Лабораторна робота 3

ВЕРИФІКАЦІЯ ПРОГРАМ НА ОСНОВІ ТЕМПОРАЛЬНОЇ ЛОГІКИ

© *Шубін І. Ю.*

Верифікація програм на основі темпоральної логіки

- **Мета заняття**
  - Навчитися проводити верифікацію автоматних моделей програм на основі темпоральної логіки LTL.

Методичні вказівки з організації самостійної роботи студентів

- Для виконання роботи використовується SPIN [1, с.4-32,6 с. 1-126] – верифікатор моделей паралельних програм, написаних на С-подібній мові Promela (Protocol Meta Language). Розроблений в дослідницькому центрі Bell Labs Джерардом Холзманном. Міжнародна асоціація ACM нагородила Spin премією ACM Software System Award за 2001 р. В 1983 році цією премією була нагороджена ОС UNIX, в 1997 р. – Tcl/Tk, в 2002 г. – мова Java.

Пакет Spin

- Пакет Spin дозволяє:
  - будувати моделі паралельних програм (протоколів, драйверів, систем логічного контролю та управління) та широкого класу дискретних систем;
  - виражати потрібні властивості їх поведінки (темпоральні властивості);
  - автоматично шляхом натиснення кнопки перевіряти виконання темпоральних властивостей моделей на основі фомального підхода.

Пакет Spin (продовження)

- Spin може використовуватися в двох режимах: як симулятор та як верифікатор. Під час симуляції Spin у графічному режимі виводить інформацію про одну конкретну траєкторію виконання побудованої моделі – графічне подання поведінки у вигляді діаграми «послідовності повідомлень» (Message Sequence Diagrams), що виникає під час функціонування паралельних процесів.

Пакет Spin (продовження)

- Виконуючи симуляцію треба розуміти, що ніяка кількість симуляцій не може **довести** властивостей моделей. Для цього треба виконувати **верифікацію**. Верифікатор намагається знайти контрприклад – неправильну, помилкову траєкторію поведінки, що спростовує задану користувачем властивість, через аналіз всіх можливих виконань моделі. Для цього він будує синхронну комбінацію моделі переходів системи, що перевіряєтья, та автомату Бюхі, який задає всі небажані некоректні поведінки. Якщо перетин моделей непустий – це і є контр приклад, Spin демонструє його користувачу у керованому режимі. Таким чином симуляція та верифікація у Spin тісно пов’язані.

Використання Spin на платформі Windows

- Для використання spin на платформі Windows знадобиться наступний мінімальний набір утиліт:
  - mingw – windows port GNU компілятора gcc;
  - ActiveTcl – інтерпретатор високорівневої мови написання скриптів Tcl/Tk;
  - Graphviz – бібліотека для побудови широкого спектру графіків та діаграм;
  - Spin – консольна утиліта для інтерпретації програм на Promela;
  - xspin – Tcl/Tk скрипт, який реалізує графічну оболонку для полегшення та автоматизації роботи з інтерпретатором Spin.

Використання Spin на платформі Windows (продовження)

- Директорії bin всіх утиліти повинні бути включені до системної змінної PATH.
- Синтаксис мови Promella детально розглядався на лекції та в [1,2]. Розглянемо приклад роботи з системою. Для запуску утиліти треба створити та запустити скрипт run.bat наступного змісту:
  - wish -f xspin525.tcl,
- де xspin525.tcl – скрипт графічної оболонки. Зовнішній вигляд окна візуальної оболонки наведено на рис. 1.

Використання Spin на платформі Windows (продовження)

