Проблемы верификации ПО

Введение

Наталья Олеговна Гаранина
garanina@iis.nsk.su

Введение

План лекций

1. Постановка задачи. Надежность программ и систем. Логический язык спецификаций. Понятие корректности программ.
2. Метод Флойда.
    1. Метод индуктивных утверждений Флойда доказательства частичной
    корректности программ.
    2. Метод фундированных множеств Флойда доказательства завершаемости программ.
3. Метод Хоара верификации программ.
    1. Частичная корректность программ.
    2. Аксиоматическая семантика элементарных конструкций и циклов.

Введение

План лекций

4. Проверка моделей. Моделирование систем. Проверка моделей для временных логик.
    1. Структура Крипке. Логическое представление систем.
    2. Примеры параллельных и взаимодействующих систем.
    3. Спецификация программных систем с помощью временных логик CTL и LTL.
    4. Алгоритмы проверки на модели формул временных логик CTL и LTL.
    5. Устройство автоматической системы проверки моделей.
5. Основная проблема проверки моделей: методы решения.
    1. Символьная проверка моделей.
    2. Использование особенностей моделей: редукция, композиция, абстракция, симметрия.

Методы верификации

Статический анализ свойств ПО

- проверка формализованных правил корректного построения ПО;
- поиск часто встречающихся дефектов по некоторым шаблонам;
- хорошо автоматизируется;
- способен обнаруживать только ограниченный набор типов ошибок;
- проблема точности:
  - строгие методы анализа не допускают пропуска ошибок, но приводят к большому количеству ложных сообщений об ошибках
  - более лояльные методы выдают точный набор сообщений об ошибках, но могут пропустить ошибку.

Методы верификации

Динамические методы верификации

- анализ и оценка свойств программной системы делаются по результатам ее реальной работы (или моделей и прототипов).
- *(имитационное) тестирование*, *мониторинг*, *профилирование*.
- необходимо иметь работающую систему, ее компоненты, прототипы
  - нельзя использовать их на первых стадиях разработки
- можно контролировать характеристики работы системы в ее реальном окружении, которые иногда невозможно проанализировать с помощью других подходов.

Методы верификации

Динамические методы верификации

- дополнительная подготовка
  - разработка тестовой системы;
  - системы мониторинга, позволяющей контролировать характеристики поведения системы.
  - создание тестов,
    - готовятся заново для каждой проверяемой системы;
    - позволяет обнаружить множество дефектов в описании требований и проектных документах;
    - довольно трудоемкая задача, если необходимо получить адекватную оценку качества сложной системы;
- очень широко используется на практике
  - не слишком надежные, но достаточно дешевые техники,
    - (нестрогое) вероятностное тестирование
      - тестовые данные генерируются случайным образом;
    - тестирование на основе простейших сценариев использования.

Методы верификации

Формальные методы верификации

- формальные модели требований, поведения ПО и его окружения.
- анализ формальных моделей:
  - *дедуктивный анализ* (theorem proving),
  - *проверка моделей* (model checking),
  - *абстрактная интерпретация* (abstract interpretation)…
- значительные усилия на построение формальных моделей.
  - специалисты по формальным методам (их мало и они дороги);
  - построение формальных моделей нельзя автоматизировать;
  - анализ свойств в значительной мере может быть автоматизирован
    - требуется специфический набор навыков и знаний в разделах математической логики и алгебры.

Методы верификации

Формальные методы верификации

- активно используются в ряде областей, где последствия ошибки могут оказаться чрезвычайно дорогими;
- способны обнаруживать сложные ошибки, практически не выявляемые с помощью экспертиз или тестирования;
- формализация требований и проектных решений возможна только при их глубоком понимании
  - совместная работа специалистов по формальным методам и экспертов в предметной области;
- инструменты, эффективно решающие ограниченные задачи верификации ПО из определенного класса для промышленных проектов, требующие для применения минимальных специальных навыков и знаний.
- часто формальные методы верификации на практике применяются к аппаратному обеспечению.

Концептуальные ошибки

- Космос
  - NASA Mars Global Surveyor (2.10.2006)
    - Испортился от перегрева
    - Неправильные адреса памяти
    - Выбранное положение устройства относительно солнца привело к тому, что батарея, которая должна охлаждаться, наоборот — нагревалась.

Концептуальные ошибки

- Космос
  - June 4, 1996 — Ariane 5 Flight 501
    - Гибель аппарата на 40й сек после старта
    - Рабочий код Ariane 4 был использован в новой системе Ариан 5, но более быстрые двигатели новой системы запустили баг, вызвали ошибку в вычислениях в полетном компьютере.
    - Ошибка была в коде, который конвертировал 64-битные floating-point в 16-битные целые со знаком.
    - Результатом было большее число с последующим переполнением.
    - Сначала «упал» бэкапный компьютер, а через 0.05 сек основной.
    - Процессор управления двигателем выдал повышенную мощность, и на 40й сек ракета была уничтожена.

Концептуальные ошибки

- Медицина
  - Therac-25 — аппарат для радиационной терапии (1980-е)
  - Убило 5 человек.
  - Вместо слабого излучения включило сильное и не распознало этого.
  - Ошибка состязания (Race condition): результат зависит от порядка и/или времени выполнения действий.

Концептуальные ошибки

- 1992 – Система управления вызовами неотложной помощи Лондона, The London Ambulance Service Computer Assisted Dispatch (LASCAD)
  - Система должна была заменить «ручное управление», улучшить коммуникацию, локализацию и диспетчирование машин скорой помощи.
  - При работе в полу-ручном режиме были замечены некоторые проблемы, которые исправлялись на лету вручную. 26 окт 1992 система охватила весь Лондон и ручное управление было прекращено.
  - Хотя полная функциональность была реализована, в первые 2 дня стали накапливаться ошибки, система имела все меньше информации о локализации машин, запросы обрабатывались все медленнее, машины прибывали с опозданием.
  - Точные потери неизвестны, но считается, что результатом было от 10 до 30 смертей, которых можно было избежать

Концептуальные ошибки

- Электрификация
  - Северное затемнение ‘2003
  - 14.08.2003 северо-восток США и часть Канады
    на сутки остались без света.
  - Ошибка состязания (Race condition): результат зависит от порядка и/или времени выполнения действий.
- Телекоммуникация
  - Авария на телефонной сети AT&T (15.01.1990)
  - Отключение на большом участке сети
  - Ошибка на одном коммутационном узле посредством сообщения распространилась на соседний и т.д.
  - Случилась волна отключений.

Концептуальные ошибки

- Финансы
  - NYSE, Knight Capital Group (20 лет на рынке, 11% всей торговли американскими акциями) — Авг 2012
    - был загружен свеже-обновленный софт, и ПО начало покупать и тут же продавать более ста наименований акций, искусственно завышая их цену при каждой сделке.
    - Ошибка была обнаружена через 45 мин, все акции были проданы по реальной цене.
    - Потери составили 440 млн долларов, примерно 1.5 выручки за 2й кв. – 10 млн долл в минуту.
    - В конце 2012 KCG была поглощена компанией, Getco.
  - Нью-Йоркская фондовая биржа – Dec 18, 1995
    - начала работу на один час позже из-за коммуникационных проблем в ПО
    - Один час работы– оборот ~150-170 млрд долл США

Концептуальные ошибки

- Вооружение
  - 25 фев 1991, Война в Заливе – Патриот пропустил атаку Скада
    - Первоначально Патриот был предназначен для мобильного базирования и перехвата ракет на *коротких* дистанциях.
    - Батарея стояла развернутой в ожидании атаки более 100 часов.
    - Софт накапливал ошибки, связанные с вычислениями и конвертированием плавающий-целый.
    - За 2 недели до инцидента ошибка потери точности была замечена в Израиле, 21.02 было направлено сообщение. Обновление ПО достигло Дахрана на день позже.
    - 25.02 накопленная ошибка достигла 0.34 сек, что достаточно для выхода Скада из зоны обнаружения и поражения Патриотом.
    - Результат атаки по базе – 28 погибших и 98 раненых

Концептуальные ошибки

- Авиация
  - 26.06.1988, аэрошоу в Мулхаузе, Франция
    - Аэробус A320 разбился при выполнении показательного полета. Предварительный полет был успешен
      - Считается, что причиной стала комбинация ошибок, при которой ошибки ПО привели к неверному информированию пилота о высоте
  - Гибель Air France Flight 447, May 31, 2009, Airbus A330-200, рейс из Рио в Париж.
    - Ошибки бортовой компьютерной системы, всего было сгенерировано 24 ошибочных сообщения.
    - Погибло 216 пассажиров и 12 членов экипажа.
  - Гибель Korean Air Flight 801, August 5, 1997, Сеул – Гуам
    - Погибло 237 пассажиров и 17 членов экипажа.
    - Комбинация ошибок недавно измененнного ПО и вызванных ошибками ПО реакций пилотов

Концептуальные ошибки

- Транспорт
  - 1993-1995 – Аэропорт Денвера стоял больше года из-за внезапного отказа системы автоматической обработки багажа.
    - Задержка старта аэропорта – 16 мес. Потери составили около 0,560 млрд долл + потери только UA, Continental и FedEx около 430 млн долл
    - Комплексная проблема от архитектуры системы управления до проверки её работоспособности на уровне модели и тестирования

Концептуальные ошибки

- Компьютерные системы
  - Конкуренция за ресурсы:
    - В системах с высокой нагрузкой постоянный поток процессов с высоким приоритетом может не дать процессам с более низким приоритетом получить доступ к ресурсам никогда.
  - В MIT в 1973 был остановлен IBM 7094, были обнаружены процессы с низким приоритетом, которые были поставлены в очередь в 1967 (!) и так и не были запущены.

Случайные ошибки

- Фобос 1 «Марс» – СССР 1988
  - Запущен 7.07.1988
    - 29.08 программный сбой вызвал неверные команды и отключение системы ориентации и стабилизации.
    - Солнечные батареи потеряли Солнце, аккумуляторы быстро разрядились.
    - Аппарат стоимостью 300 млн руб был потерян. (сегодня это 2 млрд долл по офиц советскому курсу, около 200-300 млн долл по реальному)

Случайные ошибки

- Авг 2014, госпиталь в Мельбурне, Австралия (Austin Hospital)
  - 200 пациентов были объявлены умершими, одну семью успели известить
    - Объяснение: «человеческая ошибка» после изменений, сделанных в модулях и темплейтах подсистемы извещений о смерти, и сохраненных в стандартные формы – т.е., проверка связности модулей и данных не была выполнена.

Случайные ошибки

- Ноябрь 2000 — Национальный институт рака, Панама Сити
  - ПО планирования терапии от американской фирмы Multidata Systems Intl неверно вычисляло дозу для радиационной терапии.
  - Софт позволял использовать 4 блокирующих металлических экранов, врачи решили использовать 5.
  - Они обнаружили, что можно «обмануть» ПО, если устанавливать все 5 блоков как один с дырой посередине.
  - Они не понимали, что ПО дает другие рекомендации в этом случае и результатом вычислений становится доза в два раза больше.

Случайные ошибки

- 2003 – Saint Mary's Mercy Medical Center in Grand Rapids, Michigan
  - 8,500 живых пациентов были объявлены умершими.
    - При рутинном апгрейде ПО управления пациентами все шло хорошо, но в одном крошечном поле БД в определенном подмножестве пациентов терялась одна цифра.
      - ошибка отображения памяти.
    - Ошибочные сообщения о смерти были отправлены в Дел Социального страхования, который, в свою очередь, отвечает за решение о доступе пациента к программе финансирования Medicare

Случайные ошибки

- Июль 1-2, 1991 – ПО телефонных станций отключило сервис в Вашингтоне DC, Питтсбурге, Лос-Анджелесе и Сан-Франциско
  - Причина была в System 7 и ПО, разработанном компанией DSC Communications Corp для Bell Atlantic и Pacific Bell.
  - Ошибкой был один символ в одной строке кода, этого оказалось достаточно для краха системы.
- Многоцелевой самолет F-18 разбился
  - ошибка в обработке исключения (пропущено условие)
- F-14 потерян из-за неуправляемого штопора,
  - ошибки в тактическом ПО

Методы верификации

Синтетические методы

- Комбинирование нескольких перечисленных выше видов верификации.
  - динамические методы, использующие элементы формальных
    - *тестирование на основе моделей* (model-based testing, model driven testing)
    - *мониторинг формальных свойств* (runtime verification, passive testing)
  - инструменты построения тестов используют
    - формализацию некоторых свойств ПО,
    - статический анализ кода.
- Общая идея — попытаться сочетать преимущества основных подходов к верификации, смягчая их недостатки.

2. Верификация программных продуктов и систем

- Методы проверки правильности программных систем
  - Имитационное моделирование
    - тестируется модель продукта
  - Тестирование
    - тестируется собственно продукт

  - Проверяются результаты работы системы на ограниченном количестве входных данных.
    - +. Понятность и экономичность.
    - -. Отсутствие полноты.

2. Верификация программных продуктов и систем

- Методы проверки правильности программных систем
  - Дедуктивный анализ
    - с помощью аксиом и правил вывода доказывается корректность программы, аннотированной формулами некоторой логики
    - +. Применим к системам с произвольно большим количеством состояний. Полнота выявления ошибок.
    - -. Медленный метод. Плохо автоматизируемый. Часто неразрешимый.
  - Проверка моделей
    - обход заданных (всех) состояний модели системы с проверкой выполнимости в них спецификации, определённой формулой какой-либо логики.
    - +. Полная автоматизируемость. Полнота выявления ошибок.
    - -. Применим только к системам с конечным числом состояний (или сводимым к ним).

Общая схема формальной верификации

- Моделирование
  - Приведение программной системы к формальному виду, подходящему для автоматической проверки.
    - Компиляция, трансляция.
    - Абстрагирование.
- Спецификация
  - Задание свойств программной системы, которые необходимо проверить.
    - Логики: темпоральные, динамические, эпистемические, деонтические и многие другие.
    - Полнота спецификации.
      - Безопасность: ничего плохого никогда не случится.
      - Живость: что-нибудь хорошее обязательно получится.
- Верификация

Общая схема формальной верификации

- Моделирование
- Спецификация
- Верификация
  - Проверка соответствия модели системы её спецификации.
    - Полностью автоматическая.
    - Анализ результатов
      - Контрпример (ошибочная трасса)
      - Ложный контрпример
    - Большой размер модели.
      - Абстрагирование
    - Уточнение модели

Методы верификации Флойда

- Введение в методы

  дедуктивной верификации последовательных программ.
- Верификация программы:
  - установление соответствия программы ее требованиям.
- Дедуктивная верификация:
  - *логический вывод утверждения* о том, что программа соответствует требованиям на всех входах программы.
  - *математическими модели* программ и требований
    - формально определенное отношение их соответствия.

Методы верификации Флойда

- Рассматриваем только простые программы, написанные на структурном языке программирования:
  - без массивов;
  - без использования адресной арифметики;
  - без рекурсии (вызова подпрограмм);
  - без взаимодействия с окружением.

Методы верификации Флойда

- М*етоды Флойда* (*Флойда-Хоара*).
  - Предпосылки: Тьюринг (1947), идея индуктивных утверждений.
  - Развитие:
    - Роберт Флойд (1967) и
    - Тони Хоар (1969).

Методы верификации Флойда

1.1 Математическая модель программы

Переменные программы

- Конечное число переменных:
  - *Входные:* $\boldsymbol{x} = (x_1, x_2, \ldots, x_X)$
    - содержат входные значения и не меняются во время работы программы;
  - *Промежуточные:* $\boldsymbol{y} = (y_1, y_2, \ldots, y_Y)$
    - используются для хранения промежуточных результатов в процессе вычисления
  - *Выходные*: $\boldsymbol{z} = (z_1, z_2, \ldots, z_Z)$
    - содержат значения, вычисляемые данной программой

Методы верификации Флойда

1.1 Математическая модель программы

- Каждая переменная *v* принимает значения из *домена* $D_{v=}$
- *Входной домен* $D_{\boldsymbol{x}} = D_{x_1} \times D_{x_2} \times \ldots \times D_{x_X}$
- *Домен программы* $D_{\boldsymbol{y}} = D_{y_1} \times D_{y_2} \times \ldots \times D_{y_Y}$
- *Выходной домен* $D_{\boldsymbol{z}} = D_{z_1} \times D_{z_2} \times \ldots \times D_{z_Z}$
- *Универсальный домен*

$$D = D_{x_1} \cup \ldots \cup D_{x_X} \cup D_{y_1} \cup \ldots \cup D_{y_Y} \cup D_{z_1} \cup \ldots \cup D_{z_Z}$$

  - множество значений всех переменных.
- *Предикаты*
  - функции, принимающие значения из множества *{True, False}*.
- *Расширенный домен* переменной *v* : $D_{\boldsymbol{v}}^{+} = D_{\boldsymbol{v}} \cup \{\omega\}$.

