КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ТЕМПОРАЛЬНІ ЛОГІКИ ТА ЇХ ЗАСТОСУВАННЯ

Виконав студент 4-го курсу
Іван ЄРЬОМЕНКО

_____
(підпис)

Науковий керівник:
доктор фіз.-мат. наук, професор
Степан ШКІЛЬНЯК

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування

«01» червня 2022 р.

протокол № 10

Завідувач кафедри
Микола НІКІТЧЕНКО

Київ – 2022

РЕФЕРАТ

Обсяг роботи 58 сторінок, 5 ілюстрацій, 15 джерел посилань.

ПРЕДИКАТ, МОДЕЛЬ, ЧАС, ТЕМПОРАЛЬНА ЛОГІКА, ЛІНІЙНА ТЕМПОРАЛЬНА ЛОГІКА, ВЕРИФІКАЦІЯ ПРОГРАМ

Об’єктом дослідження роботи є темпоральні логічні формалізми, орієнтовані на вирішення прикладних задач інформатики та програмування.

Метою роботи є вивчення систем темпоральної логіки та можливості їх застосування, демонстрація змістовних прикладів застосування апарату темпоральної логіки на практиці.

Методи розроблення: апарат темпоральної логіки, реляційна модель Кріпке, автомат Бюхі, метод Model checking для верифікації комп’ютерних програм та систем. Інструменти розроблення: мова темпоральної логіки LTL, верифікатор SPIN, мова програмування Promela, інструментальний засіб Converter.

Результати роботи: проаналізована актуальність даної теми та проблеми які вона вирішує, виконано огляд систем темпоральної логіки, описано мови та реляційну семантику таких логік, розглянуто лінійну пропозиційну темпоральну логіку, наведено приклад її застосування для вирішення проблеми взаємного виключення алгоритмом Петерсона. Описано потужний метод верифікації програмних систем Model cheking та верифікатор SPIN, які використовують апарат лінійної темпоральної логіки. Наведено змістовний приклад застосування апарату темпоральної логіки на практиці – верифікація програми автоматної реалізації математичної гри Нім із використанням SPIN.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 5

ВСТУП 6

РОЗДІЛ 1. ТЕМПОРАЛЬНІ ЛОГІКИ 9

1.1 Системи пропозиційної темпоральної логіки 11

1.2 Класифікація системи темпоральної логіки 15

1.2.1 Пропозиційні та першопорядкові темпоральні логіки 15

1.2.2 Глобальні та модульні темпоральні логіки 16

1.2.3 Темпоральні логіки лінійного та розгалуженого часу 17

1.2.4 Дискретні та неперервні темпоральні логіки 17

1.2.5 Темпоральні логіки минулого і майбутнього часу 18

1.3 Пропозиційна модальна логіка 18

РОЗДІЛ 2. ЛІНІЙНА ПРОПОЗИЦІЙНА ТЕМПОРАЛЬНА ЛОГІКА 25

2.1 Синтаксис та семантика ЛПТЛ 25

2.2 Аксіоматична система для ЛПТЛ 26

2.3 Приклади застосування ЛПТЛ 28

2.3.1 Проблема взаємного виключення 29

2.3.2 Алгоритм Петерсона взаємного виключення 30

РОЗДІЛ 3. МЕТОД MODEL CHECKING 33

3.1 Актуальність розробки систем верифікації ПЗ 34

3.2 Моделювання, специфікація, верифікація 35

3.3 Типи перевіряючих вимог 40

РОЗДІЛ 4. ВЕРИФІКАТОР SPIN 41

4.1 Розробка верифікатора SPIN 41

4.2 Promela 42

4.3 Реляційна модель Кріпке та автомат Бюхі 44

ВИСНОВКИ 47

ПЕРЕЛІК ДЖЕРЕЛ 48

ДОДАТОК. Приклад верифікації програми 50

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

SPIN – Simple Promela Interpreter, утиліта для перевірки правильності розподілених програмних моделей.

LTL – linear temporal logic, мова логіки.

TLA – temporal logic of actions, мова специфікацій, заснована на теорії множин, логіці першого порядку та темпоральній логіці.

TLS – transport layer security, протокол який забезпечує безпечну передачу даних між вузлами в Інтернеті.

CTL – computational tree logic, логіка розгалуженого часу.

GIL – Global Interpreter Lock, спосіб синхронизації потоків.

CSP – Communicating sequential processes, формальна мова для опису взаємодій в рівночасних системах.

MP – Modus Pones, коректна, проста форма аргументації.

NASA – National Aeronautics and Space Administration, національне управління з аеронавтики і дослідження космічного простору.

ПТЛ – пропозиційна темпоральна логіка.

ПМЛ – пропозиційна модальна логіка.

ЛПТЛ – лінійна пропозиційна темпоральна логіка.

ПЗ – програмне забезпечення.

ВСТУП

Час є одним з найфундаментальніших понять філософії та фізики, водночас він є одним з найпарадоксальніших явищ. Онтологічний статус часу належить до найбільш загадкових його аспектів. З одного боку, це суб'єктивне та відносне уявлення, засноване на нашому свідомому досвіді послідовних подій, з іншого боку, наша цивілізація, наука та технології базуються на розумінні того, що існує щось на зразок об'єктивного часу. Тисячоліття роздумів людства над суттю часу засвідчують, що ми проникли в деякі з його таємниць, але дуже багато ще непроясненого. Як виник час? Чи існує Всесвіт без часу? Чому реальний час одновимірний і односпрямований? Час дискретний чи неперервний, чи існують мінімальні кванти часу? Чи є час абсолютно необхідною фундаментальною компонентою в будові Всесвіту, чи це лише зручна конструкція для організації наших суб’єктивних відчуттів? Святий Августин на питання, що він має на увазі, коли вимовляє слово “Бог”, сказав, що відповісти на таке це те ж саме, що й відповісти на питання, що таке час: я можу говорити про нього, проте я розгублююсь, коли ви просите визначити, пояснити, що це таке.

Абсолютного часу не існує, що підтверджують сучасні фізичні теорії, найперше спеціальна та загальна теорії відносності. Ваш час, взагалі кажучи, не може бути таким же, як мій час. Наші відчуття кажуть, що майбутнє відрізняється від минулого, що майбутнє дає нам розмаїття можливостей, а минуле пов’язане з єдиною ситуацією та його ніяк не зміниш. Є безперечна однобічність того, як речі розкриваються у часі. Ще ніхто не бачив, як розбите на підлозі яйце, яке впало зі столу, знову збирається воєдино і опиняється ціленьке на столі. Так звідки і чому виникає стріла часу? Закони фізики в цілому симетричні відносно напрямку часу. Певну відповідь на це дає другий закон термодинаміки:

фізичні системи розвиваються в напрямку збільшення ентропії, від менш ймовірних ситуацій до більш ймовірних, від більш впорядкованих до менш впорядкованих. Проте не будемо зараз далі розглядати ці основоположні фізичні та філософські проблеми, зазначимо лише, що час відіграє фундаментальну роль у нашому мисленні, в розвитку різних галузей науки, зокрема, філософії, логіки, фізики, інформатики, штучного інтелекту.

Важливим розділом логіки, яка при дослідженні робить акцент на причинно-наслідкових зв'язках в умовах часу, є темпоральна, або часова логіка. Вона використовується для опису послідовностей явищ та їх взаємозв'язку з часовою шкалою. Темпоральні логічні системи дозволяють формалізувати твердження, істинність яких змінюється з часом.

Перші дослідження в галузі темпоральної логіки відомі ще часів античності (Арістотель, Діодор Кронос), вони продовжувались і в часи середньовіччя (В.Оккам, Ж.Бурідан, Альберт Саксонський). На сучасному етапі темпоральна логіка як окрема галузь науки та підрозділ модальної логіки починає формуватися в середині XX ст. найперше зусиллями Артура Прайора. Дослідження в галузі темпоральної логіки проводили Г.Х. фон Врігт, М.Решер, Х. Укварт та інші.

Тема дослідження та прикладного застосування темпоральної логіки залишається вельми актуальною і на сьогоднішній день. Поняття і методи темпоральної логіки використовуються у філософії для з'ясування загальних питань про час, в лінгвістиці як формалізм для визначення семантики часових виразів у природній мові. Темпоральна логіка дуже гнучка, вона може ефективно використовуватись для аналізу та моделювання різноманітних предметних областей і пов'язаних з поняттям часу аспектів людської діяльності.

Розвиток темпоральної логіки не припиняється і в наш час. Зокрема, поєднуючи можливості традиційної темпоральної логіки та програмно-орієнтованих композиційно-номінативних логік часткових квазіарних предикатів, було запропоновано транзиційні композиційні-номінативні модальні логіки. Такі логіки враховують аспект зміни й розвитку предметних областей, вони описують переходи від одного стану світу до іншого. Найважливішим різновидом цих логік є композиційно-номінативні темпоральні логіки часткових предикатів.

Особливого значення темпоральна логіка набуває у зв'язку зі створенням сучасних інформаційних та програмних систем. Апарат темпоральної логіки застосовується в інформатиці, штучному інтелекті, програмуванні (див., напр., [8–14, 1, 2].). Дуже важливим є використання темпоральної логіки для опису та моделювання складних динамічних систем, адже їх функціонування безпосередньо пов’язане з часом, для формального аналізу, специфікації та верифікації комп’ютерних програм і систем. На базі темпоральної логіки збудовано багато різноманітних систем та мов специфікацій (Temporal Logic, Petri nets, TLA+, TLS, StateCharts, GIL, CSP).

РОЗДІЛ 1
ТЕМПОРАЛЬНІ ЛОГІКИ

Традиційні логіки орієнтовані на опис часово-просторового зрізу світу, тобто ми описуємо один конкретний стан світу. Якщо ж світ мінливий, якщо він змінюється та розвивається, то для його опису доцільно використовувати модальні логіки.

Модальності – це властивості тверджень, які в тому чи іншому аспекті характеризують міру їх істинності чи наше ставлення до них. Можна виділити низку різновидів модальних логік – найперше, це виділяються загальні (буттєві, атлетичні) модальні логіки, темпоральні логіки, а також логіки епістемічні, деонтичні тощо (див., напр., [3, 4].).

Ще в античні часи розглядались твердження, які можна охарактеризувати певною мірою істинності, а не тільки трактувати їх як лише істинні чи лише хибні. Такими дослідженнями займався Арістотель, він вивчав тверджень вигляду "кожне S необхідно P", "кожне S можливо P" тощо.

Модальності "необхідно" й "можливо" – це загальні або алетичні модальності. Їх зазвичай позначають □ та ◊.

Модальності, які мають часовий зміст, називають часовими, або темпоральними. Часові модальності, як і атлетичні, відомі з античних часів, їх вивчали Арістотель, Діодор Кронос, філософи-стоїки.

Основними, базовими часовими модальностями є такі: "завжди було", "колись було", "завжди буде", "колись буде".

Міркування, які містять часові модальності, є предметом вивчення часової, або темпоральної логіки. Темпоральна логіка досліджує природу, характерні ознаки, логічні зв'язки тверджень із часовими модальностями.