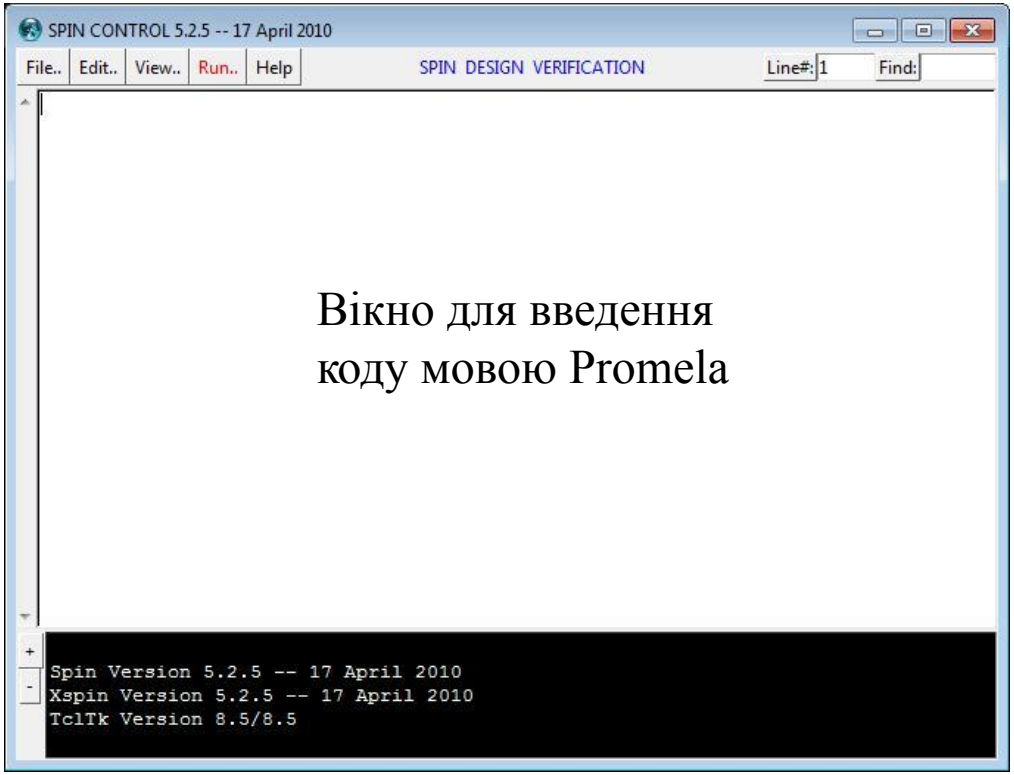

Рисунок 1 – xspin під час роботи

Використання Spin на платформі Windows (продовження)

- Програму можна вводити прямо в редакторі чи завантажити з файла *.pml через File/Open. Завантажте до середовища файл altbit.pml. В ньому реалізовано протокол альтернуючого біта. В алгоритмі альтернуючого біта процес-відправник почерзі відправляє повідомлення, помічені то бітом 1, то бітом 0, і очікує відповідні підтвердження. Процес-отримувач отримує повідомлення, помічені то бітом 1, то бітом 0, і відсилає процесу-відправнику підтвердження на них.

Використання Spin на платформі Windows (продовження)

```
      mtype = { msg, ack };/*объявляем два возможных типа сообщения */
      /*объявляем канал отправителя */
      chan to_sndr = [2] of { mtype, bit };
      /*объявляем канал получателя */
      chan to_rcvr = [2] of { mtype, bit };

active proctype Sender()/* процесс отправителя */
{
again: to_rcvr!msg,1; /* отправляем сообщение, помеченное битом 1 */
      to_sndr?ack,1; /* получаем подтверждение, помеченное битом 1 */
      to_rcvr!msg,0; /* отправляем сообщение, помеченное битом 0 */
      to_sndr?ack,0; /* получаем подтверждение, помеченное битом 0 */
```

Використання Spin на платформі Windows (продовження)

```
goto again
}
/* процесс получателя */
active proctype Receiver(){
again: to_rcvr?msg,1;/* получаем сообщение, помеченное битом 1 */
     to_sndr!ack,1;/* отправляем подтверждение, помеченное битом 1 */
     to_rcvr?msg,0; /* получаем сообщение, помеченное битом 0 */
     to_sndr!ack,0;/* отправляем подтверждение, помеченное битом 0 */
goto again
}
```

Використання Spin на платформі Windows (продовження)

- Перевірте синтаксичну коректність програми, обравши Run/Run Syntax Check. У вікно повідомлень повинно видатися «no syntax errors». Тепер можна запустити симуляцію Run/Set Simulation Parameters (рис. 2)

Використання Spin на платформі Windows (продовження)