Методы верификации Флойда

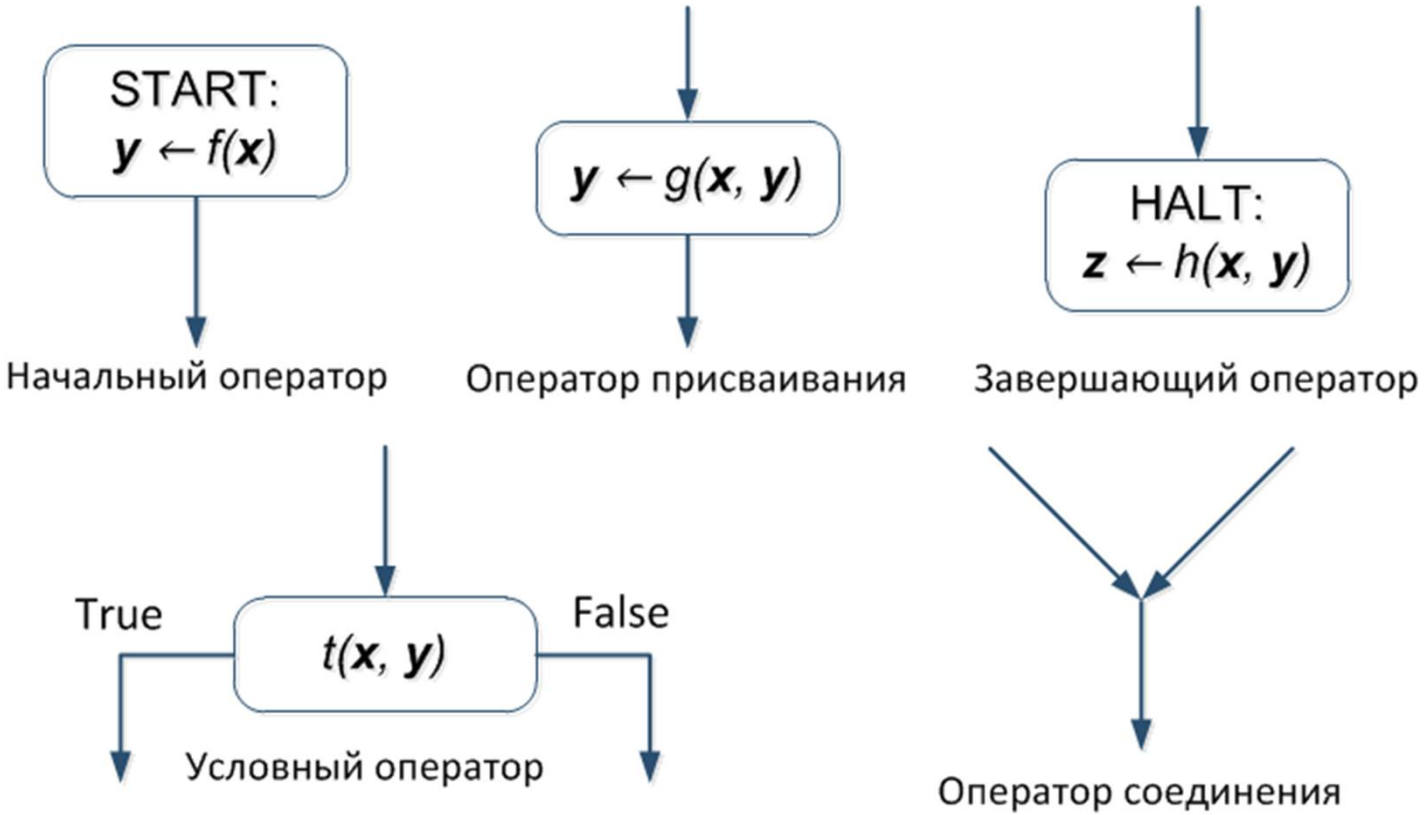
1.1 Математическая модель программы

Операторы программы

1. *Начальный оператор* START: $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$
   - Инициализирующая функция $f : D_{\boldsymbol{x}} \longrightarrow D_{\boldsymbol{y}}^{+}$;
2. *Оператор присваивания* ASSIGN: $\boldsymbol{y} \leftarrow g(\boldsymbol{x}, \boldsymbol{y})$
   - Вычисляющая функция $g: D_{\boldsymbol{x}} \times D_{\boldsymbol{y}} \longrightarrow D_{\boldsymbol{y}}^{+}$;
3. *Условный оператор* TEST: t (**x**, **y**)
   - Предикат $t : D_{\boldsymbol{x}} \times D_{\boldsymbol{y}} \longrightarrow \{True, False\}^{+}$;
4. *Оператор соединения* JOIN;
5. *Оператор завершения* HALT: $\boldsymbol{z} \leftarrow h(\boldsymbol{x}, \boldsymbol{y})$
   - Выходная функция $h: D_{\boldsymbol{x}} \times D_{\boldsymbol{y}} \longrightarrow D_{\boldsymbol{z}}^{+}$.

Методы верификации Флойда

1.1 Математическая модель программы



Графическое представление операторов блок-схемы

Методы верификации Флойда
1.1 Математическая модель программы

- В программу могут входить несколько операторов одного и того же типа, помеченных одной и той же функцией.
- Каждый оператор программы состоит из
  - *метки оператора* ℓ_i и *тела оператора*;
  - для обеспечения уникальности операторов с одинаковым телом в рамках одной программы.
- Множество меток всех операторов программы P — Λ_P.

Методы верификации Флойда

1.1 Математическая модель программы

Блок-схемы

- Модель программы — *блок-схема* (V, N, E), где
  - V — конечное множество переменных программы,
  - N — конечное множество операторов блок-схемы,
  - $E \subseteq N \times \{ True, False, \varepsilon \} \times N$ — конечное множество связок блок-схемы, помеченных символами *True*, *False* или ε.
- Ориентированный граф блок-схемы
  - вершины — операторы программы,
  - помеченные ребра — связки программы.

Методы верификации Флойда

1.1 Математическая модель программы

Корректно-определенная блок-схема:

1. Имеет ровно один начальный оператор и не менее одного завершающего оператора.
2. Любой оператор находится на ориентированном пути от начального оператора к некоторому завершающему оператору.
3. Число связок, выходящих из каждого оператора, и их пометки соответствуют типу оператора:
    1. Из начального оператора, операторов присваивания и соединения выходит ровно 1 дуга, помеченная символом ε.
    2. Из условного оператора выходит ровно 2 дуги, причем одна из них помечена символом True, а другая – символом False.
    3. Из завершающего оператора не выходит ни одной дуги.
4. Число связок, входящих в каждый оператор, соответствует его типу:
    1. В начальный оператор не входит ни одна дуга.
    2. В оператор присваивания, условный и завершающий оператор входит ровно одна дуга.
    3. В оператор соединения входит не менее одной дуги.

Методы верификации Флойда

1.1 Математическая модель программы

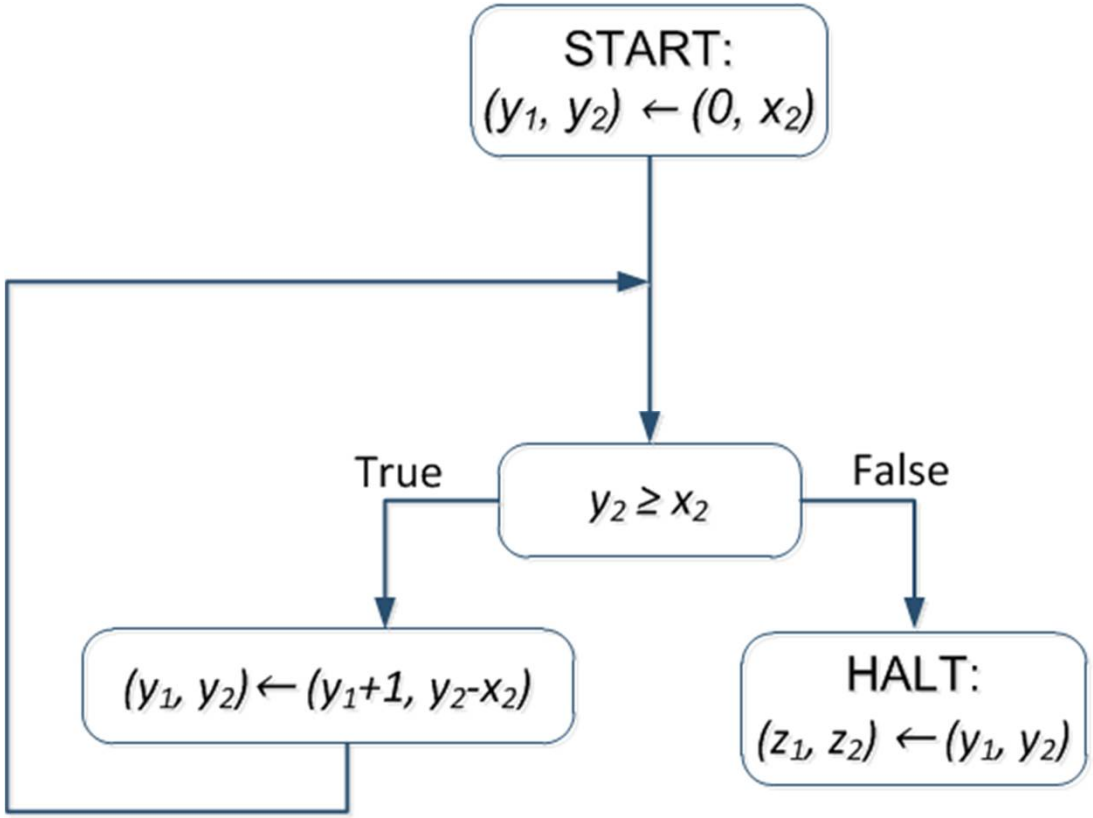
- Для каждого оператора *n* и символа *s* в корректно-определенной блок-схеме (*V*, *N*, *E*) существует не более одного оператора *n′*, что (*n*, *s*, *n′*) ∈ *E*.
  - Оператор *n′* — *последователь* оператора *n* по пометке *s*: *succ*(*n*, *s*).
- Функции: $(y_1, y_2, \ldots, y_Y) \leftarrow (f_1(\mathbf{x}, \mathbf{y}), f_2(\mathbf{x}, \mathbf{y}), \ldots, f_Y(\mathbf{x}, \mathbf{y}))$.

Методы верификации Флойда

1.1 Математическая модель программы

- $V = \{ x_1, x_2, y_1, y_2, z_1, z_2 \}$, D = $\mathbb{N}$.

Блок-схема программы целочисленного деления



Методы верификации Флойда

1.1 Математическая модель программы

Семантика блок-схем

- *Конфигурация* программы P — пара (ℓ, σ), где
  - $\ell \in \Lambda_P$ — метка текущего оператора программы,
  - $\sigma = (d_1, d_2, \ldots, d_{X+Y}) \in D_{\mathbf{x}}^{+} \times D_{\mathbf{y}}^{+}$ — вектор значений входных и промежуточных переменных программы.
- Пусть функция $f: D_{\mathbf{x}}^{+} \times D_{\mathbf{y}}^{+} \longrightarrow D_{\mathbf{y}}^{+}$; вычисляет новые значения переменных **y**, тогда
  - $\sigma[\mathbf{y} \leftarrow f(\mathbf{x}, \mathbf{y})]$ — вектор значений входных и промежуточных переменных программы, полученный путем замены в σ значений переменных **y** на $f(\sigma)$.

Методы верификации Флойда
1.1 Математическая модель программы

- *Вычисление* — (бес-)конечная последовательность конфигураций $C_1, \ldots, C_n, \ldots$ программы P, такая что

1. Метка первой конфигурации программы является меткой начального оператора.
2. Значения всех входных переменных программы являются определенными ($\neq \omega$) и неизменными во всех конфигурациях вычисления.
3. Значения промежуточных переменных в первой конфигурации равны ω (не определены).

Методы верификации Флойда

1.1 Математическая модель программы

Если метка ℓ_i текущего оператора конфигурации C_i — это метка

4. начального оператора START: $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$, то

$$C_{i+1} = succ(n_i, \varepsilon): \sigma_{i+1} = \sigma_i[\boldsymbol{y} \leftarrow f(\boldsymbol{x})];$$

5. оператора присваивания ASSIGN: $\boldsymbol{y} \leftarrow g(\boldsymbol{x}, \boldsymbol{y})$, то

$$C_{i+1} = succ(n_i, \varepsilon): \sigma_{i+1} = \sigma_i[\boldsymbol{y} \leftarrow g(\boldsymbol{x}, \boldsymbol{y})];$$

6. условного оператора TEST: $t(\boldsymbol{x}, \boldsymbol{y})$ и предикат $t(\boldsymbol{x}, \boldsymbol{y})$ при значениях переменных σ_i принимает значение *True*, то

$$C_{i+1} = succ(n_i, True): \sigma_{i+1} = \sigma_i;$$

7. условного оператора TEST: $t(\boldsymbol{x}, \boldsymbol{y})$ и предикат $t(\boldsymbol{x}, \boldsymbol{y})$ при значениях переменных σ_i принимает значение *False*, то

$$C_{i+1} = succ(n_i, False): \sigma_{i+1} = \sigma_i;$$

Методы верификации Флойда

1.1 Математическая модель программы

Если метка ℓ_i текущего оператора конфигурации C_i — это метка

8. оператора соединения JOIN, то

$C_{i+1} = succ(n_i, \varepsilon)$: $\sigma_{i+1} = \sigma_i$;

9. завершающего оператора HALT: $\boldsymbol{z} \leftarrow h(\boldsymbol{x}, \boldsymbol{y})$, то

C_i — последняя конфигурацией вычисления.

10. Если в конфигурации C_{i+1} значение какой-либо промежуточной переменной равно ω, то это последняя конфигурация вычисления.

Методы верификации Флойда

1.1 Математическая модель программы

- Три вида вычислений:
  1. конечные последовательности конфигураций, в последней конфигурации никакие значения переменных не равны ω;
  2. конечные последовательности конфигураций, в последней конфигурации значения некоторых переменных равны ω;
  3. бесконечные последовательности конфигураций.

- **Лемма 1.** *Для каждой блок-схемы P и вектора значений ее входных переменных* **x** *существует единственное вычисление, в первой конфигурации которого значения входных переменных равны* **x**.

Методы верификации Флойда

1.1 Математическая модель программы

- Каждой блок-схеме *P* мы поставим в соответствие функцию $M[P] : D_{\boldsymbol{x}} \longrightarrow D_{\boldsymbol{z}}^{+}$.
- Если вычисление блок-схемы *P* на векторе входных переменных $\boldsymbol{x}$ является конечным, в последней конфигурации которого нет ω, и завершается на операторе HALT,
  - то функция $M[P](\boldsymbol{x}) = h(\boldsymbol{x}, \boldsymbol{y_n})$, где
    - *h* – функция завершающего оператора последней конфигурации вычисления,
    - $\boldsymbol{y_n}$ – вектор значений промежуточных переменных из последней конфигурации вычисления.
  - иначе $M[P](\boldsymbol{x}) = \omega$.

Методы верификации Флойда
1.2 Математическая модель требований

- *Спецификация программы* —
  - математическая модель требований к верифицируемой программе;
    - требования к функциональности программ:
      - ограничения на результат вычисления программы в зависимости от значений ее входных данных.

