазначеної книжки).

7. Робота банкомату.

Банкомат - це автомат для видачі готівки по кредитних пластикових картках. У його склад входять наступні пристрої: дисплей, панель керування із кнопками, приймач кредитних карт, сховище грошей і лоток для їхньої видачі, сховище конфіскованих кредитних карт, принтер для печатки довідок, сервісна консоль. Банкомат підключений до лінії зв'язку для обміну даних з банківським комп'ютером, що зберігає відомості про рахунки клієнтів.

Обслуговування клієнта починається з моменту приміщення пластикової картки в банкомат. Після розпізнавання типу пластикової картки, банкомат видає на дисплей запрошення ввести персональний код. Персональний код являє собою чотиризначне число. Потім банкомат перевіряє правильність уведеного коду, звіряючи з кодом, що з на карті. Якщо код зазначений невірно, користувачеві надаються ще дві спроби для введення правильного коду. У випадку повторних невдач карта переміщається в сховище карт, і сеанс обслуговування закінчується. Після введення правильного коду банкомат пропонує користувачеві вибрати операцію. Клієнт може або зняти наявні з рахунку, або довідатися залишок на його рахунку.

8. Турнікет метро.

Турнікет має приймач карт, пристрій для перекривання доступу, таймер, три оптичних датчики для визначення проходу пасажира, пристрій подачі звукових сигналів, індикатори "Прохід" та "Стоп", індикатор кількості поїздок, що залишилися.

У початковому стані турнікета запалений індикатор "Стоп", індикатор "Прохід" погашений. Якщо один з датчиків посилає сигнал, то прохід через турнікет відразу ж перекривається, і подається попереджувальний звуковий сигнал. Для проходу пасажир повинен помістити карту в приймач карт. Кожна

карта має строк придатності, після закінчення якого вона не може бути використана для проходу. Карти бувають двох типів: з фіксованою кількістю поїздок і з необмеженою кількістю поїздок. Турнікет зчитує з карти дані: строк придатності карти, номер карти, тип карти й кількість поїздок. Якщо дані не вдається вважати, або карта прострочена, або кількість поїздок нульове, то карта повертається пасажирові, і турнікет залишається у вихідному стані. Інакше з карти з фіксованою кількістю поїздок списується одна поїздка, карта повертається із приймача, індикатор "Стоп" гасне, запалюється індикатор "Прохід", індикатор кількості поїздок, що залишилися, висвічує поточне значення й пасажир може пройти через турнікет. Одержавши від одного з датчиків сигнал, турнікет очікує час, відведений на прохід пасажира (5 секунд), після чого він повертається в початковий стан. Якщо карта має необмежену кількість поїздок, то її номер запам'ятовується, щоб протягом п'яти хвилин після проходу пасажира із цією картою блокувати спроби проходу з нею через всі турнікети даної станції метро.

Турнікет заносить у свою пам'ять час всіх проходів. Наприкінці робочого дня він передає всю інформацію, накопичену за день, в АСУ метрополітену.

ОСНОВНИЙ СПИСОК ЛІТЕРАТУРИ

1. J.M. Spivey. Understanding Z: A specification language and its formal semantics. – Cambridge University press. – 1988. – 131 p.
2. J.M. Spivey. The Z Notation: A Reference Manual. – Oriel College, Oxford, OX1 4EW, England. – 1998. – 158 p.
3. Woodcock J.C.P., Davies J. Using Z: Specification, Refinement and Proof. – Prentice Hall, 1996. – 523 p.
4. Abrial J.R., Schuman S.A., Meyer B. Specification Language Z. – Boston: Massachusetts Computer Associates inc., 1979. – 378 p.
5. Martin. Relating Z and First-Order Logic. – In FM'99: World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 1999. Proceedings, Vol. 2. – LNCS 1709, p. 1266–1280.
6. A Tutorial of ZANS - A Z Animation System, Xiaoping Jia. May 1995. An Approach to Animating Z Specifications. Xiaoping Jia. Proceedings of the 19th Annual IEEE International Computer Software and Application Conference (COMPSAC 1995). August 1995, Dallas, TX. pp.108-113.
7. Никитченко Н.С. Композиційно–номінативний підхід к уточненню поняття програми // Проблеми програмування.– 1999.– N.1.– С. 16–31.
8. Редько В.Н. Основи композиційного програмування // Програмування. – 1979. – № 3. – С.3-13.
9. Омельчук Л.Л. Аксиоматичні системи специфікацій програм над номінативними даними: Дисертація к.ф.-м.н.: 01.05.01. - К., 2007. – 142 с.
10. Нікітченко М.С., Шкільняк С.С. Математична логіка. Додаткові розділи: Навчальний посібник. – Київ.: Видавничо-поліграфічний центр “Київський університет”, 2004. – 77 с.
11. Аналіз існуючих підходів до задання семантики мов специфікацій предметних областей та програмних систем: Звіт про науково-дослідну роботу (проміжний) / Київський національний університет імені Тараса Шевченка. – № ДР 0108Г002463. – Київ: 2008. – 57 с.