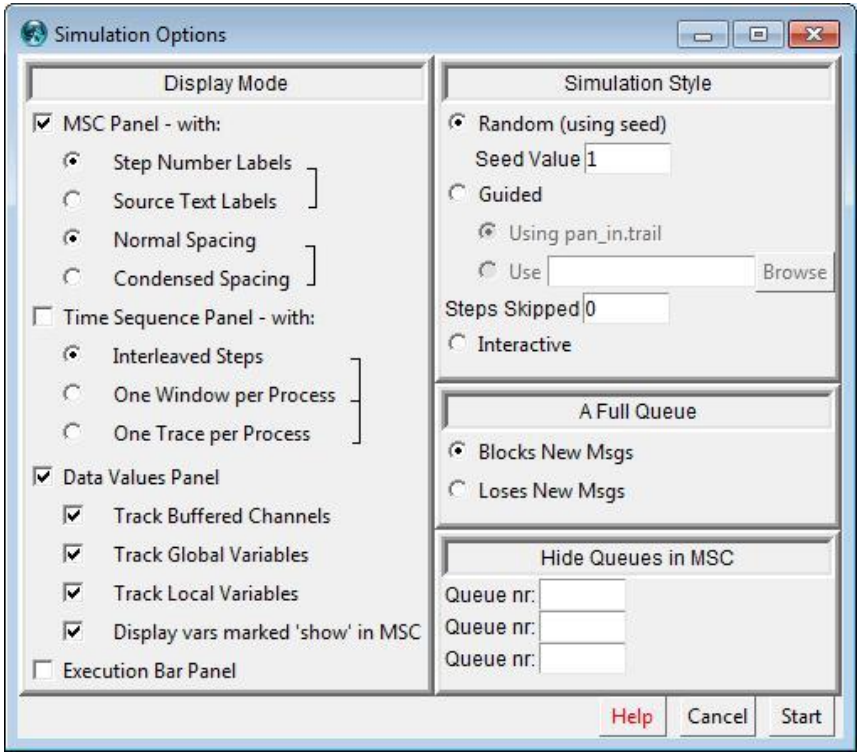

Рисунок 2 – Вікно налаштування параметрів симуляції

Використання Spin на платформі Windows (продовження)

- Натисніть Start. Відкриється профіль симуляції, де у вікні Simulation Output будуть виводитися повідомлення програми, а в Sequence Chart – графічно відображатися граф взаємодії процесів(рис. 3). Симуляція підтримує 2 режими: покрокова (Single Step) та безперервна (Run).

Використання Spin на платформі Windows (продовження)