Методы верификации Флойда
1.2 Математическая модель требований

Входной и выходной предикаты

- Спецификация Φ программы над переменными V :
  - входной предикат (*предусловие*) $\varphi : D_x \longrightarrow \{ True, False \}$
    - определяет, при каких значениях входных переменных требуется выполнение ограничений, описанных в выходном предикате.
  - выходной предикат (*постусловие*) $\psi : D_x \times D_z \longrightarrow \{ True, False \}$
    - определяет, какие значения выходных переменных программы являются допустимыми относительно значений входных переменных.

Методы верификации Флойда
1.3 Задача верификации

Частичная и полная корректность программ

- Пусть программа задана своей моделью в виде блок-схемы *P*, а ее спецификация Φ – предикатами φ и ψ.

- Программа *P* *частично корректна* относительно φ и ψ, если

$$\forall \boldsymbol{x}.\ \varphi(\boldsymbol{x}) = True \wedge M[P](\boldsymbol{x}) \neq \omega \Rightarrow \psi(\boldsymbol{x}, M[P](\boldsymbol{x})) = True.$$

$$\{\varphi\} P \{\psi\}$$

- Программа *P* *полностью корректна* относительно φ и ψ, если

$$\forall \boldsymbol{x}.\ \varphi(\boldsymbol{x}) = True \Rightarrow M[P](\boldsymbol{x}) \neq \omega \wedge \psi(\boldsymbol{x}, M[P](\boldsymbol{x})) = True.$$

$$[\varphi]\, P\, [\psi]$$

Методы верификации Флойда
1.3 Задача верификации

- **Лемма 2.**

  Пусть даны программа P и спецификация $\Phi = (\varphi, \psi)$.

  В этом случае $[\varphi]P[\psi]$ *тогда и только тогда, когда* $\{\varphi\}P\{\psi\}$ *и* $[\varphi]P[True]$.

  - для доказательства полной корректности программы необходимо и достаточно доказать ее частичную корректность и завершаемость.

- **Лемма 3.**

  Пусть дана программа P. Тогда для любых предикатов φ, ψ_1 *и* ψ_2 *выполнены следующие утверждения:*

  - *из* $\{\varphi\}P\{\psi_1\}$ *и* $\{\varphi\}P\{\psi_2\}$ *следует* $\{\varphi\}P\{\psi_1 \wedge \psi\}$,
  - *из* $[\varphi]P[\psi_1]$ *и* $[\varphi]P[\psi_2]$ *следует* $[\varphi]P[\psi_1 \wedge \psi_2]$.
    - По нескольким утверждениям о частичной и полной корректности можно получать новые утверждения.

Методы верификации Флойда
1.3 Задача верификации

- **Лемма 4.**

  Пусть дана программа P. Пусть предикаты φ, φ', ψ и ψ' таковы, что формулы $\varphi' \rightarrow \varphi$ и $\psi \rightarrow \psi'$ истинны. Тогда

  - *из $\{\varphi\}P\{\psi\}$ следует $\{\varphi'\}P\{\psi\}$ и $\{\varphi\}P\{\psi'\}$,*
  - *из $[\varphi]P[\psi]$ следует $[\varphi']P[\psi]$ и $[\varphi]P[\psi']$.*
    - Частичная и полная корректность сохраняются при замене входного предиката на более сильный и выходного на более слабый.

Методы верификации Флойда
1.3 Задача верификации

- Мы докажем полную корректность программы целочисленного деления P*div* относительно спецификации:
  - $\varphi = (x_1 \geq 0) \wedge (x_2 > 0)$
  - $\psi = (x_1 = z_1 x_2 + z_2) \wedge (0 \leq z_2 < x_2)$

  - Два этапа доказательства:

1. докажем, что программа частично корректна относительно более слабого входного предиката $\varphi_0 = (x_1 \geq 0) \wedge (x_2 \geq 0)$ и выходного предиката ψ.
2. докажем завершаемость программы на φ.

  - Из этого, по леммам 2 и 4, будет следовать требуемое утверждение.

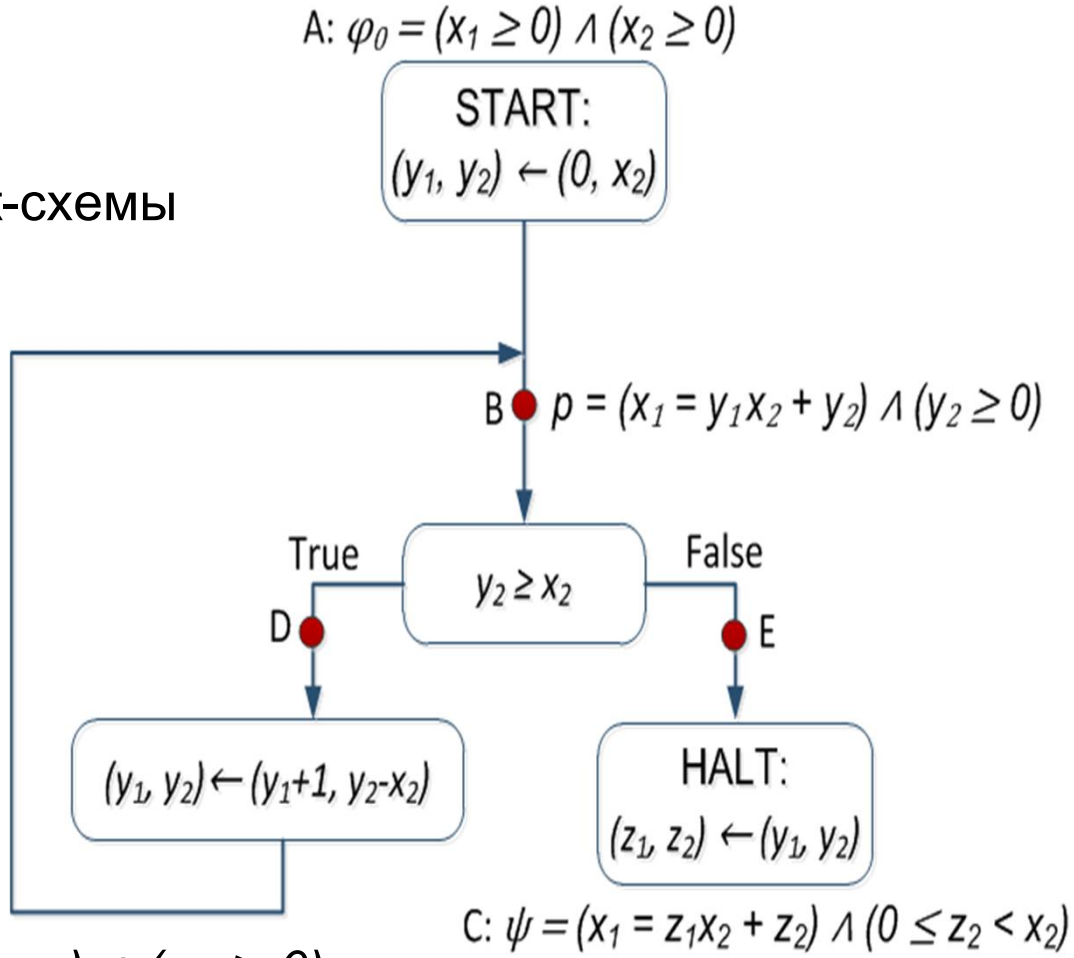
Методы верификации Флойда
1.3 Задача верификации

Частичная корректность.

Поставим в соответствие

- начальному оператору блок-схемы входной предикат φ_0,
- завершающему оператору выходной предикат ψ,
- ребру B между оператором соединения и условным оператором
- промежуточный предикат

$p(x_1, x_2, y_1, y_2) = (x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0)$



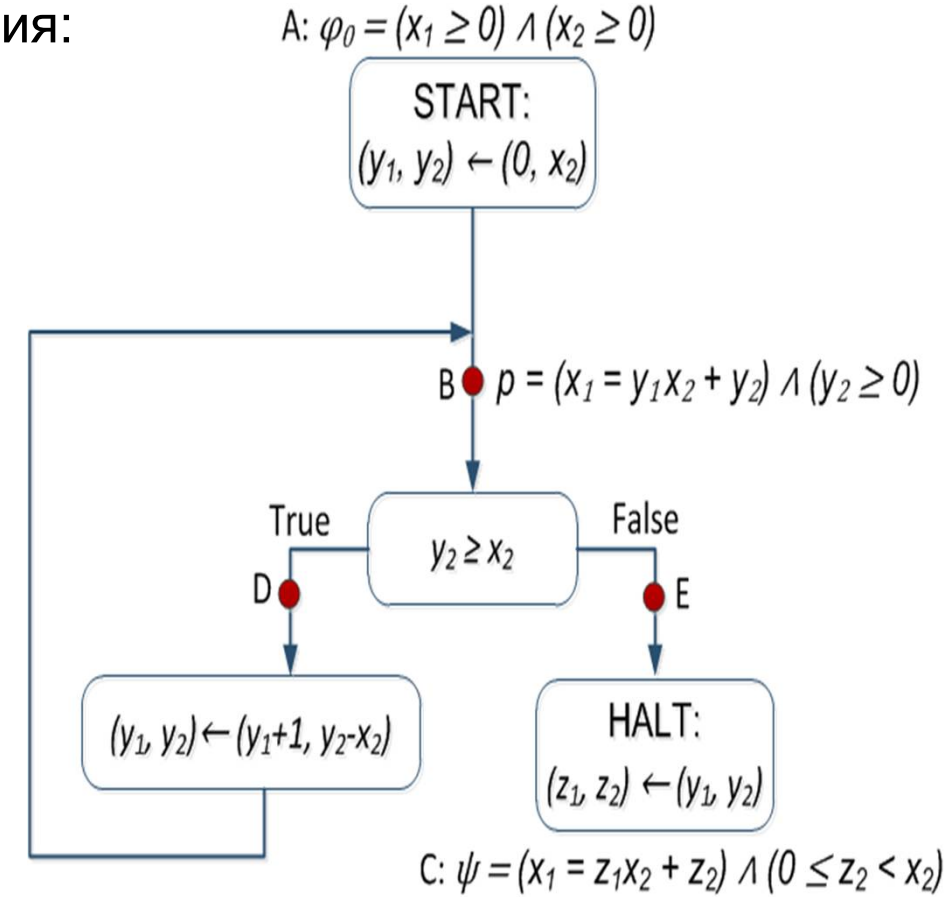
Методы верификации Флойда
1.3 Задача верификации

1. Рассмотрим путь от начального оператора до ребра *B*.

- После выполнения начального оператора переменные принимают следующие значения:

x_1	x_1
x_2	x_2
y_1	0
y_2	x_1

- Таким образом, $p(x_1, x_2, y_1, y_2) =$
  $(x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0) =$
  $(x_1 = 0 \cdot x_2 + x_1) \wedge (x_1 \geq 0) =$
  $(x_1 \geq 0)$.
  - $\varphi_0 \Rightarrow (x_1 \geq 0)$

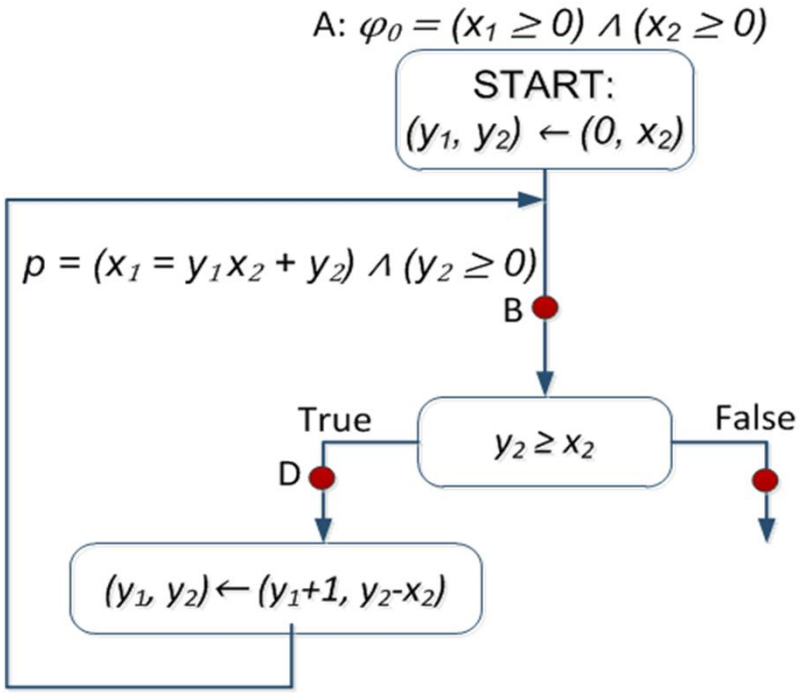


Методы верификации Флойда
1.3 Задача верификации

2. Пусть предикат *p* истинен в точке *B*. Рассмотрим путь *B-D-B*.

В точке *D* истинны

- предикат *p*
  - после выполнения условного оператора значения переменных не изменяются
- $(x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0) \wedge (y_2 \geq x_2)$
  - точка *D* лежит на ребре, помеченном символом *True*



- Докажем, что после выполнения последующего оператора присваивания предикат *p* также будет истинен:
- $D: (x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0) \wedge (y_2 \geq x_2)$
- $B: p(x_1, x_2, y_1 + 1, y_2 - x_2) = (x_1 = (y_1 + 1) x_2 + y_2 - x_2) \wedge (y_2 - x_2 \geq 0)$

Методы верификации Флойда
1.3 Задача верификации

- Нужно показать, что истинно следующее утверждение:

- $(x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0) \wedge (y_2 \geq x_2) \Rightarrow$

$$(x_1 = (y_1 + 1) x_2 + y_2 - x_2) \wedge (y_2 - x_2 \geq 0)$$

$$\Updownarrow$$

- $(x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0) \wedge (y_2 \geq x_2) \Rightarrow$

$$(x_1 = y_1 x_2 + y_2) \wedge (y_2 - x_2 \geq 0)$$

$$\Updownarrow$$

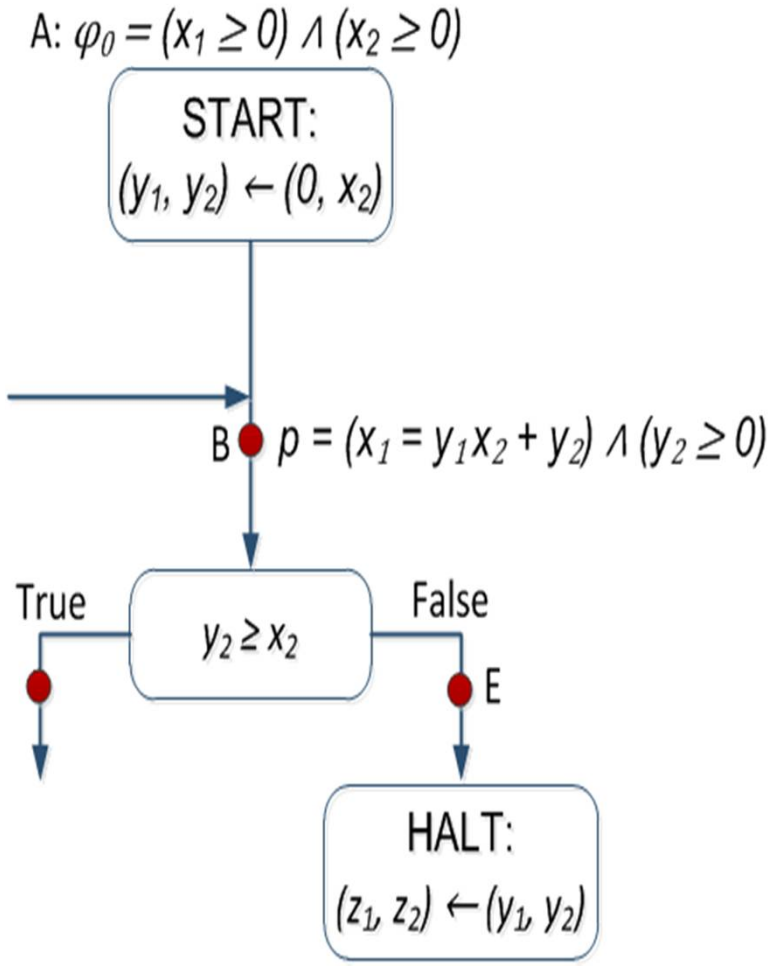
$$True$$

Методы верификации Флойда
1.3 Задача верификации

3. Пусть предикат *p* истинен в точке *B.* Рассмотрим путь от точки B до завершающего оператора.