ДОДАТКОВИЙ СПИСОК ЛІТЕРАТУРИ

1. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. – MA.: Addison-Wesley Publishing Co., 1999. – 512 p.
2. Буй Д.Б., Шишацька О.В.* UML – сучасна універсальна мова моделювання: історія, специфікація, бібліографія
3. Лаврищева Е.М. Современные методы программирования: возможности и инструменты // Проблеми програмування. Спеціальний випуск: Матеріали

- п'ятої міжнародної науково-практичної конференції з програмування. – 2006. – № 2-3. – С. 60-74.
4. Hoare, C.A.R. *Communicating Sequential Processes*. London: Prentice-Hall International. – 1985.
 5. Milner R.G. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Science*, 1978, 17, p. 348-357.
 6. Joseph Goguen and Grant Malcolm, *Algebraic Semantics of Imperative Programs*. – MIT Press, 1996.
 7. Björner, Dines; Henson, Martin C. (Eds.) *Logics of Specification Languages*, Series: Monographs in Theoretical Computer Science. An EATCS Series 2008, XXII, 624 p. 69 illus., Hardcover.
 8. Jones, C.B. *Systematic Software Development using VDM*. London: Prentice-Hall International. – 1986.
 9. Björner D., Jones C.B. *Formal Specification and Software Development*. – Prentice-Hall, 1982. – 368 p.
 10. Woodcock J.C.P., Davies J. *Using Z: Specification, Refinement and Proof*. – Prentice Hall, 1996. – 523 p.
 11. Abrial J.R., Schuman S.A., Meyer B. *Specification Language Z*. – Boston: Massachusetts Computer Associates inc., 1979. – 378 p.
 12. A. Martin. Relating Z and First-Order Logic. – In FM'99: World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 1999. Proceedings, Vol. 2. – LNCS 1709, p. 1266–1280.
 13. RAISE Language Group. *The RAISE Specification Language*. BCS Practitioner Series. Prentice Hall. 1992. – 397 p.
 14. RAISE Method Group. *The RAISE Development Method*. BCS Practitioner Series. Prentice Hall. 1995. – 493 p.
 15. Brock S., George C.W. *The RAISE Method Manual*. – Technical Report LACOS/CRI/DOC/3. – CRI: Computer Resources International. – 1990. – 475 p.
 16. Björner D., Denvir T., Meiling E., Pedersen j.S. *The RAISE project: fundamental issues and requirements*. – report RAISE/DDCEM/1/V6. – Dansk datamatic Center. – 1985. – 137 p.
 17. Rod M. Burstall, Joseph A. Goguen, *The Semantics of CLEAR, A Specification Language*, Proceedings of the Abstract Software Specifications, 1979, Copenhagen Winter School, p. 292-332, January 22-February 02.
 18. Guttag, J.V. and Horning J.J. *The Algebraic Specification of Abstract Data Types*. *Acta Informatica*, 10, 1978, p. 27-52.