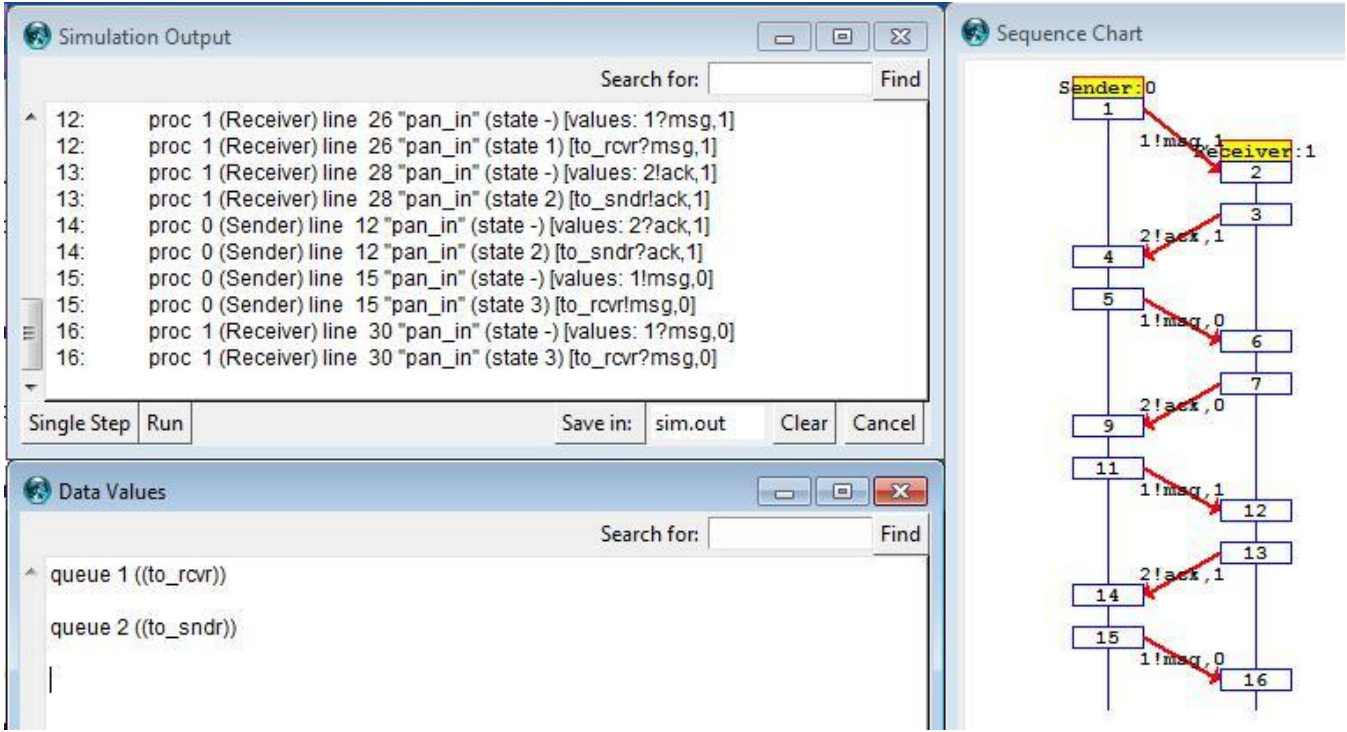

Рисунок 3 – Профіль симуляції взаємодії процесів

Перевірка коректності моделі на основі LTL

- Перевірка коректності моделі на основі верифікації LTL формул полягає в тому, що у виді формули LTL виражається деякі властивості її «правильної» поведінки. Всі такі властивості повинні бути перевірені по черзі одна за одною.

Класи властивостей розподілених систем

- Традиційно властивості розподілених систем поділяються на наступні класи:
- властивості досяжності (*reachability*), встановлюють, що специфічний стан системи може бути досягнутий, загальний вид LTL формули **EF**ϕ;
- властивості безпеки (*safety*), встановлюють, що дещо небажане ніколи не станеться в системі, загальний вид LTL формули **G**¬ϕ;
- властивості живості (*liveness*), встановлюють, що за деяких умов дещо “добре” в кінці кінців відбудеться при будь-якому сценарії поведінки системи, загальний вид LTL формули **GF**ϕ;
- властивості справедливості (*fairness*), встановлюють, що дещо буде виконуватися невизначено часто, загальний вид LTL формули

Перевірка коректності моделі на основі LTL (продовження)

- Для перевірки моделі на основі LTL введемо до програми додаткову змінну accepted, яка буде встановлена у 0 на початку запуску і кожного разу, коли повідомлення відправляється клієнту. Accepted встановлюється в 1 тільки, якщо від клієнта прийшло підтвердження. Тепер можна виразити вимогу, що модель коректна, якщо на кожну відправку обов’язково прийде підтвердження

Перевірка коректності моделі на основі LTL (продовження)

- Run/LTL Property Manager (рис. 4).

```
#define getone (accepted == 1)

#define getzero (accepted == 0)


[] (getzero -> (getzero U getone))
```

Оператори Spin у LTL

SPIN	LTL	Пояснення
<>q	Fq	Хоча б одного разу у майбутньому *q*
[]q	Gq	Завжди у майбутньому *q*
w U q	w U q	*w* до моменту, коли настане *q*

Перевірка коректності моделі на основі LTL (продовження)