Зауважимо, що в темпоральній логіці можна виділити два підходи до терміну "після цього":– причинно-наслідковий та хронологічний:

- "після цього" означає "внаслідок цього";
- "після цього" означає "пізніше" в сенсі часу.

Розглянемо твердження: "Я веселий": Сам зміст цього виразу не змінюється з часом, проте його істинність може змінитися. В традиційній логіці конкретне твердження в конкретний момент часу може бути істинним, або хибним, але це не може бути одночасно. В такій логіці значення тверджень не змінюються з часом. Водночас в темпоральній логіці значення твердження залежить від того, коли воно перевіряється. Темпоральна логіка дозволяє висловити затвердження типу "Я завжди буду веселий", "Я іноді був веселий", "Я веселий, поки я не стомлюсь" тощо.

Далі розглядаємо темпоральні логіки на пропозиційному рівні.

Виклад матеріалу в розділі ведемо на основі робіт [5, 6].

1.1 Системи пропозиційної темпоральної логіки

Як було зазначено вище, базовими модальностями темпоральної логіки є такі: "завжди було", "колись було", "завжди буде", "колись буде".

Відповідно до основних часових модальностей введемо такі модальні композиції (модальні оператори) темпоральної логіки: $\Box_{\uparrow}$ (завжди буде), с (колись буде) та $\Diamond_{\downarrow}$ (колись було).

Модальні оператори $\Box_{\uparrow}$, $\Box_{\downarrow}$, $\Diamond_{\uparrow}$, $\Diamond_{\downarrow}$ в літературі часто традиційно позначають (див. [3, 4]) відповідно як G H, F, P.

Розглянемо для прикладу висловлювання "Якщо я її колись бачив, тоді я її упізнаю при зустрічі". Його можна подати так:

$\Box_{\uparrow}$ ($\Diamond_{\downarrow}$ (X бачив Y) => $\Box_{\uparrow}$ (X зустрів Y => X упізнав Y))

Базові часові оператори $\Box_{\uparrow}$, $\Box_{\downarrow}$, $\Diamond_{\uparrow}$, $\Diamond_{\downarrow}$ пов'язані співвідношеннями:

$\neg\Diamond_{\uparrow}P = \Box_{\uparrow}\neg P$;

$\neg\Box_{\uparrow}P = \Diamond_{\uparrow}\neg P$;

$\neg\Diamond_{\downarrow}P = \Box_{\downarrow}\neg P$;

$\neg\Box_{\downarrow}P = \Diamond_{\downarrow}\neg P$.

Таким чином, для систем темпоральної логіки можна вважати базовими лише оператори $\Box_{\uparrow}$ та $\Box_{\downarrow}$. Тоді оператори $\Diamond_{\uparrow}$ та $\Diamond_{\downarrow}$ будуть похідними темпоральними операторами, вони визначаються так:

$\Diamond_{\uparrow}P$ означає $\neg\Box_{\uparrow}\neg P$,

$\Diamond_{\downarrow}P$ означає $\neg\Box_{\downarrow}\neg P$.

Опмшемо аксіоматичні системи пропозиційної темпоральної логіки. Мова таких систем є розширенням мови пропозиційної логіки.

Алфавіт мови:

– множина Ps предикатних символів:

– символи пропозиційних композицій $\neg$, $\vee$;

– символи $\Box_{\uparrow}$, $\Box_{\downarrow}$ відповідних темпоральних операторів.

Індуктивно визначається множина формул Fm мови:

1) Кожний $P \in Ps$ є формулою: такі формули назвемо атомарними;.

2) Нехай Ф та Ψ – формули, тоді $\neg\Phi$, $\vee\Phi\Psi$, $\Box_{\uparrow}\Phi$ $\Box_{\downarrow}\Phi$ – формули.

Символи $\Diamond_{\uparrow}$ та $\Diamond_{\downarrow}$ вважаємо скороченнями для $\neg\Box_{\uparrow}\neg$ та $\neg\Box_{\downarrow}\neg$.

Мінімальне темпоральне числення є позначають K_t.

Таке числення – це аналог алетичної системи К.

Аксіомами системи K_t є аксіоми пропозиційної логіки та аксіоми, що задаються такими схемами:

AxNr$_{\uparrow}$) $\Box_{\uparrow}(\Phi\rightarrow\Psi)\rightarrow(\Box_{\uparrow}\Phi\rightarrow\Box_{\uparrow}\Psi)$;

AxNr$_{\downarrow}$) $\Box_{\downarrow}(\Phi\rightarrow\Psi)\rightarrow(\Box_{\downarrow}\Phi\rightarrow\Box_{\downarrow}\Psi)$;

AxT$_{\uparrow}$) $\Phi\rightarrow\Box_{\uparrow}\Diamond_{\downarrow}\Phi$;

AxT$_{\downarrow}$) $\Phi\rightarrow\Box_{\downarrow}\Diamond_{\uparrow}\Phi$.

Дві перші схеми – це стандартні модальні аксіоми для $\Box_{\uparrow}$ та $\Box_{\downarrow}$.

Аксіоми AxT$_{\uparrow}$ та AxT$_{\downarrow}$ відображають відомі ще з середніх віків принципи змішування часів.

Правила виведення темпорального числення складаються із правил виведення пропозиційної логіки, до яких додаються правила модалізації для темпоральних операторів $\Box_{\uparrow}$ та $\Box_{\downarrow}$:

ПМ$_{\uparrow}$) $\Phi \vdash \Box_{\uparrow}\Phi$;

ПМ$_{\downarrow}$) $\Phi \vdash \Box_{\downarrow}\Phi$.

Для темпорального числення T_t множина аксіом складається з аксіом числення K_t та аксіом, що задаються схемами

Ax$\Box_{\uparrow}$) $\Box_{\uparrow}\Phi\rightarrow\Phi$;

Ax$\Box_{\downarrow}$) $\Box_{\downarrow}\Phi\rightarrow\Phi$.

Темпоральне числення T_t – це аналог алетичної системи Т.

Темпоральне числення B_t є аналогом алетичної системи B.

Множина аксіом B_t складається з аксіом числення T_t та аксіом, що задаються схемами

AxB↑) $\Phi \to \Box_{\uparrow} \Diamond_{\uparrow} \Phi$;

AxB↓) $\Phi \to \Box_{\downarrow} \Diamond_{\downarrow} \Phi$.

Темпоральне числення $S4_t$ є аналогом алетичної системи S4.

Множина його аксіом складається з аксіом числення T_t та аксіом, що задаються схемами

AxS4↑) $\Box_{\uparrow} \Phi \to \Box_{\uparrow} \Box_{\uparrow} \Phi$;

AxS4↓) $\Box_{\downarrow} \Phi \to \Box_{\downarrow} \Box_{\downarrow} \Phi$.

Аналогом алетичної системи S5 є темпоральне числення $S5_t$.

Множина аксіом $S5_t$ складається з аксіом числення T_t та аксіом, що задаються такими схемами

AxS5↑) $\Diamond_{\uparrow} \Phi \to \Box_{\uparrow} \Diamond_{\uparrow} \Phi$;

AxS5↓) $\Diamond_{\downarrow} \Phi \to \Box_{\downarrow} \Diamond_{\downarrow} \Phi$.

Правила виведення темпоральних числень T_t, B_t, $S4_t$, $S5_t$ ті самі, що й для числення K_t.

Задамо реляційну семантику пропозиційної темпоральної логіки. Вона задається так, як і для пропозиційної алетичної модальної логіки.

Введемо поняття темпоральної моделі можливих світів.

Темпоральною моделлю можливих світів, або реляційною темпоральною моделлю, назвемо трійку $\mathbf{M} = (S, \triangleright, I)$.

Тут S – множина світів (станів світу), $\triangleright$ – бінарне відношення на S, I – це відображення $I : Ps \times S \to \{T, F\}$ інтерпретації атомарних формул на світах.

Відношення досяжності на світах $\triangleright$ фактично вказує напрямок часу – від минулого до майбутнього.

Відображення $I : Ps \times S \to \{T, F\}$ індуктивно продовжимо до відображення інтерпретації формул на світах $I : Fm \times S \to \{T, F\}$:

– $I(\neg\Phi, \alpha) = \neg(I(\Phi, \alpha))$;

– $I(\vee\Phi\Psi, \alpha) = \vee(I(\Phi, \alpha), I(\Phi, \alpha))$;

– $J(\Box_{\uparrow}\Phi, \alpha) = T \Leftrightarrow J(\Phi, \beta) = T$ для всіх $\beta \in S$ таких, що $\alpha \triangleright \beta$, інакше $J(\Box_{\uparrow}\Phi, \alpha) = F$;

– $J(\Box_{\downarrow}\Phi, \alpha) = T \Leftrightarrow J(\Phi, \beta) = T$ для всіх $\beta \in S$ таких, що $\beta \triangleright \alpha$, інакше $J(\Box_{\downarrow}\Phi, \alpha) = F$.

Формула Φ істинна в моделі **M** (позначаємо $\mathbf{M} \models \Phi$), якщо

$I(\Phi, \alpha) = T$ для всіх $\alpha \in \mathbf{S}$.

Практично важливим різновидом пропозиційної темпоральної логіки є лінійні темпоральні логіки.

Традиційними модальними операторами лінійної темпоральної логіки є відомі оператори $\Box$ та $\Diamond$, до яких додають оператор 0.

Тут 0P означає "у наступний момент P".

Особливе зацікавлення виявляють такі розширення мінімальної темпоральної логіки, моделі яких мають ті чи інші властивості фізичного часу (лінійність чи розгалуженість, дискретність чи континуальність, скінченність чи нескінченність, щільність, циклічність і т. п.) , див. про це в [3, 4].

Вкрай важливими є дослідження, присвячені метричним темпоральним логікам.

Мінімальне метричне темпоральне числення запропоновано А. Прайором. Основні оператори метричного темпорального числення:

– $\uparrow$n – "буде через n одиниць часу",

– $\downarrow$n – "було n одиниць часу тому".

Неметричні темпоральні оператори $\Box_{\uparrow}$, $\Box_{\downarrow}$, $\Diamond_{\uparrow}$, $\Diamond_{\downarrow}$ можна виразити через оператори $\uparrow n$ та $\downarrow n$:

$$\Box_{\uparrow} P = \forall n\,(\uparrow n\, P),$$

$$\Diamond_{\uparrow} P = \exists n\,(\uparrow n\, P),$$

$$\Box_{\downarrow} P = \forall n\,(\downarrow n\, P),$$

$$\Diamond_{\downarrow} P = \exists n\,(\downarrow n\, P).$$

Д. Кліффорд запрпонував узагальнення метричної темпоральної логіки шляхом введення потоків часу та відповідних часових операторів $\tau\uparrow n$ і $\tau\downarrow n$. Вони відповідно означають "у потоці τ через n одиниць часу буде істинним" і "у потоці τ n одиниць часу тому було істинним", Д.Кліффорд далі описав відповідні темпоральні числення.