19. J. Guttag et al. The Larch Family of Specification Languages. IEEE Software, Vol. 2, No.5, September 1985, p. 24-36.
20. Guttag J.V., Horning J.J. Larch: Languages and Tools for Formal Specification. – Springer-Verlag, Texts and monographs in Computer Science, 1993. – 259 p.
21. David C. Luckham. Programming with Specifications: An Introduction to ANNA, A Language for Specifying Ada Programs. Texts and Monographs in Computer Science. Springer-Verlag, October, 1990.
22. D. Harel, "Executable Object Modeling with Statecharts", 1997, IEEE COMPUTER, Vol. 30, issue 7, JULY, p. 31-42
23. Laura K. Dillon and Y.S. Ramakrishna. Generating oracles from your favorite temporal specifications. In Proceedings of the International Symposium on Foundations of Software Engineering, 1996.
24. Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, New York, 1991.
25. Shlomit Pinter and Pierre Wolper. A temporal logic for reasoning about partially ordered computations. In Proc. 3rd ACM Symposium on Principles of Distributed Computing, p. 28-37, Vancouver, August 1984.
26. T. Murata Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, Vol. 77, No 4, April, 1989, p. 541-580.
27. George S. Avrunin, James C. Corbett, and Laura K. Dillon. Analyzing partially-implemented real-time systems. In Eighteenth International Conference on Software Engineering, may 1997.
28. Sophie Dupuy, RoZ version 0.3 an environment for the integration of UML and Z, Laboratoire Logiciels, Systemes et Reseaux – IMAG, BP 72 – 38402 Saint-Martin d’Heres Cedex,
<http://www-lsr.imag.fr/Les.Groupe/PFL/RoZ/docRoZ.rtf>
29. Wirsing M. & Broy M., An Analysis of Semantic Models for Algebraic Specifications. In Theoretical Foundations of Programming Methodology, ed. M. Broy & G. Schmidt. Dordrecht, Netherlands: D. Reidel.
30. Френкель А., Бар-Хиллел И. Основания теории множеств. – М.: Мир, 1966. – 556 с.
31. Редько В.Н. Экспликативное программирование: ретроспективы и перспективы // Перша міжнародна науково-практична конференція з програмування УкрПРОГ'98 – Праці. – Україна, Київ, Кібернетичний центр Національної академії наук України. – 1998. – С. 22–41.
32. Никитченко Н.С. Композиционно–номинативный подход к уточнению понятия программы // Проблемы программирования. – 1999. – N.1. – С. 16–31.

33. Nikitchenko N. A Composition Nominative Approach to Program Semantics. – Technical Report IT-TR: 1998-020. – Technical University of Denmark. – 1998. – 103 p.

ЗМІСТ

ЧАСТИНА 1

МОВИ ТА МЕТОДИ СПЕЦИФІКАЦІЇ ПРОГРАМ	3
1.1. Достовірність тестування ПЗ	3
1.2. Методи формальної розробки, мови специфікацій	3
1.3. Досвід використання формальних методів	5
1.4. Використання специфікацій програм на різних етапах життєвого циклу програми	6
1.5. Класифікація методів специфікації програм	8
1.5.1. Транзиційні специфікації	10
1.5.2. Темпоральні (часові) та паралельні специфікації	10
1.5.3. Специфікації абстрактної моделі	10
1.5.4. Алгебраїчні специфікації	11
1.5.5. Аксиоматичні специфікації	11
1.6. Рівні застосування ФМ	11
1.6.1. Формальна специфікація	12
1.6.2. Формальна розробка і перевірка	12
1.6.3. Доведення	12
Контрольні запитання та вправи	13

ЧАСТИНА 2

ОСНОВНІ ПОНЯТТЯ МОВИ Z	15
2.1. Типи даних мови Z	15
2.1.1. Об'єкти та типи	15
2.1.1.1. Множини та множина типів	15
2.1.1.2. Кортежі та декартові добутки	16
2.1.1.3. Відношення та функції	16
2.1.2. Властивості та схеми	17
2.1.2.1. Об'єднання властивостей	17
2.1.2.2. Оформлення та перейменування	18
2.1.2.3. Об'єднання схем	19
2.1.3. Змінні та область їх дії	21
2.2. Огляд синтаксису мови Z	22
2.2.1. Короткий огляд синтаксису мови Z	22
2.2.2. Word та ідентифікатори	27
2.2.3. Символи операцій	27
2.2.4. Специфікації	28
2.2.5. Визначення базових типів	28
2.2.6. Аксиоматичні описи	29

2.2.7. Обмеження	29
2.2.8. Визначення схеми	29
2.2.9. Скорочені визначення	31
2.3. Зсилки схеми	31
2.4. Декларація змінних	31
2.5. Текст схеми	32
2.6. Вирази	32
2.7. Предикати	38
2.8. Схема виразів	42
2.9. Види	45
2.9.1 Видові схеми	45
2.9.2 Видові константи	46
2.10. Математичні засоби	48
2.10.1. Множини	48
2.10.1.1 Способи завдання множин	48
2.10.1.2 Операції над множинами	49
2.10.2. Відношення	51
2.10.3. Функції	56
2.10.4. Числа та обмеження	57
2.10.5. Послідовності	59
2.11. Вільні типи	61
2.12. Δ та Ξ домовленості	62
2.13. Оцінювання Z специфікації	63
2.14. Приклад застосування	64
Контрольні запитання та вправи	68
ВАРІАНТИ ЗАВДАНЬ ДО ЧАСТИНИ 2	70
ОСНОВНИЙ СПИСОК ЛІТЕРАТУРИ	73
ДОДАТКОВИЙ СПИСОК ЛІТЕРАТУРИ	73
ЗМІСТ	77