- В точке *E* истинно утверждение:
  $(x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0) \wedge (y_2 < x_2)$
- После завершающего оператора будет истинен предикат ψ:
- E: $(x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0) \wedge (y_2 < x_2)$
- C: $\psi(x_1, x_2, y_1, y_2) =$
  $(x_1 = y_1 x_2 + y_2) \wedge (0 \leq y_2 < x_2)$

$$(x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0) \wedge (y_2 < x_2) \Rightarrow (x_1 = y_1 x_2 + y_2) \wedge (0 \leq y_2 < x_2)$$
$$\Updownarrow$$
$$True$$



Методы верификации Флойда
1.3 Задача верификации

- Из рассмотренных свойств программы следует, что
  - для любого конечного вычисления программы целочисленного деления
    - при значениях входных переменных, удовлетворяющих предусловию,
      - значения выходных переменных будут удовлетворять постусловию.
- Программа целочисленного деления является частично корректной относительно спецификации $\Phi_0 = (\varphi_0, \psi)$.

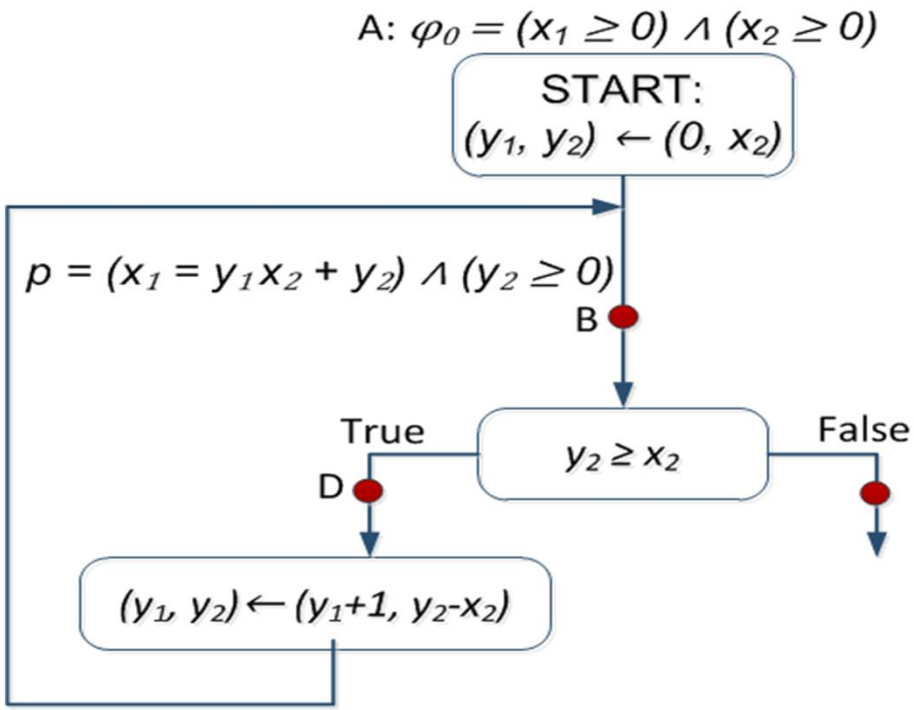
Методы верификации Флойда
1.3 Задача верификации

Завершаемость.

- Ранее доказано, что если значения входных переменных удовлетворяют предикату φ_0, то при всяком прохождении точки B верно, что $(x_1 = y_1 x_2 + y_2) \wedge (y_2 \geq 0)$
- Из этого следует, что если значения входных переменных удовлетворяют более сильному предикату φ, то при всяком прохождении точки B так же верно, что $(y_2 \geq 0)$.
- Поскольку значения входных переменных не изменяются в ходе выполнения программы, предикат φ является истинным в любой промежуточной точке.
- Следовательно, что в точке B верно $(y_2 \geq 0) \wedge (x_2 > 0)$.

Методы верификации Флойда
1.3 Задача верификации

- В цикле *B-D-B* значение переменной y_2 уменьшится на положительную величину x_2, но останется неотрицательным.
- Цикл *B-D-B* не может выполняться бесконечное число раз
  - если верно φ,
  - не существует бесконечной убывающей последовательности неотрицательных целых чисел.
- Программа целочисленного деления завершается на входном предикате φ.



- Мы доказали, что $\{\varphi_0\}$ *Pdiv* $\{\psi\}$ и $[\varphi]$ *Pdiv* $[True]$.
- Отсюда, по леммам 2 и 4, следует $[\varphi]$ *Pdiv* $[\psi]$.

Методы верификации Флойда

1.4 Доказательство корректности программ

- Определим методы доказательства корректности программ, представленных в виде блок-схем, в общем случае.
  1. Частичная корректность программы.
     - *Метод индуктивных утверждений Флойда*.
  2. Доказательство завершаемости программы.
     - *Метод фундированных множеств Флойда*.

Методы верификации Флойда
1.4 Доказательство корректности программ

- Рассмотрим *пути в блок-схемах* $\alpha = n_1 - e_1 \rightarrow n_2 - e_2 \rightarrow \ldots - e_k \rightarrow n_{k+1}$.
  - связка e_i *входит в оператор* e_{i+1},
  - операторы $n_2, n_3, \ldots, n_k$ — внутри пути.
  - суффикс пути $\alpha^m = n_m - e_m \rightarrow \ldots - e_k \rightarrow n_{k+1}$ для $m \in \{1, \ldots, k\}$.
- *Предикат (допустимости) пути* α $R_\alpha(\mathbf{x},\mathbf{y}):D_x \times D_y \longrightarrow \{True, False\}$
  - определяет, какое должно быть значение входных и промежуточных переменных в начале пути, чтобы дальнейшее вычисление шло по пути α.
- *Функция пути* α $r_\alpha(\mathbf{x},\mathbf{y}) : D_x \times D_y \longrightarrow D_y$.
  - определяет, как изменятся значения промежуточных переменных в результате исполнения последовательности операторов блок-схемы, находящихся внутри пути α.
- Для суффиксов пути $R_\alpha^m(\mathbf{x},\mathbf{y})$ и $r_\alpha^m(\mathbf{x},\mathbf{y})$.

Методы верификации Флойда

1.4 Доказательство корректности программ

- *Метод обратных подстановок*
  - для составления формул $R_\alpha(\boldsymbol{x},\boldsymbol{y})$ и $r_\alpha(\boldsymbol{x},\boldsymbol{y})$.
- $R_\alpha(\boldsymbol{x},\boldsymbol{y}) = R_\alpha^1(\boldsymbol{x},\boldsymbol{y})$ и $r_\alpha(\boldsymbol{x},\boldsymbol{y}) = r_\alpha^1(\boldsymbol{x},\boldsymbol{y})$.
- Предикаты $R_\alpha^m(\boldsymbol{x},\boldsymbol{y})$ и функции $r_\alpha(\boldsymbol{x},\boldsymbol{y})$ определим индукцией по m.
- *Базис индукции:*
  - $R_\alpha^k(\boldsymbol{x},\boldsymbol{y}) = True$, $r_\alpha^k(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{y}$.
- *Индуктивное предположение:*
  - Зафиксируем некоторое m < k.
  - Пусть определены $R_\alpha^{m+1}(\boldsymbol{x},\boldsymbol{y})$ и $r_\alpha^{m+1}(\boldsymbol{x},\boldsymbol{y})$.

Методы верификации Флойда

1.4 Доказательство корректности программ

- *Индуктивный переход:*
- Можно определить $R_\alpha^m(\mathbf{x},\mathbf{y})$ и $r_\alpha^m(\mathbf{x},\mathbf{y})$ в зависимости от оператора n_{m+1}:
- n_{m+1} = ASSIGN: $\mathbf{y} \leftarrow g(\mathbf{x}, \mathbf{y})$:

  $R_\alpha^m(\mathbf{x},\mathbf{y}) = R_\alpha^{m+1}(\mathbf{x}, g(\mathbf{x}, \mathbf{y})) \wedge g(\mathbf{x}, \mathbf{y}) \neq \omega$ $\quad r_\alpha^m(\mathbf{x},\mathbf{y}) = r_\alpha^{m+1}(\mathbf{x}, g(\mathbf{x}, \mathbf{y}))$.
- n_{m+1} = TEST: $t(\mathbf{x}, \mathbf{y})$ и связка $e_m \leftarrow True$:

  $R_\alpha^m(\mathbf{x},\mathbf{y}) = R_\alpha^{m+1}(\mathbf{x}, \mathbf{y})) \wedge t(\mathbf{x}, \mathbf{y})$ $\quad r_\alpha^m(\mathbf{x},\mathbf{y}) = r_\alpha^{m+1}(\mathbf{x}, \mathbf{y}))$.
- n_{m+1} = TEST: $t(\mathbf{x}, \mathbf{y})$ и связка $e_m \leftarrow False$:

  $R_\alpha^m(\mathbf{x},\mathbf{y}) = R_\alpha^{m+1}(\mathbf{x}, \mathbf{y})) \wedge \neg\, t(\mathbf{x}, \mathbf{y})$ $\quad r_\alpha^m(\mathbf{x},\mathbf{y}) = r_\alpha^{m+1}(\mathbf{x}, \mathbf{y}))$.
- n_{m+1} = JOIN:

  $R_\alpha^m(\mathbf{x},\mathbf{y}) = R_\alpha^{m+1}(\mathbf{x}, \mathbf{y}))$ $\quad r_\alpha^m(\mathbf{x},\mathbf{y}) = r_\alpha^{m+1}(\mathbf{x}, \mathbf{y}))$.

Методы верификации Флойда

1.4 Доказательство корректности программ

- Начальный и завершающий оператор
  - не могут встречаться внутри пути;
  - в них могут входить и выходить первая или последняя связка пути.
- Начало первой связки пути α START: $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$:

$$R_\alpha'(\boldsymbol{x}) = R_\alpha(\boldsymbol{x}, f(\boldsymbol{x}, \boldsymbol{y})) \wedge f(\boldsymbol{x}) \neq \omega$$

$$r_\alpha'(\boldsymbol{x}) = r_\alpha(\boldsymbol{x}, f(\boldsymbol{x}, \boldsymbol{y})).$$

- Конец последней связки пути α HALT: $\boldsymbol{z} \leftarrow h(\boldsymbol{x}, \boldsymbol{y})$:

$$R_\alpha''(\boldsymbol{x}) = R_\alpha(\boldsymbol{x},\boldsymbol{y})) \wedge h(\boldsymbol{x}, r_\alpha(\boldsymbol{x},\boldsymbol{y})) \neq \omega$$

$$r_\alpha''(\boldsymbol{x}) = h(\boldsymbol{x}, r_\alpha(\boldsymbol{x},\boldsymbol{y})).$$

Методы верификации Флойда

1.4 Доказательство корректности программ

Индуктивные утверждения. Частичная корректность.

Пусть P – блок-схема, а $\Phi = (\varphi, \psi)$ – ее спецификация.

- *Шаг 1. Точки сечения.*

- Выберем подмножество связок блок-схемы — *точки сечения*.
  - каждый цикл в блок-схеме должен содержать, по крайней мере, одну точку сечения.
- *Промежуточные базовые пути*

- *Начальные базовые пути*

- *Конечные базовые пути*

- *Простые базовые пути*

Методы верификации Флойда
1.4 Доказательство корректности программ

- *Шаг 2. Индуктивные утверждения*.
- Индуктивное утверждение для каждой точки сечения *i*:
  - предикат $p_i(x, y)$, который характеризует отношение между переменными блок-схемы при прохождении связки *i*.
- Входной предикат $\varphi(x)$ сопоставляется
  - начальному оператору блок-схемы.
- Выходной предикат $\psi(x, z)$ сопоставляется
  - всем завершающим операторам.

Методы верификации Флойда
1.4 Доказательство корректности программ

- *Шаг 3. Условия верификации.*
- *Условия верификации* для каждого промежуточного базового пути α из точки сечения *i* в точку сечения *j*:

$$\forall \boldsymbol{x} \in D_{\boldsymbol{x}}\ \forall \boldsymbol{y} \in D_{\boldsymbol{y}}\ [\ \varphi(\boldsymbol{x}) \wedge p_i(\boldsymbol{x},\boldsymbol{y}) \wedge R_\alpha(\boldsymbol{x},\boldsymbol{y}) \Rightarrow p_j\ (\boldsymbol{x},\ r_\alpha(\boldsymbol{x},\boldsymbol{y}))\ ]$$

- Если предикат $p_i(\boldsymbol{x},\boldsymbol{y})$ истинен для некоторых значений ***x*** и ***y***,
- и эти значения такие, что, начиная из точки сечения *i,* вычисление пойдет по пути α и успешно достигнет конца пути,
- то предикат $p_j(\mathbf{x}, \mathbf{y})$ будет истинен для значений переменных **x** и **y**, после прохождения по пути α.

Методы верификации Флойда
1.4 Доказательство корректности программ

- Для начального базового пути α до точки сечения j:

$$\forall \boldsymbol{x} \in D_{\boldsymbol{x}} \, [\, \varphi(\boldsymbol{x}) \wedge R_\alpha'(\boldsymbol{x},\boldsymbol{y}) \Rightarrow p_j(\boldsymbol{x}, r_\alpha'(\boldsymbol{x},\boldsymbol{y})) \,]$$

- Для каждого конечного базового пути α из точки сечения i:

$$\forall \boldsymbol{x} \in D_{\boldsymbol{x}} \; \forall \boldsymbol{y} \in D_{\boldsymbol{y}} \, [\, \varphi(\boldsymbol{x}) \wedge p_i(\boldsymbol{x},\boldsymbol{y}) \wedge R_\alpha''(\boldsymbol{x},\boldsymbol{y}) \Rightarrow \psi(\boldsymbol{x}, r_\alpha''(\boldsymbol{x},\boldsymbol{y})) \,]$$

- Для простого базового пути α:

$$\forall \boldsymbol{x} \in D_{\boldsymbol{x}} \; \forall \boldsymbol{y} \in D_{\boldsymbol{y}} \, [\, \varphi(\boldsymbol{x}) \wedge R_\alpha'(\boldsymbol{x},\boldsymbol{y}) \wedge h(\boldsymbol{x}, r_\alpha'(\boldsymbol{x})) \neq \omega \Rightarrow \psi(\boldsymbol{x}, h(\boldsymbol{x}, r_\alpha'(\boldsymbol{x})) \,]$$

Лемма 5.

- Пусть все условия верификации истинны. Пусть дано вычисление блок-схемы P, входные переменные которого удовлетворяют входному предикату φ.
  - Тогда для каждого прохода вычисления $C_k - C_{k+1}$ через точку сечения i, предикат $p_i(\boldsymbol{x}, \boldsymbol{y})$ будет истинен на значениях переменных $\boldsymbol{x}$ и $\boldsymbol{y}$ в конфигурации C_k.

Методы верификации Флойда

1.4 Доказательство корректности программ

Теорема 1. (Метод индуктивных утверждений Флойда)

- *Пусть даны блок-схема P и ее спецификация $\Phi = (\varphi, \psi)$. Выполним следующие действия:*

1. *Выберем точки сечения;*
2. *Найдем подходящий набор индуктивных утверждений;*
3. *Построим условия верификации для всех базовых путей.*

- *Если все условия шага 3 истинны, то блок-схема P частично корректна относительно спецификации Φ.*

- Шаг 1 и 3 могут быть выполнены относительно автоматически.
- Выбор подходящего набора индуктивных утверждений требует хорошего понимания функционирования программы и поэтому сложно поддается автоматизации.

Методы верификации Флойда
1.4 Доказательство корректности программ

Метод фундированных множеств Флойда. Завершаемость.

- *Частично-упорядоченное множество* $(W, \prec)$, где
  - W — непустое множество;
  - $\prec$ — бинарное отношение, такое что для всех $a, b, c \in W$ оно
    1. транзитивно: $(a \prec b) \wedge (b \prec c) \Rightarrow (a \prec c)$,
    2. асимметрично: $(a \prec b) \Rightarrow \neg(b \prec a)$,
    3. иррефлексивно: $\neg (a \prec a)$.
- *Фундированное множество* $(W, \prec)$
  - частично-упорядоченное множество, такое что не существует бесконечно убывающей последовательности $a_1 \succ a_2 \succ a_3 \succ \ldots$

- Пример: множество натуральных чисел с отношением порядка <.