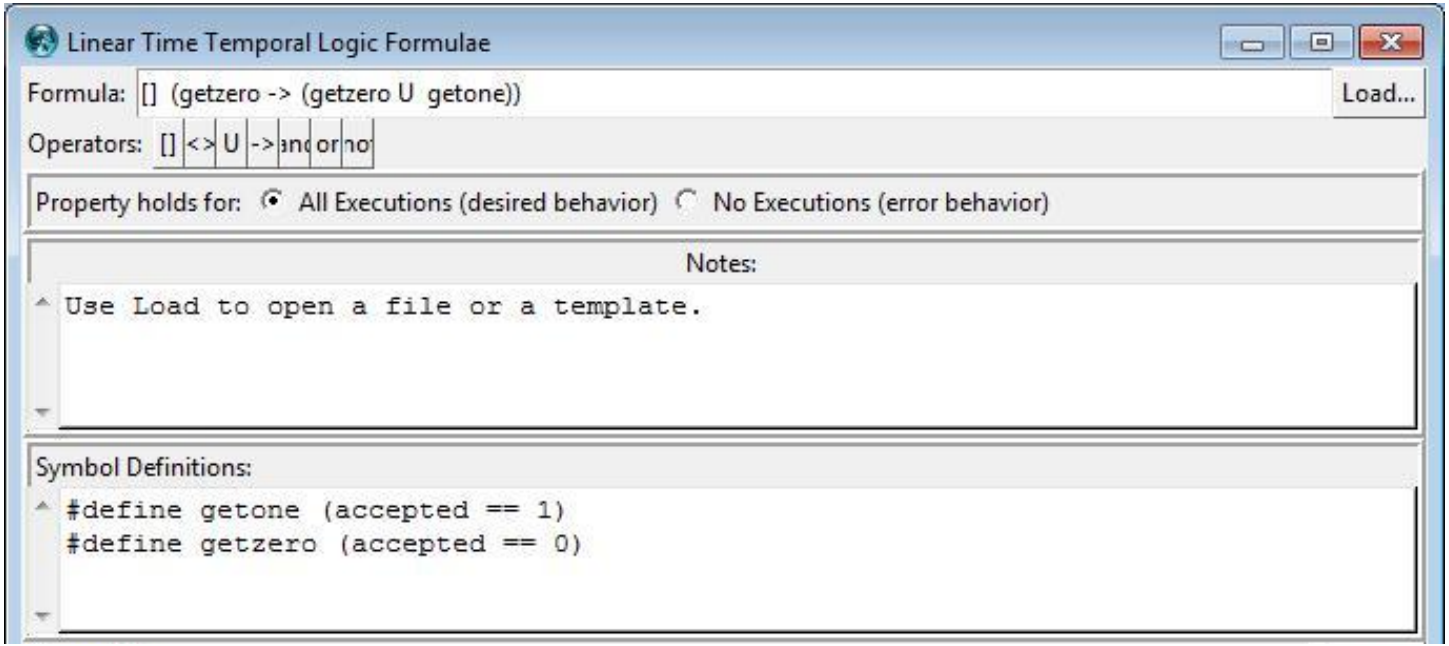

Рисунок 4.1 – Вікно верифікації системи

Перевірка коректності моделі на основі LTL (продовження)

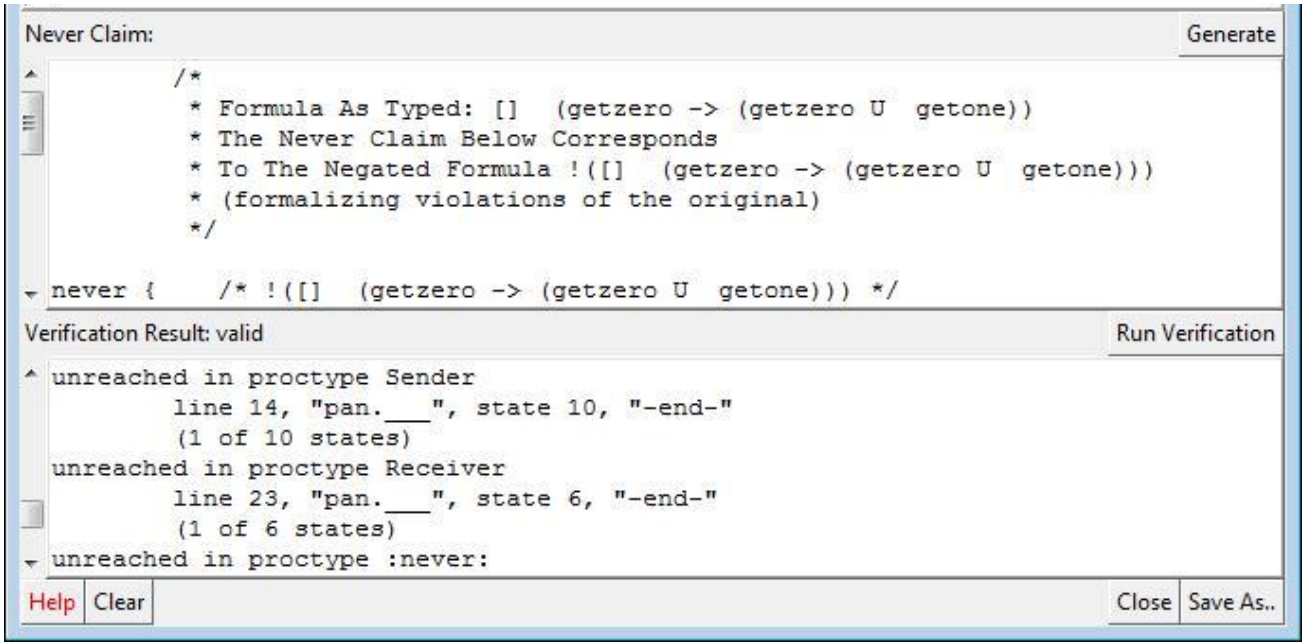

Рисунок 4.2 – Вікно верифікації системи

Перевірка коректності моделі на основі LTL (продовження)