1.2 Класифікація системи темпоральної логіки

Розглянемо тепер класифікацію систем темпоральної логіки, яка вважається загальноприйнятою.

1.2.1 Пропозиційні та першопорядкові темпоральні логіки

Немодальною частиною пропозиційної темпоральної логіки (ПТЛ) є класична пропозиційна логіка. Це означає, що формули мови ПТЛ будуються з атомарних за допомогою символів логічних зв’язок та темпоральних операторів.

Вирази першопорядкової темпоральної логіки будуються із атомарних за допомогою символів змінних, констант, функцій, предикатів та кванторів.

Розрізняють неінтерпретовану (де не робляться припущення щодо властивостей структур, що розглядаються) та інтерпретовану

темпоральної логіки першого порядку, якій приписуються певну структуру.

В цілком інтерпретованій темпоральній логіці першого порядку кожна змінна має свою область визначення, для кожного функціонального символу є своя конкретна функція над областю визначення тощо, тоді як в частково інтерпретованій темпоральній логіці не всі функціональні символи та змінні можуть отримати інтерпретацію.

Також часто виділяють локальні змінні, які згідно семантики приймають різні значення на різних станах, і глобальні змінні, значення яких зберігається над усіма станами.

Можна накладати ті чи інші синтаксичні обмеження на взаємодію кванторів і темпоральних операторів. Необмежений синтаксис дозволяє, наприклад, модальний оператор в області дії квантора. Такі логіки без обмежень будуть нерозв’язні. З іншого боку, можна заборонити квантифікацію над темпоральними операторами і отримати обмежену темпоральну логіку першого порядку.

1.2.2 Глобальні та модульні темпоральні логіки

В ендогенній (внутрішній) темпоральній логіці всі темпоральні змінні інтерпретуються в одному світі, що відповідає одній конкурентній програмі. Такі темпоральні логіки підходять для глобального доведення над цілою конкурентною програмою.

В екзогенній (зовнішній) темпоральній логіці синтаксис темпоральних операторів дозволяє виражати властивості коректності відносно декількох різних програм (або їхніх фрагментів) в одній формулі. Ці логіки спрощують модульне доведення програм: можна проводити верифікацію цілої програми, специфікуючи та верифікуючи її складові

підпрограми з подальшим об’єднанням їх в одну програму. Для доведення коректності доведення підпрограм використовуються як леми.

1.2.3 Темпоральні логіки лінійного та розгалуженого часу

Описуючи систему темпоральної логіки, можна по-різному розглядати природу часу. Перша ідея: час має лінійну семантику, тобто в кожен момент часу існує єдиний можливий наступний момент. Отримуємо систему темпоральної логіки з лінійним часом.

Інший спосіб – час з розгалуженою семантикою. Він має деревовидну структуру: в кожен його момент час може розділитися на альтернативні потоки, представляючи різні варіанти майбутнього. Так отримуємо системи темпоральної логіки з розгалуженим часом.

Відповідно цьому темпоральні модальності або описують події вздовж єдиної лінії часу, або відбивають розгалужену природу часу, дозволяючи квантифікацію над можливими майбутніми. Обидва підходи використовуються для доведення програм.

1.2.4 Дискретні та неперервні темпоральні логіки

Найчастіше моделі темпоральної логіки для доведення коректності програм базуються на темпоральних операторах, які отримують істиннісні значення в дискретних точках часу. Водночас в деяких моделях (наприклад, інтервальна темпоральна логіка) значення темпоральних операторів оцінюються над певними інтервалами часу.

В переважній більшості темпоральних логік, які використовуються для доведення коректності програм, час є дискретним. Це означає, що теперішній момент відповідає поточному стану програми, а наступний – його безпосередньому стану-наступнику. Таким чином, темпоральною

структурою – відповідником виконання програми, тобто послідовності станів, є невід’ємні цілі числа.

На неперервній часовій структурі стани темпоральної логіки інтерпретується як дійсні або раціональні числа. Останній тип логік може застосовуватися в програмах реального часу з накладеними строгими кількісними вимогами.

1.2.5 Темпоральні логіки минулого і майбутнього часу

Темпоральні модальності з великим успіхом можуть служити для опису подій як в минулому, так і в майбутньому часі.

В більшості темпоральних логік для доведень над конкурентними системами присутні тільки оператори майбутнього. Цього достатньо, оскільки виконання програми, як правило, починається у визначений момент часу, і може бути показано, що включення операторів минулого не додасть виразної сили.

Водночас для модульних специфікацій оператори минулого часу теж важливі, оскільки потрібно враховувати і минулі події.

1.3 Пропозиційна модальна логіка

В цьому підрозділі детальніше розглянемо синтаксис і семантику пропозиційної модальної логіки (ПМЛ).

Формули ПМЛ будуються за допомогою тих самих правил, які застосовувались до побудови пропозиційної логіки. Але мова ПМЛ додатково використовує два нових символи модальних операторів □ і ◊,. Обидва ці оператори є унарними.

Пропозиційні символи позначають атомарні висловлювання.

Використаємо розширений набір логічних зв’язок $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.

Формули ПМЛ дістаємо за таким індуктивним означенням.

Означення 1 (формули ПМЛ).

1) Всі пропозиційні символи атомарні є формулами ПМЛ.
2) Якщо A і B є формулами ПМЛ, то $\neg A$, $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \leftrightarrow B$, $\Box A$, $\Diamond A$ теж є формулами ПМЛ.

Як зазначено вище, оператори $\Box$ і $\Diamond$ пов'язані між собою. Це дає змогу подати означення оператор $\Diamond$ за допомогою оператора $\Box$:

$\Diamond$ означає $\neg\Box\uparrow\neg$.

Отже, кожну формулу ПМЛ можна записати, користуючись лише символом $\Box$, але для зручності використовують обидва оператори.

Оператори $\Box$ і $\Diamond$ можуть трактуватися дещо різними способами.

Ось деякі зі способів читання цих операторів.

$\Box$A означає:

Необхідно, щоб A була істинна (традиційне трактування).

Завжди A буде істинна.

Вимагається, щоб A була істинна.

Відомо, що A істинна.

Результат довільного виконання програми задовольняє A.

$\Diamond$A означає:

Можливо, що A істинна.

Інколи A буде істинна.

Дозволяється, щоб A була істинна.

Невідомо супротивне до A.

Є таке виконання програми, що результат задовольняє A.

Опишемо тепер реляційну семантику формул ПМЛ Для цього розглянемо поняття реляційної структури.

Означення 2. Реляційною структурою називається пара (W, R), де W – деяка непуста множина, а $R \subseteq W \times W$ – бінарне відношення в множині W.

Елементи множини W – це змістовно світи чи стани світу, далі називаємо їх точками.

Означення 3. Нехай P є множиною атомарних формул ПМЛ.

Реляційною P-моделлю на структурі $F = (W, R)$ називається трійка $M = (W, R, f)$, де $f : P \to B(W)$ − функція із множини формул P в булеан множини W.

Для $p \in P$ множину f(p) можна неформально інтерпретувати як множину точок з W, в яких висловлювання p є істинним.

Якщо P є фіксованою множиною формул, то символ P в означенні моделі опускається, і в цьому разі говоримо просто про модель.

Нехай $w \in W$ і A є формулою ПМЛ. Вираз $M \models_w A$ означає, що формула A є істинна в точці w моделі M = (W,R,f).

Означення 4. ПМЛ-формула A виконувана тоді і тільки тоді, коли існує модель M і деяка точка w цієї моделі така, що $M \models_w A$.

Означення 5(семантика формул ПМЛ). Семантика формул ПМЛ визначається на моделі $M = (W, R, f)$ таким чином:

1) $M \models_w A$, якщо $A \in P$ і $w \in f(A)$,
2) $\models_w A \to B$, якщо з $M \models_w A$ випливає $M \models_w B$,
3) $M \models_w \Box A$, якщо для кожного $t \in W$ такого, що wRt, маємо $M \models_t A$.

Із останнього випливає, що формула $\Box A$ має бути істинна в усіх точках t моделі M, для яких відношення wRt істинне.

Правила 1-4 називаються базовими правилами. За їх допомогою можна знайти значення формул: 1, $\neg$, $A \wedge B$, $A \vee B$, $A \leftrightarrow B$ і $\Diamond A$.

Наприклад, логічну сталу 1 (*T* в інших позначеннях) можна записати як $0 \rightarrow A$, а формулу $\neg A$ – як $A \rightarrow 0$.

З правила $\neg A = A \rightarrow 0$ випливає таке семантичне правило:

1) $M \models_w \neg A$, якщо $M \not\models_w A$,
2) $M \models_w \Diamond A$, якщо $M \models_t A$ принаймні для одного $t \in W$ такого, що wRt,
3) $M \models_w A \vee B$, якщо $M \models_w A$ або $M \models_w B$,
4) $M \models_w A \wedge B$, якщо $M \models_w A$ і $M \models_w B$.

Означення 6. Формула А називається істинною в моделі М, якщо вона істинна в кожній точці М, тобто $M \models_w A$ для всіх $w \in W$.

Записується це як $M \models A$.

Формула А називається істинною в структурі $F = (W,R)$, якщо вона істинна в кожній моделі $M = (W,R,f)$, тобто $\forall M = (W,R,f)$ маємо $M \models A$.

Формула А називається модальною тавтологією, якщо вона істинна в усіх структурах (W, R). Записується це так: $\models A$.

Теорема 1. Наведені, нижче формули є модальними тавтологіями:

a) $\Box A \leftrightarrow \neg(\Diamond(\neg A))$;

b) $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$;

c) $\Diamond(A \rightarrow B) \rightarrow (\Diamond A \rightarrow \Diamond B)$.

Слід зауважити, що з пункту а) цього твердження випливає справедливість означення $\Diamond A = \neg(\Box(\neg A))$, яке було назване правилом двоїстості для модальних операторів $\Box$ і $\Diamond$.

Описана семантика формул ПМЛ називається реляційною семантикою, або семантикою можливих світів, або семантикою Кріпке, за іменем автора, який вперше ввів її до розгляду для модальних логік..

Продемонструємо на прикладі, як показати істинність чи хибність деякої формули ПМЛ в реляційній моделі Кріпке.

Приклад. Нехай дана формула $\Box A \vee \Box B$ і модель Кріпке, яка показана на рис. 1.1.