Методы верификации Флойда
1.4 Доказательство корректности программ

Пусть P – блок-схема, а φ – ее входной предикат.

- *Шаг 1. Точки сечения.*
- Выберем:
  - *множество точек сечения* блок-схемы таким образом, чтобы каждый цикл в блок-схеме содержал, по крайней мере, одну точку сечения,
  - некоторое *фундированное множество* $(W, \prec)$.
  - индуктивное утверждение $q_i(\boldsymbol{x}, \boldsymbol{y})$ для каждой точки сечения i.
- Для начальных и промежуточных базовых путей построим условия верификации индуктивных утверждений $q_i(\boldsymbol{x}, \boldsymbol{y})$ и докажем их истинность.

Методы верификации Флойда

1.4 Доказательство корректности программ

- *Шаг 2. Оценочные функции.*
- *Оценочная функция* для каждой точки сечения *i*

$$u_i(x,y): D_x \times D_y \longrightarrow W_i, \text{ где } W_i \supseteq W.$$

- Условие корректности определения оценочной функции:

$$\forall \boldsymbol{x} \in D_x \; \forall \boldsymbol{y} \in D_y \; [\; \varphi(\boldsymbol{x}) \wedge q_i(\boldsymbol{x},\boldsymbol{y}) \Rightarrow u_i(x,y) \in W \;].$$

Методы верификации Флойда
1.4 Доказательство корректности программ

- *Шаг 3. Условия завершимости*.
- Условие завершимости для каждого промежуточного базового пути α из точки сечения *i* в точку сечения *j*:

  $\forall \mathbf{x} \in D_{\mathbf{x}}\ \forall \mathbf{y} \in D_{\mathbf{y}}\ [\ \varphi(\mathbf{x}) \wedge q_i(\mathbf{x},\mathbf{y}) \wedge R_\alpha(\mathbf{x},\mathbf{y}) \Rightarrow (u_i(\mathbf{x},\mathbf{y}) \succ u_j(\mathbf{x}, r_\alpha(\mathbf{x},\mathbf{y}))\ ]$
- Если предикат $q_i(\mathbf{x},\mathbf{y})$ истинен для некоторых значений *x* и *y*, и
- эти значения такие, что, начиная из точки сечения *i*, вычисление пойдет по пути α, то
- результат оценочной функции $u_j(\mathbf{x},\mathbf{y})$ на значениях переменных *x* и *y* после прохождения по пути α будет меньше
- результата оценочной функции $u_i(\mathbf{x},\mathbf{y})$ на исходных значениях.

Методы верификации Флойда
1.4 Доказательство корректности программ

- *Шаг 4. Условия успешности вычисления функций.*
- Необходимо показать, что функции, приписанные операторам блок-схемы, которые могут равняться ω, не вызываются с такими аргументами, при которых они равны ω.

- Оператор START: $\boldsymbol{y} \leftarrow f(\boldsymbol{x})$:

$$\forall \boldsymbol{x} \in D_{\boldsymbol{x}} \, [\, \varphi(\boldsymbol{x}) \wedge f(\boldsymbol{x}) \neq \omega \,]$$

- Оператор ASSIGN: $\boldsymbol{y} \leftarrow g(\boldsymbol{x}, \boldsymbol{y})$
  - Для каждого промежуточного базового пути α из точки сечения i, завершающегося перед оператором ASSIGN:

$$\forall \boldsymbol{x} \in D_{\boldsymbol{x}} \; \forall \boldsymbol{y} \in D_{\boldsymbol{y}} \, [\, \varphi(\boldsymbol{x}) \wedge q_i(\boldsymbol{x},\boldsymbol{y}) \wedge R_\alpha(\boldsymbol{x},\boldsymbol{y}) \Rightarrow g(\boldsymbol{x}, r_\alpha(\boldsymbol{x},\boldsymbol{y})) \neq \omega \,].$$

Методы верификации Флойда
1.4 Доказательство корректности программ

- Оператор TEST: *t* (**x**, **y**)
  - Для каждого промежуточного базового пути α из точки сечения *i*, завершающегося перед оператором TEST :

$$\forall \mathbf{x} \in D_x \; \forall \mathbf{y} \in D_y \; [\; \varphi(\mathbf{x}) \wedge q_i(\mathbf{x},\mathbf{y}) \wedge R_\alpha(\mathbf{x},\mathbf{y}) \Rightarrow t(\mathbf{x}, r_\alpha(\mathbf{x},\mathbf{y})) \neq \omega \;].$$

- Оператор HALT: $\mathbf{z} \leftarrow h(\mathbf{x}, \mathbf{y})$
  - Для каждого промежуточного базового пути α из точки сечения *i*, завершающегося перед оператором HALT:

$$\forall \mathbf{x} \in D_x \; \forall \mathbf{y} \in D_y \; [\; \varphi(\mathbf{x}) \wedge q_i(\mathbf{x},\mathbf{y}) \wedge R_\alpha(\mathbf{x},\mathbf{y}) \Rightarrow h(\mathbf{x}, r_\alpha(\mathbf{x},\mathbf{y})) \neq \omega \;].$$

- Для каждого простого базового пути α из оператора START с функцией *f* в оператор HALT с функцией *h*:

$$\forall \mathbf{x} \in D_x \; [\; \varphi(\mathbf{x}) \wedge R'_\alpha(\mathbf{x},\mathbf{y}) \Rightarrow h(\mathbf{x}, r'_\alpha(\mathbf{x},\mathbf{y})) \neq \omega \;].$$

Методы верификации Флойда

1.4 Доказательство корректности программ

- **Теорема 2. (Метод фундированных множеств Флойда)**
- *Пусть даны блок-схема P и ее входной предикат α. Выполним следующие действия:*

1. *Выберем точки сечения с подходящим набором индуктивных утверждений и фундированное множество;*
2. *Выберем подходящий набор оценочных функций;*
3. *Построим условия верификации для всех базовых начальных и промежуточных путей, условия корректности определения всех оценочных функций и условия завершимости для всех промежуточных базовых путей.*
4. *Построим условия успешности вычисления функций, приписанных всем операторам START, ASSIGN, TEST и HALT.*

- *Если все условия на шагах 3 и 4 истинны, то блок-схема P успешно завершается на α.*

Методы верификации Флойда
1.4 Доказательство корректности программ

Замечания к методам Флойда

- При доказательстве полной корректности блок-схемы на этапе доказательства завершаемости можно использовать *те же точки сечения и индуктивные утверждения*, которые были использованы для доказательства частичной корректности.
  - сокращение объема доказательства полной корректности.
  - стоит избегать громоздких индуктивных утверждений, лучше строить новые.
  - можно использовать следствия индуктивных утверждений.

Методы верификации Флойда
1.4 Доказательство корректности программ

- Далеко не всегда удается сразу предложить набор индуктивных утверждений, достаточный для доказательства частичной корректности или завершаемости.
  - если обнаружится, что выбранного индуктивного утверждения недостаточно, необходимо начать доказательство частичной корректности сначала.
  - если этим пренебречь, то получится неверное доказательство полной корректности,
    - например, измененные индуктивные утверждения не являются корректными для путей, уже ранее рассмотренных в этом доказательстве.

Метод верификации Хоара

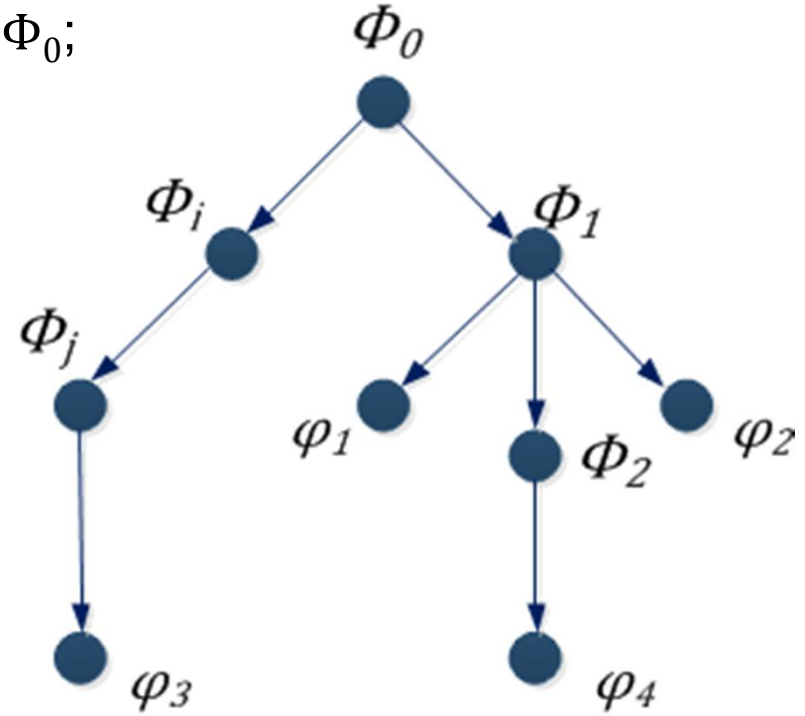
- Тройка Хоара — выражение вида $\varphi\{\pi\}\psi$, где
  - φ, ψ — формулы логики предикатов
  - π — императивная программа
    - ASS: x ← t
    - COMP: $\pi_1; \pi_2$
    - IF: if *C* then π_1 else π_2 fi
    - WHILE: while *C* do π_2 od
- Программа частично корректна относительно предусловия φ и постусловия ψ, если при условии выполнимости φ после исполнения программы π выполнено ψ, либо программа π не завершается.

Метод верификации Хоара

- Правила вывода Хоара:
- ASS: $\dfrac{\varphi\{x/t\}\{x \leftarrow t\}\varphi}{\mathbf{true}}$
- CONS: $\dfrac{\varphi\{\pi\}\psi}{\varphi\rightarrow\varphi',\ \varphi'\{\pi\}\psi',\psi'\rightarrow\psi}$
- COMP: $\dfrac{\varphi\{\pi_1;\pi_2\}\psi}{\varphi\{\pi_1\}\chi,\ \chi\{\pi_2\}\psi}$
- IF: $\dfrac{\varphi\ \{\ \text{if}\ c\ \text{then}\ \pi_1\ \text{else}\ \pi_2\ \text{fi}\ \}\ \psi}{(\varphi\wedge C)\{\pi_1\}\psi,\ (\varphi\wedge\neg C)\{\pi_2\}\psi}$
- WHILE: $\dfrac{\varphi\ \{\ \text{while}\ c\ \text{do}\ \pi\ \text{od}\ \}(\varphi\wedge\neg C)}{(\varphi\wedge C)\{\pi\}\varphi}$

Метод верификации Хоара

- Вывод в логике Хоара тройки $\Phi_0=\varphi_0\{\pi_0\}\psi_0$ — это корневое дерево, вершинами которого служат тройки и формулы логики предикатов , так что
  - корень дерева — это тройка Φ_0;
  - $\Phi_i \longrightarrow \Phi_j \Leftrightarrow \frac{\Phi_i}{\Phi_j}$
  - листья дерева — формулы логики предикатов.



Метод верификации Хоара

- Вывод тройки $\Phi=\varphi\{\pi\}\psi$ в логике Хоара называется успешным, если дерево вывода является конечным, и все его листья — это истинные формулы логики предикатов.

- Теорема корректности.
- Для любого правила вывода логики Хоара верно следующее утверждение: если истинны все формулы следствия правила, то истинна и его посылка.

- Следствие.
- Если тройка $\Phi = \varphi\{\pi\}\psi$ имеет успешный вывод, то программа π частично корректна относительно предусловия φ и постусловия ψ.

Метод верификации Хоара

- Для построения успешного вывода необходимо иметь эффективный доказатель для проверки истинности формул.
- CONS: $\dfrac{\varphi\{\pi\}\psi}{\varphi\to\varphi',\ \varphi'\{\pi\}\psi',\psi'\to\psi}$
- Что выбирать в качестве φ' и ψ' ?
- Пусть заданы императивная программа π и постусловие ψ. Тогда формула φ_0 называется слабейшим предусловием для программы π и постусловия ψ, если
  - $\varphi_0\{\pi\}\psi$
  - для любой формулы φ, если $\varphi_0\{\pi\}\psi$, то $\varphi\to\varphi_0$
- Слабейшее предусловие обозначим как *wpr*(π, ψ).

Метод верификации Хоара

- Теорема
- $\varphi\{\pi\}\psi \Leftrightarrow wpr(\pi, \psi)\{\pi\}\psi$ и $\varphi \rightarrow wpr(\pi, \psi)$
- Задача построения успешного вывода сводится к задаче вычисления $wpr(\pi, \psi)$.

- Теорема.
- $wpr(\ x \leftarrow t,\ \psi) = \{x/t\}$,
- $wpr(\pi_1; \pi_2, \psi) = wpr(\pi_1, wpr(\pi_2, \psi))$,
- $wpr(\ \text{if } C \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi},\ \psi) = C \wedge wpr(\pi_1, \psi) \vee \neg C \wedge wpr(\pi_2, \psi)$.

Метод верификации Хоара

- Оператор цикла.
- Верифицировать этот оператор можно с помощью правила
- WHILE-GEN: $\dfrac{\varphi \{ \text{while } c \text{ do } \pi \text{ od } \}(\psi)}{\varphi \to \chi \ (\chi \wedge C)\{\pi\}(\chi \wedge \neg C) \to \psi}$
- Это правило требует введения вспомогательной формулы , которая называется инвариантом цикла, который зависит от программы π и условия C.
- Автоматическая генерация инвариантов цикла — это ключевая задача в решении проблемы автоматической верификации программ.

Проверка на модели

Введение

- Проверка на модели — автоматическая техника верификации различных систем с (бес-)конечным числом состояний.
- Имеет ряд преимуществ относительно других подходов к верификации, основанных на тестировании, симуляции, дедукции и т.п.
- Этот метод успешно использовался в верификации очень сложных электронных схем, коммуникационных протоколах и современных полупроводниковых устройств и процессоров.
- Основная проблема проверки на модели — это «взрыв пространства состояний».
  - Он возникает в случае, когда система имеет много параллельно взаимодействующих компонент и/или содержит структуры данных, которые принимают много различных значений.

Введение

- Проверка на модели vs другие методы верификации
- Использование этого метода для верификации сложных систем
- Различные подходы к проблеме взрыва пространства состояний
- Моделирование систем
- Логики CTL, LTL, CTL*
- Проверка на модели для CTL
- Проверка на модели для LTL

2. Верификация программных продуктов и систем

- Методы проверки правильности программных систем
  - Имитационное моделирование
    - тестируется модель продукта
  - Тестирование
    - тестируется собственно продукт

  - Проверяются результаты работы системы на ограниченном количестве входных данных.
    - +. Понятность и экономичность.
    - -. Отсутствие полноты.

2. Верификация программных продуктов и систем

- Методы проверки правильности программных систем
  - Дедуктивный анализ
    - с помощью аксиом и правил вывода доказывается корректность программы, аннотированной формулами некоторой логики
    - +. Применим к системам с произвольно большим количеством состояний. Полнота выявления ошибок.
    - -. Медленный метод. Плохо автоматизируемый. Часто неразрешимый.
  - Проверка моделей
    - обход заданных (всех) состояний модели системы с проверкой выполнимости в них спецификации, определённой формулой какой-либо логики.
    - +. Полная автоматизируемость. Полнота выявления ошибок.
    - -. Применим только к системам с конечным числом состояний (или сводимым к ним).

2. Верификация программных продуктов и систем

- Методы проверки правильности программных систем
  - Синтез дедуктивного метода и проверки моделей.
    - Фрагменты системы, имеющие конечное число состояний могут проверяться автоматически.