- Зверніть увагу, ми хочемо перевірити, що формула істина при всіх виконаннях системи, тому обрано радіокнопку “All executions” Тепер натисніть Generate – створиться процесс Never Claim, який мовою Promela містить вираження введеної формули. Тепер натисніть Run Verification. У полі Verification Results буде видано позитивний звіт за результатами перевірки: **valid**

Перевірка коректності моделі на основі LTL (продовження)

```
unreached in proctype Sender
    line 14, "pan.___", state 10, "-end-"
    (1 of 10 states)
unreached in proctype Receiver
    line 23, "pan.___", state 6, "-end-"
    (1 of 6 states)
unreached in proctype :never:
    line 50, "pan.___", state 14, "-end-"
    (1 of 14 states)
```

Перевірка коректності моделі на основі LTL (продовження)

- Аналогічним чином виконайте індивідуальне завдання.

- Звіт повинен містити код моделі мовою Promela, автомат станів моделі, фрагмент симуляції, перелік всіх (на вашу думку) необхідних LTL формул для верифікації моделі та відповідні їм протоколи (скріншоти екранів).

Варіанти індивідуальних завдань

1. Студент/Автомат з видачі напоїв;
2. Банкомат/користувач
3. Алгоритм знаходження НОД
4. Моделювання взаємодії процесів з N розподілених кластерів для забезпечення ексклюзивного доступу процесів до спільного ресурсу.

Варіанти індивідуальних завдань (продовження)

- *Пояснення*. N кластерів з M процесами користувачів на кожному та по одному процесу менеджеру на кластер. Загальний опис ідеї протоколу: коли процес-користувач хоче отримати доступ до ресурсу, він подає заявку своєму менеджеру та чекає на надання доступу, користується ним та повинен сповістити менеджера, коли ресурс звільниться. Менеджери взаємодіють один з одним, щоб контролювати/надати кожного моменту часу лише одному клієнту доступ до ресурсу. Менеджер, який ексклюзивно володіє ресурсом, тримає так званий віртуальний токен, який він передає клієнту за запитанням. Менеджер, який отримує заявку клієнта, але не має віртуального точена повинен запросити його у інших менеджерів.

Варіанти індивідуальних завдань (продовження)

- Прикад заготовки процесу-клієнту наведено нижче:

```
proctype user(chan mgr) /* user process in cluster */
{
req: mgr!P; /* request access from local manager */
mgr?G; /* gain access */
progress:
cs: count++;
assert(count == 1);
count--;
mgr!V; /* release access */
out:
skip;
goto req
}
```

Варіанти індивідуальних завдань (продовження)

- Додайте всі необхідні складові (інші процеси, необхідні глобальні та локальні змінні), щоб моделювання стало можливим. Створіть два кластера і по 2 процеси-клієнти на кожному.

- Докажіть на створеній системі наступні правила:
  а) тільки один процес може одночасно звертатися до ресурсу;
  б) відсутність голодування (якщо клієнт запросив ресурс, то рано чи пізно він його отримає).

Варіанти індивідуальних завдань (продовження)

5. Криптографічний протокол Нідхама-Шредера. Реалізувати протокол, ввести третю сторону (зловмісника) та довести можливість зовнішньої атаки (зловмисник дізнається секрет одної із сторін).

Контрольні запитання і завдання

1. Дайте визначення темпоральній логіці.
2. Які конструкції відрізняють темпоральну логіку від логіки предикатів?
3. Які підкласи темпоральних логік існують?
4. Дайте визначення моделі Кріпке та автомату Буші, як вони пов’язані між собою.
5. Поясніть стратегію перевірки істинності формули логіки CTL.
6. Поясніть стратегію перевірки істинності формули логіки LTL.
7. Які принципи роботи утиліти xSpin при вірифікації моделей програм на основі темпоральної логіки?

Перелік посилань

1. Шошмина, И.В. Введение в язык Promela и систему комплексной верификации Spin [Текст] / И.В. Шошмина, Ю. Г. Карпов; Санкт-Петербургский государственный политехнический университет. – СПб:Университет, 2009 г. – 66 с.
2. G. Holzmann The Spin Model Checker: Primer and Reference Manual [Електронний ресурс] / SPIN HomePage / Режим доступу: www/URL: http://spinroot.com/spin/Doc/Book_extras/ – 19.05.2012 г. – Загол. з екрану.

Запитання