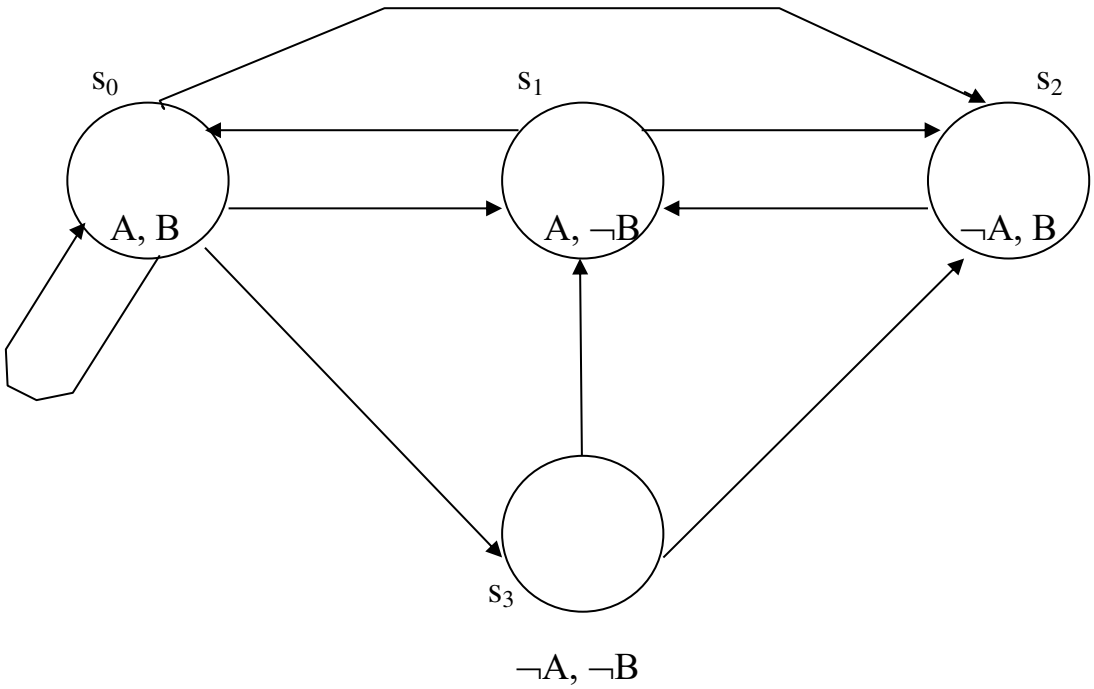

Рисунок 1.1 – Модель Кріпке для $\Box A \vee \Box B$

Для цієї моделі маємо:

$W = \{s_0, s_1, s_2, s_3\}$,

$R = \{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_0, s_3), (s_1, s_0), (s_1, s_2), (s_2, s_1), (s_3, s_2), (s_3, s_1)\}$.

Позначки станів показують, що в

1) s_0 істинні формули A, B;
2) s_1 істинні формули A, $\neg$ B;
3) s_2 істинні формули $\neg$ A, B;
4) s_3 істинні формули $\neg$ A, $\neg$ B.

За цієї інтерпретації формула $\Box A \vee \Box B$ є

a) хибна в стані s_0, оскільки стани s_0, s_1, s_2, s_3 є досяжними із стану s_0 і в стані s_3 формули A і B хибні;

b) істинна в стані s_1, оскільки стани s_0 і s_2 є досяжними із стану s_1 і формула B є істинна в станах s_0 і s_2;

c) істинна в стані s_2, оскільки стан s_1 є досяжним із стану s_2 і в стані s_1 істинна формула A;

d) істинна в стані s_3, оскільки стани s_1, s_2 досяжні із стану s_3 і в стані s_1 істинна формула A, а в стані s_2 істинна формула B.

Розглянемо аксіоматичнуа систему для ПМЛ.

ПМЛ можна трактувати як розширенням класичної пропозиційної логіки. Формальна аксіоматична система (числення) для ПМЛ будується за тими самим правилами, що і для пропозиційної логіки.

Розглядаємо мову ПМЛ із символами пропозиційних зв'язок $\neg$ та $\rightarrow$, символом модального оператора $\Box$.

Це дає індуктивне визначення формул ПМЛ таке:

Якщо A і B є формулами, то $\neg A$, $A \rightarrow B$, $\Box A$ є формулами.

Схемами аксіом аксіоматичної теорії для ПМЛ є такі формули:

1) $A \rightarrow (B \rightarrow A)$,

2) $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$,

3) $(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$,

4) $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

Правилами виведення є:

modus ponens (MP): з A і $A \rightarrow B$ випливає B,

правило модальної необхідності (RN): з A випливає $\Box A$.

Інші логічні зв'язкм вводяться за допомогою відомих тотожностей:

$A \wedge B \leftrightarrow \neg(A \rightarrow \neg B)$,

$A \vee B \leftrightarrow \neg A \rightarrow B$,

$$(A \leftrightarrow B) \leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A).$$

Означення 7. Доведенням (виведенням) в системі ПМЛ називається довільна послідовність формул і A_1, A_2, …, A_n така, що кожна формула A_i є або аксіомою, або безпосереднім наслідком попередніх формул цієї послідовності за одним із правил виведення.

Формула А називається теоремою ПМЛ, якщо існує доведення в ПМЛ для А таке, що останнім елементом у ньому є формула А.

Таке доведення називається доведенням формули А в теорії ПМЛ.

Аксіоматична система ПМЛ, яка включає аксіому К, називається нормальною модальною системою.

РОЗДІЛ 2
ЛІНІЙНА ПРОПОЗИЦІЙНА ТЕМПОРАЛЬНА ЛОГІКА

В базовій темпоральній логіці на відношення R не накладалося жодних обмежень. У практичних застосуваннях відношення R часто є відношенням часткового або лінійного порядку. Якщо відношення R є відношенням лінійного порядку, то одержуємо логіку, яка називається лінійною пропозиційною темпоральною логікою (ЛПТЛ). Відношення лінійного порядку R має таку властивість: для кожного елемента w ϵ W існує єдиний елемент w' ϵ W, такий, що (w,w') ϵ R. Ця властивість дає можливість ввести ще один модальний оператор, який позначається як ◯ і називається X-оператором або next-оператором. Неформально ◯A означає, що формула A істинна в наступній точці моделі, яка йде безпосередньо за даною точкою згідно з відношенням R.

Розглянемо синтаксис і семантику цієї логіки, а також аксіоматичну систему для неї.

2.1 Синтаксис та семантика ЛПТЛ

Нехай Al = {A, B, C, ...} − алфавіт атомарних формул, а P − множина пропозиційних формул над цим алфавітом.

Означення 8 (синтаксис ЛПТЛ).

1) Всі атомарні формули є формулами ЛПТЛ.
2) Якщо A і B є формулами ЛПТЛ, то ¬A, A→B, □A, ◊A◯A теж є формулами ЛПТЛ.

Решта логічних зв'язок вводяться за допомогою таких тотожностей:

$$A \wedge B \leftrightarrow \neg(A \rightarrow \neg B),$$

$$A \vee B \leftrightarrow \neg A \rightarrow B,$$

$(A \leftrightarrow B) \leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$,

$\Diamond A \leftrightarrow \neg(\Box(\neg A))$

Реляційна модель для ЛПТЛ має вигляд $M = (T, R, f, t_0)$, де множину точок позначають T (а не W) і називають множиною моментів часу, $t_0 \in T$ - початковий момент часу, $R \subseteq T \times T$ − бінарне відношення лінійного порядку на T, а для кожного $p \in P$ $f(p)$ є підмножиною множини T тих моментів часу, в які формула p істинна.

Семантика формул ЛПТЛ визначається на моделі M таким чином:

1) $M \models p$ тоді і тільки тоді, коли $t_0 \in f(p)$, де $p \in P$,
2) $M \not\models 0$,
3) $M \models (\varphi \rightarrow \omega)$ тоді і тільки тоді, коли з $M \models \varphi$ випливає $M \models \omega$,
4) $M \models \Box \varphi$ тоді і тільки тоді, коли $\forall t_1 \in T$ такого, що $(t_0, t_1) \in R$ має місце $t_1 \models \varphi$,
5) $M \models \bigcirc \varphi$ тоді і тільки тоді, коли $\exists t_1 \in T$ таке, що $(t_0, t_1) \in R$ (тобто t_1 безпосередньо йде за t_0 і місце $t_1 \models \varphi$).

2.2 Аксіоматична система для ЛПТЛ

Для довільних формул A, B, C наступні формули є схемами аксіом:

A1) $A \rightarrow (B \rightarrow A)$,

A2) $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow A \rightarrow C))$,

A3) $(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$,

K) $\Box (A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$,

T1) $\Diamond A \leftrightarrow \neg(\Box(\neg A))$,

T2) $\bigcirc (A \rightarrow B) \rightarrow (\bigcirc A \rightarrow \bigcirc B)$,

T3) $\Box A \rightarrow (A \wedge \bigcirc A \wedge \bigcirc(\Box A))$,

T4) $\Box (A \rightarrow \bigcirc A) \rightarrow (A \rightarrow \Box A)$,

$T5) \bigcirc A \rightarrow \neg(\bigcirc(\neg A))$

Правилами виведення є:

modus ponens (MP): із A та $A \rightarrow B$ випливає B,

правило модальної необхідності (RN): з A випливає $\Box A$.

Аксіома T4) у вищенаведеній системі є аксіомою математичної індукції.

Кроком індуктивного доведення є $A \rightarrow \bigcirc A$, тобто припускаємо, що A істинна "сьогодні", і доводимо, що A буде істинною "завтра". Якщо індуктивний крок завжди має місце $\Box(A \rightarrow \bigcirc A)$, то звідси випливає, що $A \rightarrow \Box A$.

Дана аксіоматична система є повною і несуперечною. Зв'язок між синтаксисом і семантикою тверджень ЛПТЛ випливає з такої теореми.

Теорема 2. Формула A є теоремою ЛПТЛ тоді і тільки тоді, коли A є тавтологією ЛПТЛ .

Розглянемо деякі властивості ЛПТЛ.

Теорема 3. Для довільної ЛПТЛ-формули A мають місце такі твердження:

a) $\Box A \rightarrow A$,

b) $\Box A \rightarrow \bigcirc A$,

c) $A \rightarrow \Diamond A$,

d) $\bigcirc A \rightarrow \Diamond A$,

e) $\Box A \rightarrow \Diamond A$,

f) $\neg(\bigcirc A) \leftrightarrow \bigcirc(\neg A)$.

Теорема 4. Для довільних ЛПТЛ-формул A і B маємо:

a) $(A \rightarrow \bigcirc A) \vdash (A \rightarrow \Box A)$,

b) $(A \rightarrow B) \vdash (\Box A \rightarrow \Box B)$,

c) $(A \rightarrow B) \vdash (\bigcirc A \rightarrow \bigcirc B)$.

Теорема 5. Для довільних ЛПТЛ-формул A і B маємо:

a) $\Box\,(\bigcirc A\,) \leftrightarrow \bigcirc(\Box A)$,

b) $\Box A \leftrightarrow (A \wedge \bigcirc(\Box A))$,

c) $\Diamond A \leftrightarrow (A \vee \bigcirc(\Diamond A))$,

d) $\Diamond(\bigcirc A\,) \leftrightarrow \bigcirc(\Diamond A)$,

e) $\bigcirc(A \vee B) \leftrightarrow (\bigcirc A \vee \bigcirc B)$,

f) $\bigcirc(A \wedge B) \leftrightarrow (\bigcirc A \wedge \bigcirc B)$,

g) $\Box\,(A \wedge B) \leftrightarrow (\Box A \wedge \Box B)$,

h) $\Box\,(A \vee B) \rightarrow (\Box A \vee \Box B)$,

i) $\Box\,(\Box A\,) \leftrightarrow \Box A$,

j) $\Diamond(A \vee B) \leftrightarrow (\Diamond A \vee \Diamond B)$,

k) $\Diamond(A \wedge B) \rightarrow (\Diamond A \wedge \Diamond B)$,

l) $\Box\,(A \rightarrow B) \rightarrow (\Diamond A \rightarrow \Diamond B)$,

m) $\Box\,(\Diamond(\Box A\,)\,) \leftrightarrow \Diamond(\Box A)$,

n) $\Diamond(\Box\,(\Diamond A\,)\,) \leftrightarrow \Box\,(\Diamond A)$,

o) $\Diamond(\Box A\,) \rightarrow \Box\,(\Diamond A)$.