3. Процесс проверки на модели

- Моделирование
  - Приведение программной системы к формальному виду, подходящему для автоматической проверки.
    - Компиляция, трансляция.
    - Абстрагирование.
- Спецификация
  - Задание свойств программной системы, которые необходимо проверить.
    - Логики: темпоральные, динамические, эпистемические, деонтические и многие другие.
    - Полнота спецификации.
      - Безопасность: ничего плохого никогда не случится.
      - Живость: что-нибудь хорошее обязательно получится.
- Верификация

3. Процесс проверки на модели

- Моделирование
- Спецификация
- Верификация
  - Проверка соответствия модели системы её спецификации.
    - Полностью автоматическая.
    - Анализ результатов
      - Контрпример (ошибочная трасса)
      - Ложный контрпример
    - Большой размер модели.
      - Абстрагирование
    - Уточнение модели

Модальные логики

- Темпоральные логики
  - LTL, CTL, CTL*
  - Утверждения о развитии событий системы во времени
    - Что-то когда-то обязательно случится.
    - Что-то, возможно, будет всегда.
    - Давным-давно наверняка было кое-что, пока не случилось нечто.
- Динамические логики

4. Логики спецификаций и проверка на модели

Модальные логики

- Темпоральные логики
- Динамические логики
  - PDL, PDL*, μ-исчисление
  - Утверждения о развитии событий системы в результате некоторых действий.
    - Если сделать кое-что, то потом обязательно получится что-то.
    - Если многократно что-то делать, то, возможно, кое-что и выйдет.

4. Логики спецификаций и проверка на модели

Модальные логики

- Темпоральные логики
- Динамические логики
- Другие логики
  - Эпистемические
    - PLK, PLC
    - Утверждения о знаниях.
      - Я знаю, что он знает, что они в курсе, что я знаю секрет.
  - Деонтические
    - SDL, DDL
    - Утверждения об обязанностях
      - Нечто должно иметь место.
  - Веры
  - Комбинации логик

4. Логики спецификаций и проверка на модели

Модальные логики

- Темпоральные логики
- Динамические логики
- Другие логики
  - Эпистемические
  - Деонтические
  - Веры
    - PLB, PPLB
    - Утверждения об убеждениях
      - Я верю, что что-нибудь истинно.
  - Комбинации логик
    - Всевозможные утверждения
      - Когда-нибудь после некоторого действия я узнаю, что раньше всегда должен был поступать не так, а иначе, чтобы поверить, что ничего не изменилось бы оттого, что я мог знать что-то очевидное.

5. Символические алгоритмы

- Автоматическая проверка на модели:
  - Программы, её реализующие:
    - Алгоритмы + структуры данных = программы.
- Структуры данных:
  - Системы переходов
    1. состояния
    2. переходы между ними
  - Явное представление систем переходов
    1. перечисление
    2. списки смежности
  - Символическое представление систем переходов
    1. компактно заданное множество состояний (формула, интервал)
    2. компактно заданные переходы (формула, функция)

5. Символические алгоритмы

- Символические представления
  - Булевские модели
    - Бинарные Разрешающие Диаграммы BDD (Binary Decision Diagram)
      - Множество состояний, переход – булевская формула
      - Закодировать булевскую формулу как BDD
  - Целочисленные модели
    - Формулы арифметики Пресбургера
      - Множество состояний, переход – формула арифметики Пресбургера
    - Аффинные множества
      - Множество состояний – n-мерный интервал
      - Переход – целочисленная функция

6. Редукция частичных порядков

- Асинхронные системы
  - Действия модулей не согласованы, нет глобальных часов.
- Независимость событий
  - События (переходы) независимы, если состояние, в которое приходит система в результате их осуществления, не зависит от порядка этих событий.
- Интерливинг (чередование)
  - Параллельные события в последовательности вычислений располагаются в произвольном порядке.
- Редукция частичных порядков
  - Последовательности вычислений разбиваются на независимые классы эквивалентности.
- Разновидности
  - Упорные множества (Валмари), устойчивые множества (Годфруа), достаточные множества (Пелед).
  - Метод развёртки (МакМиллан), оцепеневшие множества (Годфруа).

7. Другие подходы к проблеме взрыва состояний

- Композиционный вывод
  - Модульность системы
    - Локальные свойства → спецификация всей системы.
  - Сильная зависимость модулей
    - Локальные свойства с предположениями → спецификация всей системы.
- Абстракция
  - Множество реальных значений данных системы кодируется множеством абстрактных значений.
- Симметрия
- Индукция

7. Другие подходы к проблеме взрыва состояний

- Композиционный вывод
- Абстракция
- Симметрия
  - В системе параллельно действуют одинаковые процессы.
- Индукция
  - Параметризованные процессы
    - Инвариант поведения — спецификация
    - Индуктивно показать, что инвариант верен для всех значений параметра.

Моделирование систем

- Формальная модель системы для верификации свойств
  - Всё необходимое
  - Ничего лишнего
    - Цифровые схемы:
      - Логические ячейки и булевые значения
      - Не уровни напряжения
    - Коммуникационные протоколы:
      - Обмен сообщениями
      - Не содержание сообщений

Моделирование систем

- Реагирующие системы (Reactive systems)
  - Взаимодействие с окружением
  - Бесконечно долгая работа
  - Состояние
    - мгновенное описание системы, значение переменных системы в данный момент времени.
  - Переход между состояниями
    - Изменение состояний
    - Состояние до и состояние после
  - Вычисление
    - (Бес-)конечная последовательность состояний, каждое из которых получено из предыдущего посредством некоторого перехода.

Моделирование систем

- Модель Крипке
  - Граф переходов специального вида
    - Пути в графе соответствуют вычислениям системы
    - Простая и универсальная модель
- Параллельные системы
  - Текст программы или диаграмма
  - Типы систем
    - синхронные, асинхронные, с разделяемыми переменными, обменивающиеся сообщениями и т.п.
  - Описываются формулами логики предикатов первого порядка
    - Формулы корректно транслируются в модель Крипке

1. Моделирование параллельных систем

Пусть AP — множество атомарных высказываний

- Модель Крипке $M = (S, S_0, R, L)$
  - S — конечное множество состояний
  - $S_0 \subseteq S$ — множество начальных состояний
  - $R \subseteq S \times S$ — тотальное отношение переходов: для каждого $s \in S$ существует $s' \in S$, такое что $R(s,s')$.
  - $L : S \to 2^{AP}$ — функция разметки, сопоставляющая каждому состоянию множество атомарных высказываний, истинных в данном состоянии.
- Путь в модели M из состояния s
  - бесконечная последовательность состояний модели $\pi = s_0, s_1, s_2, \ldots$, такая что $s_0 = s$ и для всех $i \geq 0$ верно, что $R(s_i, s_{i+1})$.

1.1. Представления первого порядка

- Логика предикатов первого порядка
  - Предикатные и функциональные символы
    - фиксированная интерпретация
  - Логические связки $\neg$, $\wedge$, $\vee$, $\rightarrow$
  - Кванторы $\forall$, $\exists$
- Описание параллельных систем
  - $V = \{v_0, v_1, v_2, \ldots v_n\}$ — переменные системы
  - D — домен (универсум) интерпретации
  - Означивание для V — функция, сопоставляющая переменным из V их значения из D.
- Состояние s — означивание $s: V \rightarrow D$.
  - Формула, истинная в точности на данном означивании
    - $V = \{v_0, v_1, v_2\}$, $s = \{v_0 \leftarrow 0, v_1 \leftarrow 1, v_2 \leftarrow 2\}$:
      - $(v_0 = 0) \wedge (v_1 = 1) \wedge (v_2 = 2)$
      - ~~$(v_0 = 0) \wedge (v_1 = 1) \vee (v_2 = 2)$~~
  - Формула представляет множество всех означиваний, в которых она истинна.

1.1. Представления первого порядка

- Переходы между состояниями
  - Формулы для представления множества упорядоченных пар состояний
  - Пусть V — переменные системы, V' — копии переменных системы
    - V — переменные текущего состояния
    - V' — переменные следующего состояния
  - Любое означивание V и V' кодирует переход
    - Означивание кодируется формулой первого порядка
    - $s = \{v_0 \leftarrow 0, v_1 \leftarrow 1, v_2 \leftarrow 2\}$,
      $s' = \{v'_0 \leftarrow 1, v'_1 \leftarrow 0, v'_2 \leftarrow 0\}$, $s'' = \{v'_0 \leftarrow 2, v'_1 \leftarrow 0, v'_2 \leftarrow 0\}$:
      - $(v_0 = 0) \wedge (v_1 = 1) \wedge (v_2 = 2) \wedge$
        $(v'_0 = 1) \vee (v'_0 = 2) \wedge \forall\, v \in \{v'_1, v'_2\} : (v = 0)$.
  - *R*(V,V') — формула первого порядка, представляющая отношение переходов *R*
- Атомарные высказывания AP
  - v=d, $v \in V$, $d \in D$: v=d истинно в состоянии s, если s(v)=d.
  - При D={true, false} вместо v=true пишем v, вместо v=false — $\neg v$

1.1. Представления первого порядка

- Построение модели Крипке M = (S, S_0, R, L) по формулам первого порядка S_0 и R(V,V').
  - S — множество всех означиваний переменных V
  - S_0 — множество всех означиваний s_0 переменных V, которые удовлетворяют формуле S_0.
  - R(s,s') верно для всех s, s' таких, что формула R истинна при означиваниях s(v) и s(v') для всех $v \in V$ и $v' \in V'$.
  - $L : S \to 2^{AP}$ — функция разметки такая, что для всех состояний s множество L(s) — все атомарные высказывания, истинные в s. Если v — логическая, то пишем $v \in L(s)$ или $v \notin L(s)$.
- Может оказаться, что какое-то состояние s не имеет последователей.
  - Дополним отношение R так, чтобы было верно R(s,s).

1.1. Представления первого порядка

- Пример системы
  - Пусть V={ x, y}, D={ 0, 1}
  - Означивания — пары $(d_1, d_2) \in D \times D$
  - Переход: x:=(x+y)(mod2), начальное состояние x=1 и y=1.
- Представление первого порядка
  - S_0(x, y) $\equiv$ (x=1)$\wedge$(y=1) — формула начальных состояний.
  - *R*(x, y, x', y') $\equiv$ x'=(x+y)(mod2)$\wedge$(y'=y)
- Модель Крипке M = (S, S_0, R, L)
  - S = D$\times$D
  - S_0={(1,1)}
  - R={((1,1), (0,1)), ((0,1), (1,1)), ((1,0), (1,0)), ((0,0), (0,0))}
  - L((0,0))={x=0,y=0}, L((0,1))={x=0,y=1}, L((1,0))={x=1,y=0}, L((1,1))={x=1,y=1}
- Единственный путь в модели. Вычисление системы
  - (1,1), (0,1), (1,1), (0,1)…

1.2. Степень детализации переходов

- Гранулярность переходов
  - Излишняя детализация
    - Состояние не должно быть результатом выполнения только части действия
  - Излишнее огрубление
    - Состояния не должны пропадать из-за «огрубления» действий, когда последовательность действий представляется как одно действие.

1.2. Степень детализации переходов

- Пример
  - α: `x := x + y;` β: `y := y + x;`
  - `x = 1, y = 2`
  - α_0: `load` R_1`, x` β_0: `load` R_2`, y`
  - α_1: `add` R_1`, y` β_1: `add` R_2`, x`
  - α_2: `store` R_1`, x` β_2: `store` R_2`, y`
  - $\alpha\beta$: `x = 3, y = 5`
  - $\beta\alpha$: `x = 4, y = 3`
  - $\alpha_0\beta_0\alpha_1\beta_1\alpha_2\beta_2$: `x = 3, y = 3`

2. Параллельные системы

- Параллельные системы
  - Набор компонентов, которые исполняются одновременно.
  - Компоненты могут взаимодействовать друг с другом.
  - Типы исполнения и взаимодействия могут быть разными.
    - синхронное
    - асинхронное
    - общие переменные
    - общение

2. Параллельные системы

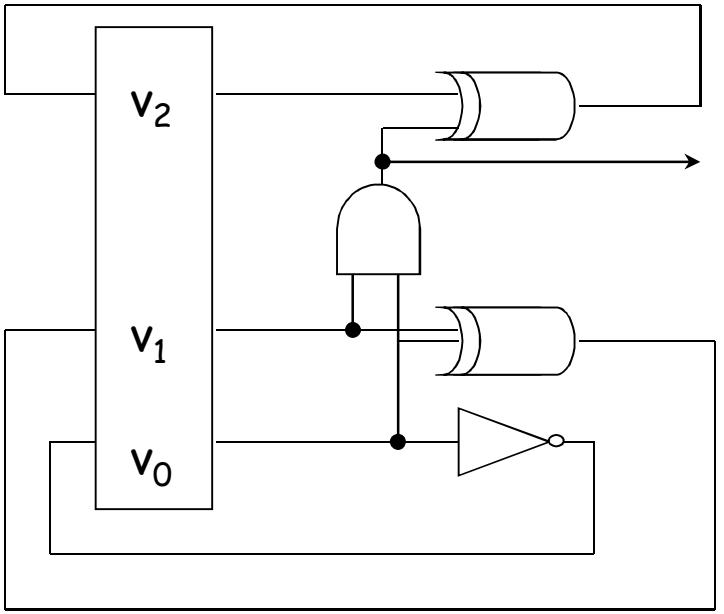
- Параллельные системы
  - Исполнение:
    - асинхронное
      - в любой момент времени только один компонент делает шаг вычисления
    - синхронное
      - все компоненты делают шаг вычисления одновременно
  - Взаимодействие:
    - изменения значения общих переменных
    - обмен сообщениями
      - очереди
      - протоколы синхронной взаимосвязи

2.1. Цифровые схемы

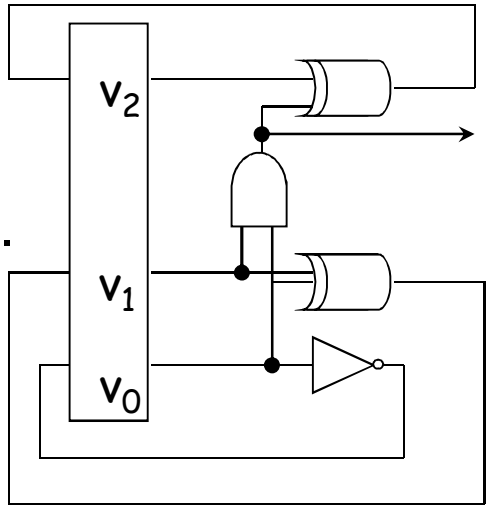
- V — множество элементов, определяющих состояние схемы
  - Элементы принимают значения 0 или 1.
  - Для синхронных схем множество V
    - выходы всех регистров схемы
    - всевозможные первичные входы
  - Для асинхронных схем множество V
    - все соединения
- Каждому элементу из V сопоставляется логическая переменная
  - Состояние определяется означиванием переменных как 0 или 1
  - Для любого означивания есть формула, истинная только на нём
    - $V = \{v_1, v_2\}$ и $(v_1 \leftarrow 1, v_2 \leftarrow 0) : v_1 \wedge \neg v_2$
- Для описания электронных схем не требуется логики предикатов первого порядка.
  - Достаточно булевых формул
  - Пусть булевы формулы
    - S_0 и — множество начальных состояний схемы
    - $R(V,V')$ — отношение переходов схемы

2.1. Цифровые схемы

- Синхронные схемы
  - Работа состоит из последовательности шагов
    1. На каждом шаге изменяются входы схемы, пока она не стабилизируется.
    2. Через схему проходит импульсный синхронизирующий сигнал, и значение элементов изменяется.
- Счетчик по модулю 8.



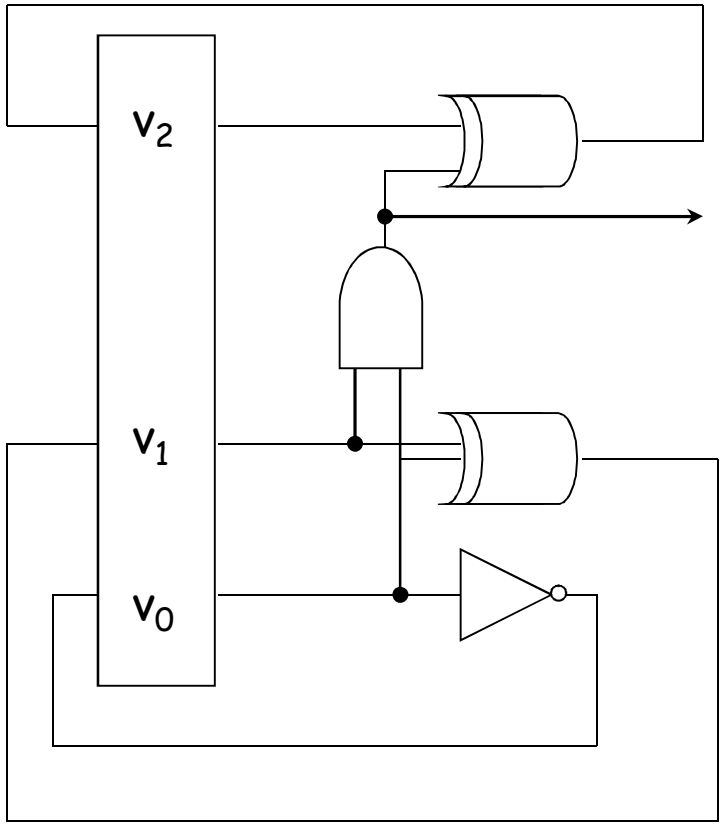
2.1. Цифровые схемы



- Счетчик по модулю 8.
  - V = {v_0,v_1,v_2} —переменные состояния схемы,
  - V` = {$v`_0$,$v`_1$,$v`_2$} — копия переменных состояния.
  - Переходы счетчика ($\oplus$ — исключающее или):
    - $v`_0 = \neg v_0$
    - $v`_1 = v_0 \oplus v_1$
    - $v`_2 = (v_0 \wedge v_1) \oplus v_2$
  - Отношения, описывающих ограничения для переменных $v`_i$:
    - $R_0(V,V`) \equiv (v`_0 \Leftrightarrow \neg v_0)$
    - $R_1(V,V`) \equiv (v`_1 \Leftrightarrow v_0 \oplus v_1)$
    - $R_2(V,V`) \equiv (v`_2 \Leftrightarrow (v_0 \wedge v_1) \oplus v_2)$
  - Все изменения происходят в одно и то же время, поэтому конъюнкция:
    - $R(V,V`) \equiv R_0(V,V`) \wedge R_1(V,V`) \wedge R_2(V,V`)$.

2.1. Цифровые схемы

- В общем случае $V = \{v_0,\ldots,v_n\}$ и $V` = \{v`_0,\ldots,v`_n\}$
- Каждой переменной состояния $v`_i$ сопоставляется булева формула $f_i(V)$, так что $v`_i = f_i(V)$.
- Определяются отношения
  - $R_i(V,V`) \equiv (v`_i \Leftrightarrow f_i(V))$
- Отношение переходов — конъюнкция
  - $R(V,V`) \equiv R_0(V,V`) \wedge \ldots \wedge R_n(V,V`)$.
- Отношение переходов синхронной схемы представляется конъюнкцией отношений переходов для отдельных процессов.



2.1. Цифровые схемы

- Асинхронные схемы
  - Отношение переходов выражается дизъюнкцией.

- Пусть все компоненты схемы имеют ровно один выход и не имеют внутренних переменных состояния.
  - Каждый компонент определяется функцией $f_i(V)$
    - по значениям переменных текущего состояния из V компонент выдает на выходе значение $f_i(V)$.
  - Если у компонента несколько выходов – используем соответствующее количество функций $f^1_i(V)$ … $f^k_i(V)$

2.1. Цифровые схемы

- Считается, что значение компонента изменяется достаточно быстро для того, чтобы два компонента изменили свое значение одновременно.
  - Используется семантика чередования
    - в каждый момент времени только один компонент изменяет свое состояние
  - Дизъюнкция $R(V,V`) \equiv R_0(V,V`) \vee \ldots \vee R_n(V,V`)$
    - где $R_i(V,V`) \equiv (v`_i \Leftrightarrow f_i(V)) \wedge \bigwedge_{j \neq i} (v`_j \Leftrightarrow v`_j)$
- Состояния некоторых компонентов могут изменяться намного чаще, чем других.
  - Во многих случаях это маловероятно.
    - В модель вводятся дополнительные ограничения *справедливости*, которые не будут допускать подобного поведения системы.

2.1. Цифровые схемы

- Разница между синхронной и асинхронной моделями
  - Пусть V = {v_0,v_1}, $v`_0 = v_0 \oplus v_1$ и $v`_1 = v_0 \oplus v_1$
  - Пусть s — состояние, в котором $v_0 = 1 \wedge v_1 = 1$
  - Синхронная модель
    - единственный последователь состояния s
      - состояние $v_0 = 0 \wedge v_1 = 0$
  - Асинхронная модель
    - состояние s имеет двух последователей:
      - $v_0 = 0 \wedge v_1 = 1$ (первым выполняется присваивание v_0)
      - $v_0 = 1 \wedge v_1 = 0$ (первым выполняется присваивание v_1)

2.2. Программы

- Последовательные программы
  - Состоит из последовательности операторов
- Механизм трансляции C преобразует
  - текст последовательной программы P
    - *в формулу первого порядка R*, представляющую множество переходов программы.
- Каждый оператор имеет единственную точку входа и единственную точку выхода.
  - Эти точки помечены уникальным образом.
- Определим преобразование разметки
  - по заданной неразмеченной программе P строит
    - размеченную программу P^L.

2.2. Программы

- Определим преобразование разметки
  - по заданной неразмеченной программе P строит размеченную программу P^L
    - Приписать метку точке входа каждого оператора в P, кроме самой программы P
      - Все метки считаются попарно различными.
    - В последовательной программе точка выхода каждого оператора совпадает с точкой входа следующего за ним оператора
      - Достаточно пометить только точки входа
    - Пометить точки входа и выхода программы P
- Уникальная разметка точек входа и выхода для всех операторов программы.
- Определим преобразование разметки для некоторых распространённых операторов

2.2. Программы

- Для заданного оператора P определим размеченный оператор P^L
  - P не является составным оператором (т. е. `x:=e`, **`skip`**, **`wait`**, **`lock`**, **`unlock`** и т. п.)
    - $P^L = P$
  - $P = P_1 ; P_2$
    - $P^L = P^L_1 ; l_2 : P^L_2$
  - P = **`if`** b **`then`** P_1 **`else`** P_2 **`end if`**
    - P^L = **`if`** b **`then`** $l_1 : P^L_1$ **`else`** $l_2 : P^L_2$ **`end if`**
  - P = **`while`** b **`do`** P_1 **`end while`**
    - P^L = **`while`** b **`do`** $l_1 : P^L_1$ **`end while`**
- Далее считаем, что P — размеченный оператор, с точкой входа m и точкой выхода m`
- *pc* — счетчик команд
  - значения — метки программы и $\perp$
    - *pc* = $\perp$ если компонента параллельной программы неактивна.

2.2. Программы

- V — множество переменных программы и *pc* — счётчик.
- V` — копия множества переменных программы и *pc*` — копия счётчика.
- same(Y) — обозначение формулы $\wedge_{y \in Y}(y`=y)$
- Формула для множество начальных состояний программы P.
  - Задано начальное условия pre(V),
    - $S_0(V,pc) = pre(V) \wedge pc = m$.

2.2. Программы

- Процедура трансляции C зависит от параметров:
  - входная метка I
  - размеченный оператор P
  - выходная метка I`.
  - Определяется рекурсивно.
- Предикат *C(I, P, I`)* представляет множество переходов программы P в виде *дизъюнкции всех переходов* этого множества.
  - Каждый дизъюнкт
    - соответствует отдельному переходу
    - определяет условия перехода:
      - значение булевого условия
      - значение счетчика команд
    - Истинен, когда переход осуществим
    - Ложен, иначе.

2.2. Программы

- Оператор присваивания
  - C(l, **v:=e**, l`) ≡ pc = l ∧ pc` = l` ∧ v` = e ∧ same(V \ {v})
- Оператор **skip**
  - C(l, **skip**, l`) ≡ pc = l ∧ pc` = l` ∧ same(V)
- Последовательная композиция операторов
  - C(l, P_1 **;** l``: $P_2$, l`) ≡ C(l, $P_1$, l``) ∨ C(l``, P_2, l`)
- Оператор ветвления **if-then-else**
  - C(l, **if** b **then** l_1: P_1 **else** l_2: P_2 **end if**, l`) дизъюнкция формул:
    1. pc = l ∧ pc` = l_1 ∧ b ∧ same(V)
    2. pc = l ∧ pc` = l_2 ∧ ¬b ∧ same(V)
    3. C(l_1, P_1, l`)
    4. C(l_2, P_2, l`)
    - Дизъюнкты 1 и 2 описывают переходы, изменяющие показания счетчика команд.
    - Дизъюнкты 3 и 4 — формулы для переходов P_1 и P_2
  - Аналогично для оператора недетерминированного выбора между несколькими альтернативами.

2.2. Программы

- Оператор итерации **`while-do`**
  - C(l, **`while`** b **`do`** l_1: P_1 **`end while`**, l`) дизъюнкция формул:
    1. pc = l $\wedge$ pc` = l_1 $\wedge$ b $\wedge$ same(V)
    2. pc = l $\wedge$ pc` = l` $\wedge$ $\neg$b $\wedge$ same(V)
    3. C(l_1, P_1, l)
       - Дизъюнкт 1: условие b истинно и следующий оператор – P_1.
       - Дизъюнкт 2: условие b ложно, цикл завершается.
       - Дизъюнкт 3: формула для переходов оператора P_1.
  - Точка выхода оператора P_1 совпадает с точкой входа в оператор итерации.

2.3. Параллельные программы

- Параллельная программа
  - Множество процессов исполняются параллельно
    - Процесс — последовательность операторов
    - Взаимодействующие процессы
- Асинхронные программы
  - Ровно один процесс может совершить переход в каждый момент времени.
- V_i – множество переменных, изменяемых процессом P_i
  - Эти множества могут пересекаться
- pc_i – счетчик команд процесса P_i
- PC – множество всех счетчиков
- Параллельная программа P:
  - **cobegin** $P_1 \parallel P_2 \parallel ... \parallel P_n$ **coend**
    - P_1,..., P_n — процессы

2.3. Параллельные программы

- Параллельная программа P:
  - **cobegin** $P_1 \| P_2 \| ... \| P_n$ **coend**
    - $P_1,..., P_n$ — процессы
- Новое правило преобразования разметки
  - Параллельная программа имеет статус оператора в составе последовательной программы
  - Метка в точке входа и точке выхода каждого оператора
  - Никакая точка выхода параллельного процесса
    - *не отождествляется* с какой-либо точкой входа
  - Точки выхода процессов помечаются отдельно.
    - все метки попарно различны
    - точки входа и выхода программы P: m и m`
- P = **cobegin** $P_1 \| P_2 \| ... \| P_n$ **coend**
  - P^L = **cobegin** $l_1: P^L_1\ l`_1 \| l_2: P^L_2\ l`_2 \| ... \| l_n: P^L_n\ l`_n$ **coend**

2.3. Параллельные программы

- Начальные состояния параллельной программы P
  - $S_0(V,PC) = pre(V) \wedge pc = m \wedge \wedge^n_{i=1} (pc = \perp)$
    - $pc_i = \perp$
      - процесс P_i еще не был активизирован и не исполняется
- Процедура трансляции C для параллельных программ
  - C(I, **cobegin** $P_1 \parallel P_2 \parallel ... \parallel P_n$ **coend**, I`) дизъюнкция формул:
    - $pc = I \wedge pc`_1 = I_1 \wedge \ldots \wedge pc`_n = I_n \wedge pc` = \perp$
    - $pc = \perp \wedge pc_1 = I`_1 \wedge \ldots \wedge pc_n = I`_n \wedge pc` = I` \wedge \wedge^n_{i=1} (pc`_i = \perp)$
    - $\vee^n_{i=1} (C(I_i, P_i, I`_i) \wedge same(V \backslash V_i) \wedge same\{PC \backslash \{pc_i\}))$.

2.3. Параллельные программы

- C(I, **cobegin** $P_1 \parallel P_2 \parallel \ldots \parallel P_n$ **coend**, I`) дизъюнкция формул:
  1. $pc = I \wedge pc`_1 = I_1 \wedge \ldots \wedge pc`_n = I_n \wedge pc` = \bot$
     - Инициализация параллельных процессов
       - Переход из точки входа **cobegin** в точки входа процессов
  2. $pc = \bot \wedge pc_1 = I`_1 \wedge \ldots \wedge pc_n = I`_n \wedge pc` = I` \wedge \wedge^n_{i=1} (pc`_i = \bot)$
     - Завершение параллельной программы
       - Переход из точек выхода процессов в точку выхода оператора **cobegin**
         - Исполняется при завершении всех процессов
  3. $\vee^n_{i=1} (C(I_i, P_i, I`_i) \wedge same(V \backslash V_i) \wedge same(PC \backslash \{pc_i\}))$
     - Чередующееся исполнение параллельных процессов
       - Конъюнкция:
         - формула для отношения переходов процесса P_i
         - формула неизменности переменных и счётчиков других процессов, в каждый момент времени только один процесс может совершить переход.

2.3. Параллельные программы

Разделяемые переменные

- Программы с разделяемыми переменными
  - параллельные программы, для которых множества переменных V_i, изменяемых процессами, пересекаются
- Процедура трансляции С для общепринятых операторов синхронизации процессов
  - Операторы синхронизации
    - средства исключительного доступа процессов к разделяемым переменным
    - атомарные операторы, обрабатываются преобразованием разметки обычным образом.

2.3. Параллельные программы

- Оператор **`wait(b)`**
  - Реализация оператора посредством ожидания по условию
    - программы с конечным числом состояний
    - не рассматриваем реализации, требующие сложных структур данных (очереди процессов)
  - **`wait(b)`** периодически проверяет значение булевой переменной **`b`**, пока не узнает, что **`b`** стала истиной, после чего осуществляется переход к следующему оператору программы
  - C(I, **`wait(b)`**, I`) дизъюнкция формул
    - $pc_i = I \wedge pc`_i = I \wedge \neg b \wedge same(V_i)$
    - $pc_i = I \wedge pc`_i = I` \wedge b \wedge same(V_i)$

2.3. Параллельные программы

- Оператор **`lock(v)`**.
  - Действует как **`wait(v = 0)`** и при выходе значение **`v`** изменяется на 1.
  - Применяется, чтобы обеспечить взаимное исключение
    - В критической секции не больше одного процесса
  - C(I, **`lock(v)`**, I`) дизъюнкция формул:
    - $pc_i = I \wedge pc`_i = I \wedge v = 1 \wedge same(V_i)$
    - $pc_i = I \wedge pc`_i = I` \wedge v = 0 \wedge v` = 1 \wedge same(V_i \setminus \{v\})$
- Оператор **`unlock(v)`**
  - Присваивает значение 0 переменной v
    - Дает разрешение процессу на вход в критическую секцию
  - C(I, **`unlock(v)`**, I`) $\equiv pc_i = I \wedge pc`_i = I` \wedge v` = 0 \wedge same(V_i \setminus \{v\})$

3. Пример трансляции программы

- Программа со взаимным исключением
  - P = m: **cobegin** P_0 || P_1 **coend** m`
- pc — счетчик команд P, его значения:
  - m — входная метка P
  - m` — выходная метка P
  - ⊥ — значение pc, когда P_0 и P_1 активны
- pc_i — счетчиком команд процессов P_i
  - его значения: l_i, $l`_i$, NC_i, CR_i и ⊥
- Разделяемая переменная **turn**.
- $V = V_0 = V_1 = \{$**turn**$\}$ и $PC = \{pc, pc_0, pc_1\}$.
- $pc_i = CR_i$: процесс i находится в критической секции.
  - Запрещается находиться в критической секции одновременно.
- $pc_i = NC_i$: процесс i пребывает в некритической секции.
  - Процесс дожидается выполнения **turn** = i для входа в критическую секцию.