Дана теорема є основою, на якій ґрунтується метод семантичного табло для ЛПТЛ.

2.3 Приклади застосування ЛПТЛ

ЛПТЛ є формальною логічною мовою, за допомогою якої можна записати і довести деякі властивості систем, що включають паралельні обчислення. Розглянемо один приклад застосування ЛПТЛ для специфікації паралельного алгоритму.

2.3.1 Проблема взаємного виключення

Проблема взаємного виключення часто з'являється в системах паралельних програм. Припустимо, що декілька паралельних процесів потребують доступу до єдиного спільного ресурсу (наприклад, декілька персональних комп'ютерів, що працюють в мережі, потребують доступу до принтера). Цей доступ завжди можливий, коли можливе взаємне виключення: в кожний момент часу тільки один з процесів може використовувати спільний ресурс.

Застосування ЛПТЛ для специфікації подібного типу розподілених обчислень полягає в тому, що за допомогою формул цієї логіки можна виразити такі властивості реальних систем, як mutex (взаємне виключення), fairness (якщо якась подія має відбутися, то наступить такий момент, коли вона відбудеться) і т. д. Наприклад, різниця між формулами $\Box(\Diamond p) \wedge \Box(\Diamond q)$ і $\Diamond(\Box p) \wedge \Diamond(\Box q)$ полягає в тому, що за першою формулою p і q є істинними нескінченно часто і, зокрема, p може бути істинною в ті моменти, коли q є хибною, і навпаки. За другою формулою p колись буде істинною і в кожний такий момент колись буде істинною формула q.

Ця різниця відіграє ключову роль у застосуванні ЛПТЛ під час формалізації властивості fairness:

- Слабка fairness: якщо деякий процес P буде колись активним нескінченно часто, то завжди можливий початок виконання цього процесу

  $\Diamond(\Box act(P)) \rightarrow \Box(\Diamond execute(P))$.

- Сильна fairness: якщо деякий процес P буде активним нескінченно часто, то завжди можливий початок виконання цього процесу

  $(\Box(\Diamond act(P)) \rightarrow \Box(\Diamond execute(P)))$.

Інша важлива властивість розподілених програм, яка називається liveness, теж виражається за допомогою ЛПТЛ:

- liveness: якщо деякий процес розподіленої (паралельної) програми хоче почати працювати, то колись наступить такий момент, в який цей процес буде

  $(\Box want(P) \rightarrow \Diamond work(P))$.

2.3.2 Алгоритм Петерсона взаємного виключення

Алгоритм Петерсона дає можливість розв'язати проблему взаємного виключення в окремому випадку для двох процесів P_1 і P_2, які взаємодіють між собою за допомогою розділених змінних. Тут нас не цікавить те, що відбувається всередині процесів, а тільки взаємодія, яка гарантує взаємне виключення. Цей механізм використовує три розділені змінні: дві з них − Cl і C2 є булевими, а третя змінна last є змінною, що приймає два значення 1 та 2.

Алгоритм Петерсона є симетричним: перехід від процесу P_1 до процесу P_2 і назад реалізується заміною змінної Cl на C2, а також заміною значень 1 на 2 для змінної last. Алгоритм працює таким чином. Розглянемо, наприклад, процес P_1. У початковий момент P_1 працює і не потребує доступу до спільного ресурсу. В цьому випадку говорять, що процес перебуває в некритичній секції.

Якщо спільний ресурс потрібний, то спочатку виконується протокол входу: змінна C1 набуває значення 1, що говорить про те, що процес P_1 потребує спільного ресурсу. Потім змінна last набуває значення 1 і процес P_1 отримує дозвіл на доступ до спільного ресурсу. Цей доступ затримується доти, доки значення змінної last не дорівнюватиме 2 або доки C2 не стане рівним 0. C2 = 0 означає, що процес P_2 останнім

використовував спільний ресурс. Після цього процес P_1 входить до своєї критичної секції і використовує спільний ресурс.

Кожний з процесів, який ввійшов до критичної секції, повинен колись вийти з неї.

Алгоритм Петерсона

(* глобальні змінні P_1, P_2, last *)

var C1, C2: 0..1 : = 0; lastl..2;

Процес P_1:

while true do

begin

a_1: некритична-секція1;

b_1: C1 := 1;

c_1: last := 1;

d_1: repeat until (C2 = 0 ∨ last = 2);

e_1: критична-секція1;

f_1: C1 := 0;

end;

end.

Процес P_2:

while true do

begin

a_2: некритична-секція1;

b_2: C2 := 1;

c_2: last := 1;

d_2: repeat until (C1 = 0 ∨ last = 2);

e_2: критична-секція1;

f_2: C2 := 0;

end;

end.

Отже, алгоритм Петерсона має задовольняти таким двом умовам:

- mutex: тільки один процес − P_1 або P_2 може перебувати в своїй критичній секції;
- liveness: якщо процес P_1 або P_2 хоче ввійти до своєї критичної секції, то він колись ввійде до неї.

Таким чином, правильність алгоритму Петерсона можна описати за допомогою таких формул:

$$\Box(\neg(\, at(e_1) \wedge at(e_2)\,)\,) \qquad \text{(mutex)}$$

$$\Box\,(\, at(b_1) \rightarrow \Diamond at(e_1)\,),\ \Box\,(\, at(b_2) \rightarrow \Diamond at(e_2)\,) \qquad \text{(liveness),}$$

де at(s) означає виконання оператора s.

РОЗДІЛ 3
МЕТОД MODEL CHECKING

Метод "перевірка моделі" (Model Checking) є дуже перспективним методом верифікації для використання в промисловій розробці програмного забезпечення (ПЗ).

Model Checking – це формальна перевірка того, чи є дана формула (темпоральної логіки) істинною на даній моделі розроблюваної програмної системи. Формула описує бажані вимоги до поведінки цієї системи. Цей метод успішно використовується для автоматичної формальної верифікації паралельних систем із скінченним числом станів. Він дозволяє перевірити, чи задовольняє задана модель системи формальним специфікаціям. Важливим етапом при цьому є опис поведінки моделі системи, яку надалі можна специфікувати та верифікувати.

Як моделі в Model Checking зазвичай використовуються моделі реляційного типу. Специфікації задаються на мові формальної логіки, здебільшого використовують темпоральну логіку, яка дозволяє описувати поведінку системи в часі.

Задача Model Checking полягає у перевірці виконання описаних за допомогою темпоральної логіки властивостей системи. Серед них виділяють два основних типи: властивості безпеки (safety properties) і liveness-властивості. Властивості безпеки – це такі властивості системи, за яких небажані події не трапляються. Під liveness-властивостями маються на увазі такі властивості, які показують, як саме має працювати система, що бажані події будуть відбуватися.

3.1 Актуальність розробки систем верифікації ПЗ

Складність промислового програмного забезпечення постійно зростає. Сучасний програмний продукт може містити десятки мільйонів рядків коду. Подумки охопити функціонування таких систем не в змозі ні одна людина навіть при використанні сучасних технологій та методів абстрагування та управління складністю. Як наслідок, разом зі складністю зростає і кількість потенційних помилок в програмах. Тільки в США фінансові втрати від помилок у програмах оцінюються в десятки мільярдів доларів на рік.

На жаль вже сьогодні ми стаємо свідками жахливих наслідків, викликаних запровадженням у життя не перевірених належним чином моделей. Майже щодня у світі відбуваються катастрофи, великі аварії, і все частіше їх причиною стають комп’ютерні системи та програмне забезпечення, не функціонуюче належним чином. У епіцентрі небезпеки – авіація, космос, медицина, технологічні процеси на ядерних, хімічних виробництвах, де навіть невелика помилка у розрахунках може призвести до жахливих наслідків: достатньо великих матеріальних втрат та смерті багатьох людей.

Катастрофи космічних кораблів Шатл "Колумбія" та "Челенджер" є знаковими прикладами, які демонструють важливість верифікації я даних,

01.02.2003 року за 16 хвилин до посадки вибухнув Шатл "Колумбія" (Columbia). Катастрофа сталася при швидкості 20000 км/год, на висоті біля 63 км і катапультуватися астронавти не могли. Загинуло 7 членів екіпажу.

28.01.1986 року космічний корабель "Челенджер" (Challenger) вибухнув під час 25-го запуску на 79-ій секунді після старту. Усі 7 членів екіпажу загинули. Як встановила президентська комісія, катастрофа сталася через недостатню надійність конструкції поєднання сегментів у твердопаливному ракетному прискорювачі.

Ще можна згадати про дві жахливі авіакатастрофи з літаками *Boeing* 737Max в 2018 та 2019 роках. Причиною стали значні дефекти нової системи контролю за польотами MCAS (Maneuvering Characteristics Augmentation System), яка не пройшла належної верифікації.

Особливо важко піддаються аналізу помилки в паралельних, розподілених, багатопоточних програмах, характерних для бортових систем управління технікою. Добре відомо, що навіть у тих випадках, коли функціонування кожної з паралельних взаємодіючих компонент системи абсолютно ясно, людині важко зрозуміти роботу всієї паралельної системи, процеси в якій взаємозалежні. При розробці паралельної програми необхідно контролювати можливі комбінації частково впорядкованих подій, що значно складніше, ніж контроль повністю впорядкованих подій у звичайних програмах. Паралельні системи, які працюють правильно "майже завжди", роками можуть зберігати тонкі помилки, що виявляються в рідкісних, виняткових ситуаціях. Такі помилки, як правило, неможливо знайти тестуванням.

У наші часи, коли людське життя все частіше залежить від надійного функціонування автоматичних систем, проблема гарантії правильності роботи програмних і апаратних компонентів цих систем набуває першорядного значення. Надійність і передбачуваність поведінки таких систем є більш важливими властивостями, ніж продуктивність, модифікованість, переносимість тощо. Верифікація ПЗ, як один з основних методів підвищення якості програмних систем, стає найважливішою областю інформатики.

3.2 Моделювання, специфікація, верифікація

Включення формальної верифікації в процес розробки розподілених програмних систем для підвищення їх надійності стає все більш необхідною. Результатом дослідження є методика проектування

програмних засобів на основі моделей (Model-Driven Engineering) з використанням формальних методів верифікації як одного з етапів проектування.

Дуже перспективним для використання в промисловій розробці ПЗ виявився розроблений в останні роки новий підхід до верифікації програм – "перевірка моделі", або Model Checking.

Model Checking – це формальна перевірка того, чи є дана формула (як правило, формула темпоральної логіки) істинною на даній структурі. Структура подає модель розроблюваної програмної системи. Формула описує бажані вимоги до поведінки цієї системи.

Перш ніж дати більш широке визначенню поняттю Model Checking, слід дати визначення ключовим поняттям, що пов’язані з цим методом, а це: верифікація, специфікація та моделювання.

Отже, власне верифікація – це набір методів для побудови моделей, математичного формулювання вимог до них, та побудова алгоритмів для формальної перевірки виконання цих вимог. Це власне три етапи верифікації: моделювання, специфікації та власне верифікації.

Моделювання (від франц. Modeler – ліпити, формувати.) – це метод дослідження, при якому відбувається заміна конкретного об’єкта досліджень іншим, подібним до нього. Моделювання – один із способів виявлення помилок в системі на ранніх етапах її розробки.

Специфікація [7] – це перелік характерних властивостей системи, на які слід звернути особливу увагу; вона допомагає зрозуміти суть системи, що вона робить, як працює. На основі специфікації проходить верифікація, власне перевірка.

У проблематиці верифікації утворилось два напрямки: аксіоматичне та алгоритмічне. При першому з них розробляється набір аксіом, за допомогою яких може бути описана як сама система, так і її властивості. Основу другого напрямку складає Model Checking. Мета досліджень в цій

області – сформулювати прозору логічну основу для створення автоматичних систем верифікації програм.

Отже, Model Checking – це метод автоматичної формальної верифікації паралельних систем з кінцевим числом станів. Він дозволяє перевірити чи задовольняє задана модель системи формальним специфікаціям.

Основна ідея полягає у моделюванні – опису розробником поведінки моделі системи, яку надалі можна верифікувати та специфікувати. Побудова моделі, як правило, абстрагується від несуттєвих властивостей. Така концепція дає можливість зменшити розмір самої моделі та пришвидшити процес її перевірки. У якості моделі зазвичай використовується модель Кріпке. Специфікації ж задаються в основному на мові формальної логіки. Для специфікації апаратного і програмного забезпечення, як правило, використовують темпоральну логіку – спеціальну мову, що дозволяє описувати поведінку системи в часі.

Якщо властивість моделі не виконується значить існує контрприклад – сценарій, в я кому модель веде себе небажаним чином. Це означає, що модель має бути переглянута, або ж неправильно задані формальні вимоги.

Алгоритми для Model Checking, зазвичай, базуються на повному перегляді станів моделі: для кожного стану перевіряється, чи задовольняє він сформульованим вимогам.

Оскільки модель скінченна, то такі алгоритми теж завершуються за скінченний час.

Задача Model Checking для LTL полягає у перевірці виконання описаних за допомогою LTL властивостей системи.

Серед можливих властивостей системи виділяють два основних типи: властивості безпеки (safety properties) і властивості життєдіяльності (liveness). Під властивостями безпеки маються на увазі такі властивості

системи, за яких небажані події не трапляються. Під властивостями життєдіяльності мається на увазі такі властивості системи, за яких бажана подія не закінчується.

Model Checking можна виконати, використавши розширення скінченного автомату на нескінченні вхідні слова: автомату Бюхі. Для цього слід побудувати два автомати Бюхі: "еквівалентний" моделі та "еквівалентний" запереченню властивостей моделі. Перетин цих двох не детермінованих автоматів буде порожнім, якщо модель задовольняє описаним властивостям.

Сьогодні метод Model Checking використовується для перевірки складних об'єктів, як програмного забезпечення, так і апаратури. Він дозволяє істотно підвищити ступінь впевненості розробників в правильності функціонування інтегральних схем, протоколів комунікації, драйверів пристроїв, бортових систем керування для автомобілів, літаків, космічних апаратів. У роботі розглядається методика розробки вбудованого ПЗ саме на основі перевірки моделі.

Верифікація програм – це прийоми і методи формального доведення (або спростування) того, що модель програмної системи задовольняє заданій формальної специфікації. Для того щоб довести формально будь-яке твердження щодо роботи реальної системи, аналізована система (реалізація) повинна бути представлена формальною моделлю. Формальна модель зазвичай простіше самої перевіряючої системи, це абстракція, в якій відображені найбільш суттєві характеристики системи.

Для верифікації вбудованої системи управління також необхідно побудувати її модель. Подібні системи зазвичай реалізуються на мовах високого рівня з формально визначеним синтаксисом і здаються повністю формалізованими. Проте з точки зору семантики це не так. Використання вказівників, обробка дійсних чисел з обмеженою точністю, складні структури даних, динамічне породження необмеженого числа потоків і їх

взаємодія – все це призводить до того, що з тексту програми не слід безпосередньо повне формальне опис її поведінки.

Оскільки в загальному випадку процес побудови формальної моделі системи не формалізований і не однозначний, має сенс виділити клас підсистем, для яких можна автоматично побудувати адекватну формальну модель. До таких систем, наприклад, відносяться реагуючі системи: системи, що дають відгук на зовнішню подію в залежності від свого стану. Реагуючі системи часто є частиною більшої програмної системи. Операційні системи, протоколи комунікації, планувальники, контролери, паралельні взаємодіють програми, системи логічного керування, драйвери – все це приклади реагуючих систем.

Консорціумом OMG запропоновано підхід до проектування програм на основі моделей, або під управлінням моделей, названий Model-Driven Engineering (MDE). У MDE формальні моделі програмних систем первинними є в процесі проектування. Специфікація системи відокремлена від технології реалізації чи платформи. Опис системи існує незалежно від конкретики реалізації і може бути формально трансльовано на велику кількість платформ. Верифікація необхідних властивостей за методом перевірки моделі проводиться саме за формальної моделі програми. Для верифікації можна використовувати будь-який з відомих верифікаторів, що реалізує метод перевірки на моделі. Саме ця стратегія розробки MDE прийнята в розробленні методології проектування вбудованих реагуючих систем.

Приклади використання підходу Model Checking для верифікації програм:

- Cambridge ring protocol,
- IEEE Logical Link Control protocol, LLC 802.2,
- Фрагменти великих протоколів XTP та TCP/IP,
- Криптографічні протоколи,

- DeepSpace1 (NASA),
- SLAM: Microsoft додав Model Checking в Driver Development Kit для Windows.

3.3 Типи перевіряючих вимог

Для аналізованого класу систем управління була розроблена методика виділення вимог щодо поведінки таких систем, виконання яких на моделі свідчить про досить високу якість цієї системи.

Можна виділити чотири типи (множини) вимог коректності:

1. Вимоги коректності, пов'язані з відсутністю типових помилок паралельних процесів (блокування (зависання) процесів, порушення властивостей безпеки і жвавості, одночасний доступ до критичних областей або доступ до них не в необхідній послідовності та деякі інші вимоги).

2. Вимоги коректності, що випливають з технічного опису функціонування системи (вимоги до поведінки системи у всіх базових сценаріях і режимах функціонування).

3. Вимоги керованості системи (досяжність всіх станів, що характеризують виконання необхідних функцій).

4. Вимоги до надійності (неможливість переходу в критичні й небезпечні режими крім суворо приписаних регламентованих процедур).

РОЗДІЛ 4
ВЕРИФІКАТОР SPIN

Існує велике число верифікаторів, в тому числі і з відкритим кодом.

В даний час найбільш популярним верифікатором є SPIN. Його популярність дозволяє сподіватися, що він не містить помилок.

Цей верифікатор – один з найбільш потужних та відомих верифікаторів. Його використовують NASA для системи Mars Exploration Rovers та багато інших організацій, таких як ATC PathStar, де потрібна підвищена надійність. Автоматні програми зручні для проектування та зрозумілі для використання. Крім того, вони дозволяють автоматично побудувати модель Кріпке та зробити верифікацію.

4.1 Розробка верифікатора SPIN

SPIN (Simple Promela Interpreter) – це утиліта для перевірки правильності розподілених програмних моделей [14]. Використовується для автоматизованої верифікації моделей будування паралельних моделей систем вираження вимог на мові LTL.

Розвивається з 1980 року центром Computing Sciences Research Center в Bell Labs. З 1991 року програма активно використовується та є безкоштовною.

На вхід цього верифікатора подаються модель програми, написана на мові Promela, і вимоги до моделі, написані на мові LTL. Верифікатор по моделі програми будує модель Кріпке, а по інверсії кожної вимоги – автомат Бюхі. Після цього верифікатор SPIN будує перетин моделі Кріпке та автомату Бюхі (не очікуючи повної побудови структури Кріпке) і якщо перетин непорожній, користувач отримує трасу з помилкою (контрприклад), яка може допомогти знайти помилку в програмі.

Іноді помилка відбувається в результаті некоректно заданої моделі або неправильної специфікації (вимог). В цьому випадку траса помилки допомагає вирішити помилку в моделюванні або специфікації.

Крім перевірки моделей, SPIN може працювати як симулятор, виконуючи один із можливих шляхів роботи системи та надаючи програмісту результати виконання.

На відміну від багатьох програм для перевірки моделей, SPIN не виконує роботу сама, а генерує програму на мові C, яка вирішує конкретну задачу. За рахунок цього досягається економія пам'яті та підвищення ефективності.

З 1995 року майже кожен рік проводяться семінари SPIN для користувачів програм і тих, хто займається дослідженнями в області перевірки моделей.

У 2001 році розробники системи отримали премію ACM System Award.

4.2 Promela

Promela – це спеціалізована мова високого рівня, ця мова використовується верифікатором SPIN. Синтаксис мови нагадує синтаксис мови C. Модель на мові Promela складається з наступних елементів:

- оголошення типів даних,
- оголошення каналів передачі змінних,
- оголошення змінних,
- оголошення процесів,
- процес init.

Процес можна розглядати як процедуру яка виконується в окремому потоці.

Приклад процесу:

```
proctype proc(int a; int b) {
byte b; /* локальна змінна */
/* тіло процесу */
}
```

Процеси можуть мати параметри та локальні змінні. Процес може бути запущений у кількох екземплярах, якщо для нього встановлений модифікатор active. Запускаються процеси за допомогою модифікатора Run.

Мова Promela має 5 типів даних:

- bit,
- bool,
- byte,
- short,
- int.

Тіло процесу складається із послідовності операторів. Оператори можуть бути здійсненними або заблокованими. Здійсненний оператор може бути виконаний негайно. Заблокований оператор - оператор, який не може бути виконаний у цей момент. Такий оператор блокує виконання процесу доти, доки він не стане здійсненним.

Наприклад, оператор x < 7 може бути виконаний тільки в тому випадку, якщо x менше за 7. В протилежному випадку він зупиняє виконання процесу до тих пір, поки значення x не стане менше за 7. Деякі оператори, наприклад оператор привласнення, здійсненні завжди.

Мова Promela також містить оператор циклу, синтаксис якого заснований на командах Дейкстри.

Приклад циклу:

```
if
    ::guard1 -> S1
    ::guard2 -> S2
```

```
        ...
        :: else -> Sk
fi
```

4.3 Реляційна модель Кріпке та автомат Бюхі

Нехай AP – множина атомарних формул. Реляційна модель (модель Кріпке) над AP – це четвірка M = (S, S0, R, L), в якій:

- S – скінченна множина станів,
- $S0 \subseteq S$ – множина початкових станів,
- $R \subseteq S \times S$ – відношення переходів,
- $L : S \rightarrow 2^{AP}$ – функція істинності.

Модель Кріпке пристосована для верифікації. Для написання вимог до неї використовується мова темпоральної логіки: LTL, CTL, CTL* та інші.

При верифікації програм із застосуванням апарату темпоральних логік поряд із реляційними моделями Кріпке використовуються спеціальні скінченні автомати – автомати Бюхі.

Нехай AP – множина пропозиційних символів мови (імен атомарних висловлювань). Автоматом Бюхі над алфавітом 2^{AP} називається четвірка A = (Q, q0, δ, F), в якій:

- Q – скінченна множина станів,
- q0 – початковий стан,
- $\delta \subseteq Q \times 2^{AP} \times Q$ – функція переходів,
- $F \subseteq Q$ – множина допустимих (фінальних) станів.

Доведено що для будь-якої LTL-формули можливо побудувати автомат Бюхі, який її виконує. Більш того, із використанням верифікатора SPIN цей автомат за LTL-формулою можна побудувати автоматично.

Приклад:

Розглянемо LTL-формулу □(pUq). Вона означає, що завжди гарантовано, що умова p залишається істиною, принаймні до тих пір, доки не стане істиною умова q. Верифікатор SPIN її транслює у конструкцію never claim:

```
never { /* □ (p U q) */
T0_init:
        if
        :: ((q)) -> goto accept_S9
        :: ((p)) -> goto T0_init
        fi;
accept_S9:
        if
        :: ((((p)) || ((q)))) -> goto T0_init
        fi;
}
```

Ця конструкція відповідає автомату Бюхі, зображеному на рис. 4.1. Подвійна лінія означає допустимий стан.

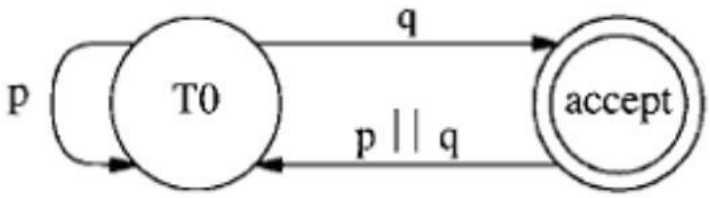

Рисунок 4.1 – Автомат Бюхі для формули □(p U q)

За допомогою автомата Бюхі можна верифікувати реляційну модель Кріпке. З погляду верифікації автоматних моделей це дуже зручний

варіант, адже він дозволяє при верифікації та специфікації майже повністю обмежитись класом скінченних автоматів.

Змістовний і досить цікавий приклад верифікації програми автоматної реалізації математичної гри Нім із використанням верифікатора SPIN наведено в додатку.

ВИСНОВКИ

У даній бакалаврській роботі був проведений аналітичний огляд та опрацювання літератури, присвяченої темпоральним логікам та можливостям їх застосування в інформатиці й програмуванні. Виконано огляд систем темпоральної логіки, описано мови та реляційну семантику таких логік. Детально розглянуто лінійну пропозиційну темпоральну логіку (ЛПТЛ), наведено приклад застосування ЛПТЛ для вирішення проблеми взаємного виключення алгоритмом Петерсона. Описано потужний метод верифікації програмних систем Model cheking, який використовує апарат лінійної темпоральної логіки. Розглянуто популярний засіб верифікації програм – верифікатор SPIN. Наведено змістовний приклад застосування апарату темпоральної логіки на практиці – верифікація програми автоматної реалізації гри Нім із використанням SPIN.

На основі проведеного дослідження можна зробити висновок, що поняття і методи темпоральної логіки можуть бути успішно застосовані фактично в усіх сферах наукової та практичної діяльності людини, де ми маємо справу з категорією часу. В наш час темпоральні логіки набувають все більшого значення у зв'язку зі створенням та розвитком сучасних інформаційних та програмних систем. Особливо важливим видається використання апарату темпоральної логіки для моделювання різноманітних предметних областей, для вирішення задач специфікації та верифікації програм.

ПЕРЕЛІК ДЖЕРЕЛ

1. Вавіленкова А.І. Застосування темпоральної логіки при побудові формальних моделей електронних текстів / А. І. Вавіленкова // Вісник НТУ «ХПІ», Серія: Нові рішення в сучасних технологіях. – Харків: НТУ «ХПІ». – 2017. – № 23 (1245). – С. 84-88.
2. Вельдер С.Э., Лукин М.А., Шалыто А.А., Яминов Б.Р. Верификация автоматных программ. – СПб: Наука, 2011, 244 с.
3. Ішмуратов А.Т. Вступ до філософської логіки. – К., 1997.
4. Конверський А.Є. Сучасна логіка / Конверський А. Є. – К.: « Центр учбової літератури», 2017.
5. М.С. Нікітченко, С.С. Шкільняк. Математична логіка та теорія алгоритмів. – К.: Київ. ун-т, 2008.
6. М.С. Нікітченко, С.С. Шкільняк. Прикладна логіка. – К.: Київ. ун-т, 2013.
7. Л.Л. Омельчук. Формальні методи специфікації програм. – К.: УкрІНТЕІ, 2010
8. С.С. Шкільняк. Математична логіка. Основи теорії алгоритмів. – К.: Персонал, 2009.
9. Handbook of Logic in Computer Science. Edited by S. Abramsky, Dov M. Gabbay and T. S. E. Maibaum. – Oxford Univ. Press. – Vol. 1–5, 1993–2000
10. E.Allen Emerson. Temporal and modal logic // Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B) – 1990.
11. F. Kröger, S. Merz. Temporal logic and state systems.– Berlin-Heidelberg: Springer-Verlag, 2008.
12. Logics of Specification Languages. EATCS Series, Monograph in Theoretical Computer Sciens / Eds. D. Bjorner, M.C. Henson. – Heidelberg: Springer, 2008.

13. Leslie Lamport. Specifying systems: the TLA+ language and tools for hardware and software engineers. – Addison-Wesley Professional; 1 edition, 2002.
14. LTL Model Checking with SPIN (http://spinroot.com/spin/whatispin.html).
15. Z. Manna, A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems: Specification. – Springer-Verlag, New York, 1991.

ДОДАТОК

Приклад верифікації програми

Розглянемо автоматну реалізацію (рис. 1) відомої математичної гри Нім.

Нім – це гра для двох гравців, кожен із яких по черзі робить хід. Перед гравцями розміщується поле з фішками. Відомі різні варіанти цієї ігри. У цьому проекті правила гри такі:

- фішки розкладаються у кілька рядів;
- гравці по черзі забирають фішки будь-якого ряду;
- не дозволяється за один хід брати фішки з кількох рядів;
- за один хід гравець має взяти хоча б одну фішку;
- виграє той, хто візьме останню фішку.

У цьому прикладі була використана одна з перших версій реалізації з одним автоматом. Цифрами вказані номери станів, надані інструментом Converter [2] – цей інструментальний засіб автоматично створює модель на мові Promela, а LTL-формула автоматично перетворюється на вигляд, придатний для верифікатора SPIN).

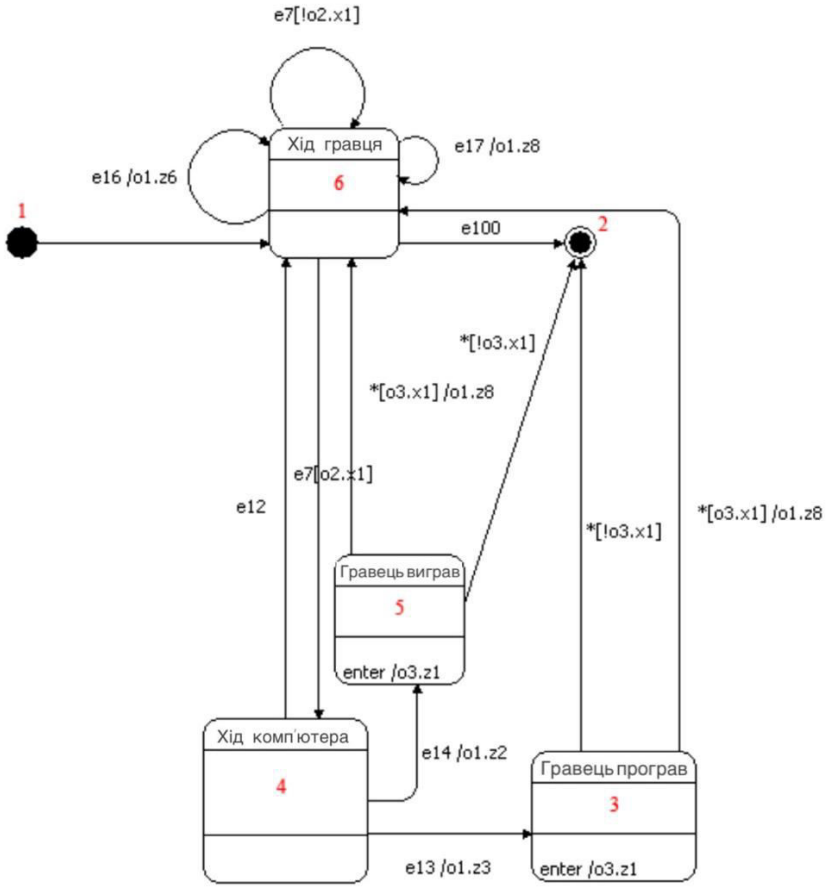

Рисунок 1 – Автоматна реалізація гри Нім

Припустимо, в ній була допущена помилка (рис. 2). Зайвий перехід виділений жирним шрифтом.

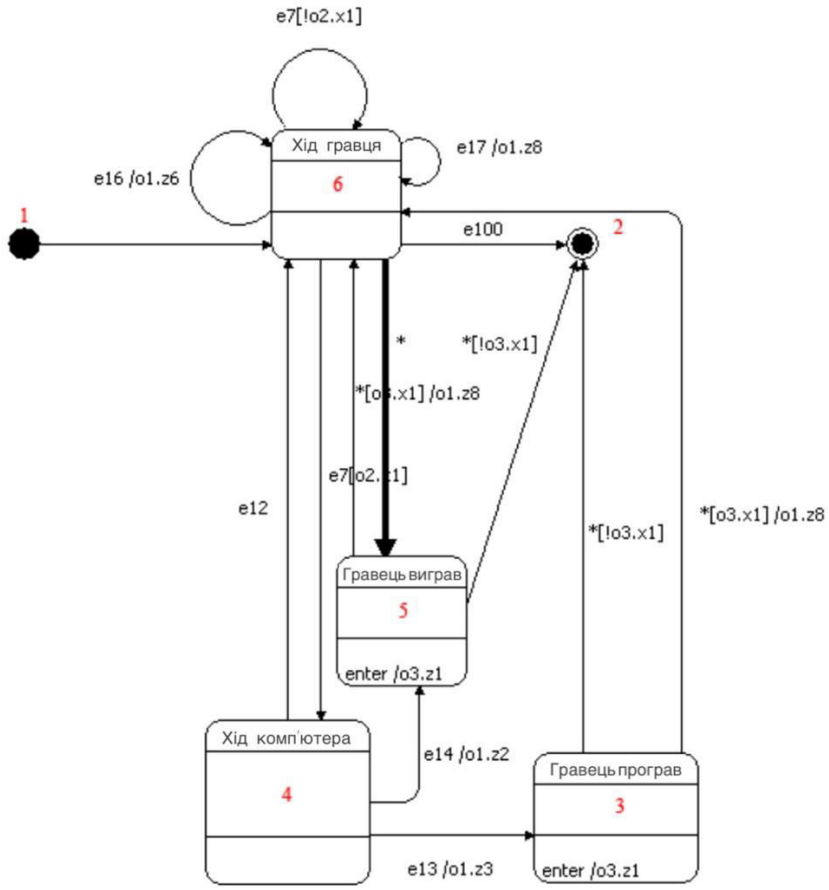

Рисунок 2 – Реалізація з помилкою

Опишемо процес верифікації програми автоматної реалізації гри Нім.

В даній реалізації рішення про те, хто виграв, приймалося в стані "Хід комп’ютера". Запишемо цю вимогу на мові LTL:

$$\neg (stateA1 != 4)\ U\ (stateA1 == 5).$$

Запишемо у форматі для Converter:

$$!(\{stateA1 != 4\}\ U\ \{stateA1 == 5\}).$$

Подамо на вхід Converter неправильний автомат та заперечення сформульованої вимоги. Converter по автомату побудує наступну модель:

```
#define p1 (stateA1 != 4)
#define p2 (stateA1 == 5)
int lastEvent;
```

```
#define STATE_0 0 /*Top*/
#define STATE_1 1 /*s2*/
#define STATE_2 2 /*s3*/
#define STATE_3 3 /*Гравець програв*/
#define STATE_4 4 /*Хід комп'ютера*/
#define STATE_5 5 /*Гравець виграв*/
#define STATE_6 6 /*Хід гравця*/
int stateA1;
inline A1() {
	stateA1 = STATE_1;
	do
	::(stateA1 == STATE_1) ->
		printf("State 1 : s2\n");
		if
			::stateA1 = STATE_6;
		fi;
	::(stateA1 == STATE_2) ->
		printf("State 2 : s3\n");
		break;
	::(stateA1 == STATE_3) ->
		printf("State 3 : Гравець програв\n");
		if
			::stateA1 = STATE_2;
			::stateA1 = STATE_6;
		fi;
	::(stateA1 == STATE_4) ->
		printf("State 4 : Хід комп'ютера\n");
		if
			::stateA1 = STATE_3;
```

```
                    lastEvent = 13;
                ::stateA1 = STATE_6;
                    lastEvent = 12;
                ::stateA1 = STATE_5;
                    lastEvent = 14;
            fi;
        ::(stateA1 == STATE_5) ->
            printf("State 5 : Гравець виграв\n");
            if
                ::stateA1 = STATE_6;
                ::stateA1 = STATE_2;
            fi;
        ::(stateA1 == STATE_6) ->
            printf("State 6 : Хід гравця\n");
            if
                ::stateA1 = STATE_6;
                    lastEvent = 16;
                ::stateA1 = STATE_4;
                    lastEvent = 7;
                ::stateA1 = STATE_6;
                    lastEvent = 7;
                ::stateA1 = STATE_6;
                    lastEvent = 17;
                ::stateA1 = STATE_2;
                    lastEvent = 100;
                ::stateA1 = STATE_5; /*неправильний перехід*/
            fi;
        od;
}
```

```
proctype Model() {
	A1();
}
init {
run Model();
}
never { /* p1 U p2 */
T0_init:
	if
	:: ((p2)) -> goto accept_all
	:: ((p1)) -> goto T0_init
	fi;
accept_all:
	skip
}
```

Верифікатор SPIN по даній моделі побудує програму на мові C. Результатом роботи програми є повідомлення про кількість помилок та, якщо помилки знайдені, trail-файл. В даному випадку trail-файл виглядає так: -2:2:-2 -4:-4:-4 1:0:55 2:1:51 3:0:55 4:2:0 5:0:55 6:2:1 7:0:55 8:2:2 9:0:55 10:2:3 11:0:55 12:2:31 13:0:55 14:2:32 15:0:55 16:2:43 17:0:53 18:2:25 19:0:59 20:1:52.

Верифікатор SPIN оброблює цей файл та будує контрприклад, а розроблений інструментальний засіб Converter все збирає воєдино та будує звіт:

```
pan: claim violated! (at depth 19)
pan: wrote models/incorr.ltl.trail
(SPIN Version 4.2.8 -- 5 June 2022)
Warning: Search not completed
	+ Partial Order Reduction
```

Full statespace search for:

never claim +

assertion violations + (if within scope of claim)

acceptance cycles - (not selected)

invalid end states - (disabled by never claim)

State-vector 24 byte, depth reached 27, **errors: 1**

20 states, stored

4 states, matched

24 transitions (= stored+matched)

0 atomic steps

hash conflicts: 0 (resolved)

2.622 memory usage (Mbyte)

Початок контрприкладу.

Starting :init: with pid 0

Starting :never: with pid 1

Never claim moves to line 75 [((stateA1!=4))]

Відображення переходу автомату Бюхі.

Starting Model with pid 2

2:proc 0 (:init:) line 69 "models/incorr.ltl"

(state 1) [(run Model())]

4:proc 1 (Model) line 15 "models/incorr.ltl"

(state 1) **[stateA1 = 1]**

Автомат A1 переходить до стану 1 (початковий стан).

6:proc 1 (Model) line 17 "models/incorr.ltl"

(state 2) [((stateA1==1))]

State 1 : s2

8:proc 1 (Model) line 18 "models/incorr.ltl"

(state 3) [printf('State 1 : s2\\n')]

10:proc 1 (Model) line 20 "models/incorr.ltl"

(state 4) **[stateA1 = 6]**

Автомат A1 переходить до стану 6 (Хід гравця).

12:proc 1 (Model) line 47 "models/incorr.ltl"

(state 2) [((stateA1==6))]

State 6 : Хід гравця

14:proc 1 (Model) line 48 "models/incorr.ltl"

(state 33) [printf('State 6 : Хід гравця\\n')]

16:proc 1 (Model) line 60 "models/incorr.ltl"

(state 44) **[stateA1 = 5]**

Автомат A1 переходить до стану 5 (Гравець виграв).

Never claim moves to line 74 [((stateA1==5))]

18:proc 1 (Model) line 41 "models/incorr.ltl"

(state 26) [((stateA1==5))]

Never claim moves to line 78 [(1)]

SPIN: trail ends after 20 steps

#processes: 2

lastEvent = 0

stateA1 = 5

20:proc 1 (Model) line 42 "models/incorr.ltl"

(state 27)

20:proc 0 (:init:) line 70 "models/incorr.ltl"

(state 2) <valid end state>

20:proc - (:never:) line 79 "models/incorr.ltl"

(state 8) <valid end state>

2 processes created

Для зручності жирним шрифтом виділено моменти входу до нового стану, а також повідомлення про кількість помилок. У звіті Converter з’явився текст "errors: 1", який повідомляє про помилку.

Проаналізуємо тепер звіт, отриманий інструментом Converter.

На початку кожного рядка траси є номер процесу та його назва в круглих дужках (Наприклад, proc 1 (Model)). Після цього записаний номер рядка в модель на мові Promela та назва файлу з моделлю ("line 15 "models/incorr.ltl"). Далі в круглих дужках написаний номер стану в моделі Кріпке побудований SPIN ((state 1)). В квадратних дужках вказаний код на мові Promela ([stateA1 = 1]).

Таким чином зі звіту слідує що контрприкладом є щлях (рис. 3): початковий стан – стан 6 (Хід гравця) – стан 5 (Гравець виграв).

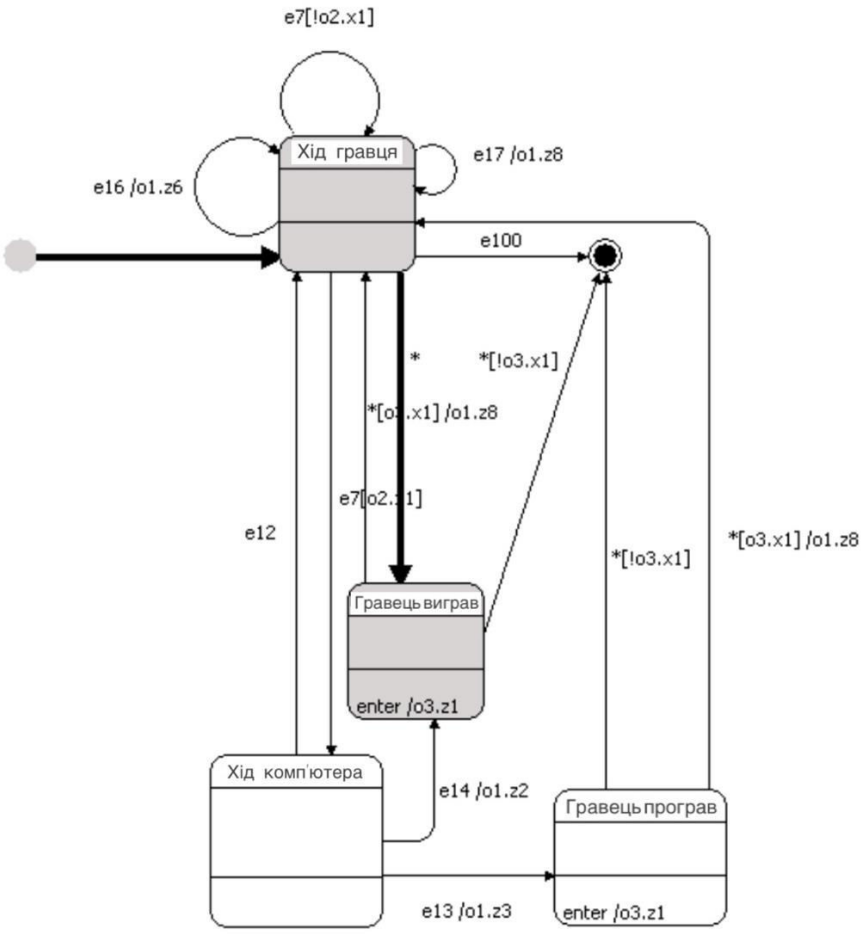

Рисунок 3 – Контрприклад

Тепер подамо на вхід Converter правильний автомат та таку ж вимогу. Рядок зі звіту "errors: 0" каже нам, що помилка була виправлена.