P_0:: l_0: **while *True* do**
  NC_0: **wait (turn = 0);**
  CR_0: **turn := 1;**
end while
$l`_0$

P_1:: l_1: **while *True* do**
  NC_1: **wait (turn = 1);**
  CR_1: **turn := 0;**
end while
$l`_1$

3. Пример трансляции программы

- Начальные состояния параллельной программы P:
  - $S_0(V,PC) \equiv pc = m \wedge pc_0 = \perp \wedge pc_1 = \perp$
    - Ограничений на значение turn не налагается
- В результате трансляции C формула для отношения переходов R(V, PC, V`, PC`) программы P: дизъюнкция формул:
  - $pc = m \wedge pc`_0 = l_0 \wedge pc`_1 = l_1 \wedge pc` = \perp$
  - $pc_0 = l`_0 \wedge pc_1 = l`_1 \wedge pc` = m` \wedge pc`_0 = \perp \wedge pc`_1 = \perp$
  - $C(l_0, P_0, l`_0) \wedge same(V \setminus V_0) \wedge same(PC \setminus \{pc_0\})$
    - $C(l_0, P_0, l`_0) \wedge same\{pc, pc_1\})$
  - $C(l_1, P_1, l`_1) \wedge same(V \setminus V_1) \wedge same(PC \setminus \{pc_1\})$
    - $C(l_1, P_1, l`_1) \wedge same\{pc, pc_0\})$

P_0:: l_0: **while *True* do**
NC_0: **wait (turn = 0);**
CR_0: **turn := 1;**
end while
$l`_0$

P_1:: l_1: **while *True* do**
NC_1: **wait (turn = 1);**
CR_1: **turn := 0;**
end while
$l`_1$

3. Пример трансляции программы

- Для каждого P_i формула $C(l_i, P_i, l`_i)$ — дизъюнкция формул:
  - $pc_i = l_i \wedge pc`_i = NC_i \wedge True \wedge same(\mathbf{turn})$
  - $pc_i = NC_i \wedge pc`_i = CR_i \wedge \mathbf{turn} = i \wedge same(\mathbf{turn})$
  - $pc_i = CR_i \wedge pc`_i = l_i \wedge \mathbf{turn} = (i+1) (mod\ 2)$
  - $pc_i = NC_i \wedge pc`_i = NC_i \wedge \mathbf{turn} \neq i \wedge same(\mathbf{turn})$
  - $pc_i = l_i \wedge pc`_i = l`_i \wedge False \wedge same(\mathbf{turn})$

P_0:: l_0: **while** ***True*** **do**
NC_0: **wait (turn = 0);**
CR_0: **turn := 1;**
end while
$l`_0$

P_1:: l_1: **while** ***True*** **do**
NC_1: **wait (turn = 1);**
CR_1: **turn := 0;**
end while
$l`_1$

3. Пример трансляции программы

- Модель Крипке строится по формулам S_0 и R
  - Процессы никогда не окажутся в своих критических секциях одновременно
    - Свойство взаимного исключения
  - Не обеспечивается отсутствие удушения (starvation)
    - один процесс может непрерывно пытаться войти в свою критическую секцию, никогда не получая при этом доступа туда, и при этом другой процесс находится в своей критической секции бесконечно долго.

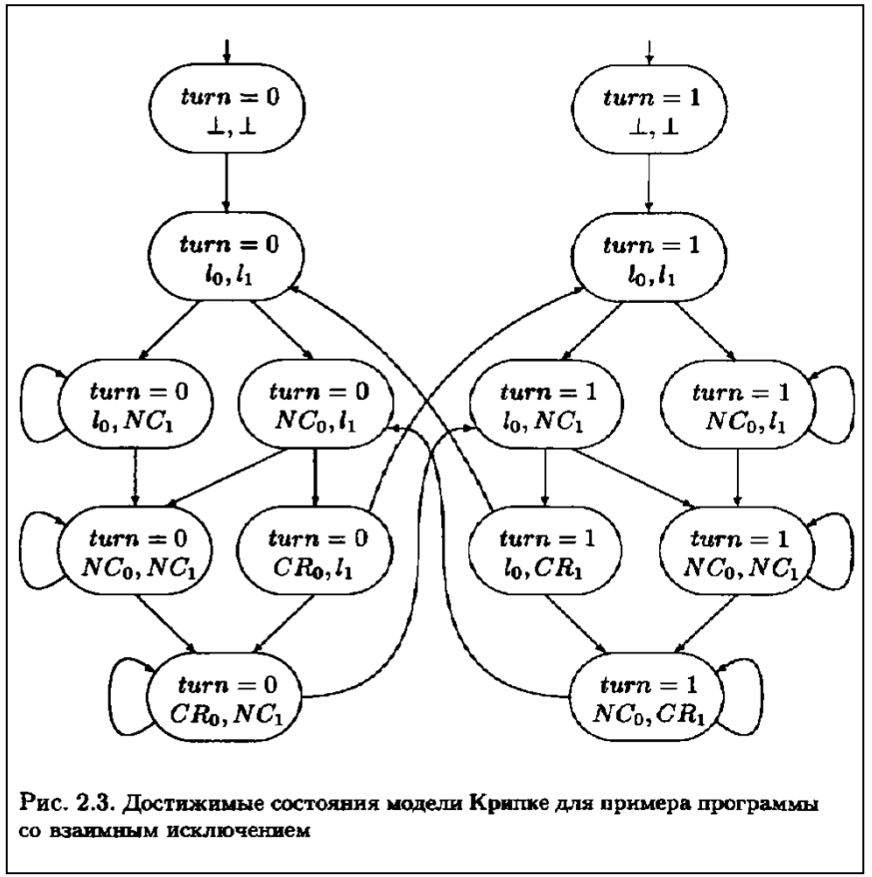


Рис. 2.3. Достижимые состояния модели Крипке для примера программы со взаимным исключением

3. Пример трансляции программы

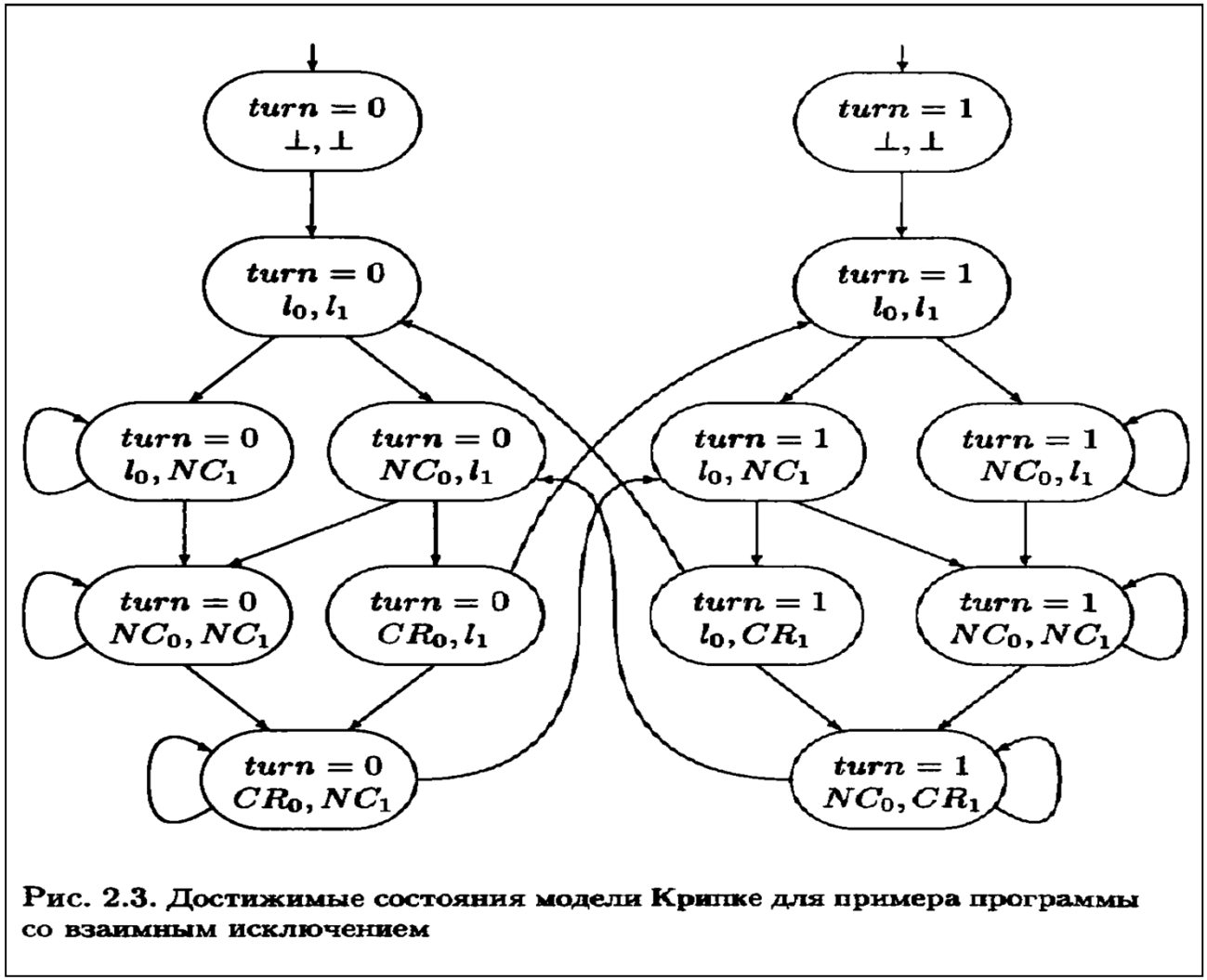


Рис. 2.3. Достижимые состояния модели Крипке для примера программы со взаимным исключением

Логики CTL, LTL

Темпоральные логики

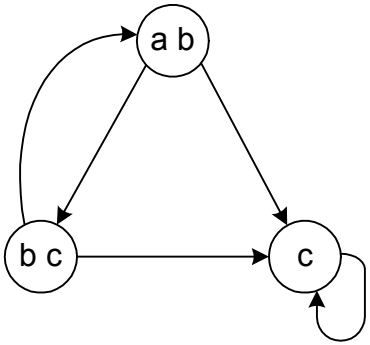
- Логики для спецификации временны̀х свойств систем переходов (моделей Крипке)
- Используются атомарные высказывания, булевы связки, кванторы и модальности, операторы, описывающие временные свойства состояний системы.
- Реагирующие системы
  - Спецификация последовательности переходов между состояниями
    - Метод Флойда-Хоара
      - Спецификация свойств входных и выходных данных
    - Темпоральные логики
      - Спецификация процесса работы (вычисления)

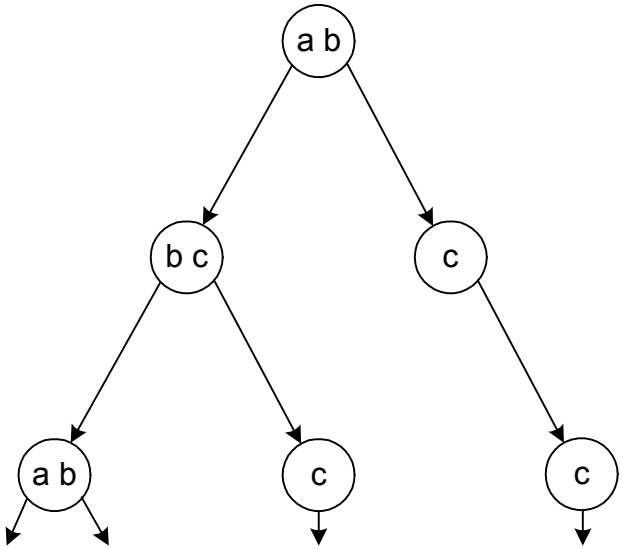
Темпоральные логики

- Время не описывается явно
  - когда-нибудь
  - никогда
- Темпоральные операторы для описания временных свойств.
- Темпоральные логики отличаются
  - набором используемых темпоральных операторов
  - семантикой этих операторов
- Выразительная логика CTL* — Computational Tree Logic Star
  - CTL — Computational Tree Logic
  - LTL — Linear Tree Logic

1 Логика деревьев вычислений CTL*

- Формулы CTL* описывают свойства деревьев вычислений.
- Дерево вычислений
  - Корень — начальное состояние
  - Потомки — состояния, получающиеся путём переходов
  - Бесконечное
  - Все возможные исполнения





1 Логика деревьев вычислений CTL*

- Формулы CTL* описывают свойства деревьев вычислений.
- Дерево вычислений
  - Корень — начальное состояние
  - Потомки — состояния, получающиеся путём переходов
  - Бесконечное
  - Все возможные исполнения
- Формулы CTL*
  - Кванторы пути
    - Описание структуры ветвления в дереве вычислений
      - **A** («для всех путей вычисления»)
      - **E** («для некоторого пути вычисления»)
      - Все пути или некоторые из путей из данного состояния, обладают некоторым свойством
  - Темпоральные операторы
    - Свойства пути, проходящего в дереве.

1 Логика деревьев вычислений CTL*

- Оператор сдвига по времени **X** (neXttime, «в следующий момент»)
  - Свойство выполняется в следующем состоянии пути
- Оператор происшествия **F** (Future, «рано или поздно», «когда-то в будущем», «когда-нибудь»)
  - Свойство будет выполняться в каком-то последующем состоянии пути
- Оператор инвариантности **G** (Globally, «всегда», «повсюду»)
  - Свойство выполняется в каждом состоянии пути

1 Логика деревьев вычислений CTL*

- Оператор условного ожидания **U** (Until, «до тех пор пока», «пока не»)
  - Свойство выполнено начиная с некоторого момента до тех пор, пока не начнёт выполняться некоторое другое свойство
- Оператор разблокировки **R** (Release, «высвободить»)
  - Второе свойство выполняется до тех пор, пока не начнёт выполняться первое.

1 Логика деревьев вычислений CTL*

Синтаксис и семантика CTL*

AP – имена атомарных высказываний

- Формулы двух типов:
  - формулы состояния (высказывание о состоянии)
    - Если $p \in AP$, то p — формула состояния
    - Если φ и ψ — формулы состояния, то
      $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$ — формулы состояния
    - Если φ — формула пути, то
      $\mathbf{E}\varphi$ и $\mathbf{A}\varphi$ — формулы состояния
  - формулы пути (высказывание о пути)
    - Если φ — формула состояния, то φ — формула пути
    - Если φ и ψ — формулы состояния, то
      $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\mathbf{X}\varphi$, $\mathbf{F}\varphi$, $\mathbf{G}\varphi$, $\varphi\mathbf{U}\psi$, $\varphi\mathbf{R}\psi$ — формулы пути
- Формулами CTL* являются все формулы состояний, построенные по этим правилам

1 Логика деревьев вычислений CTL*

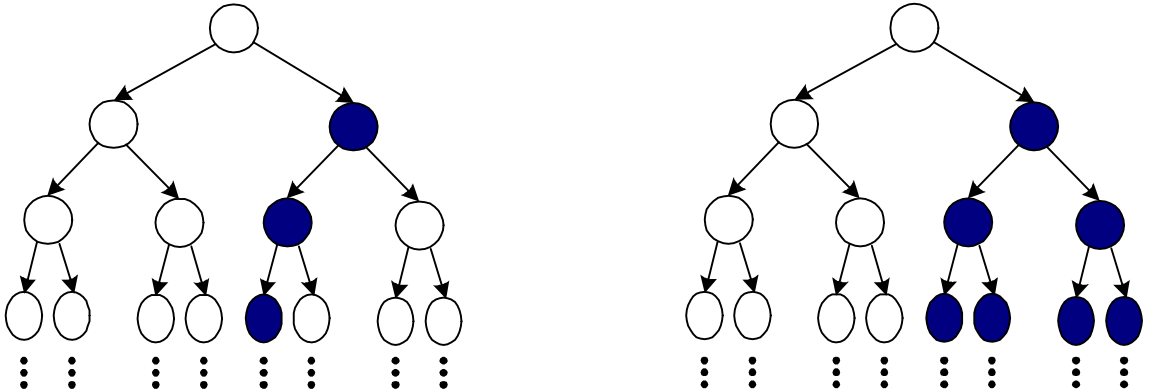
Семантика CTL*

- Определяется на моделях Крипке
  - M = (S, R, L)
    - S — множество состояний
    - R×R — тотальное отношение переходов
    - L: S → 2^{AP} — функция разметки
- Путь в модели M из состояния s
  - бесконечная последовательность состояний модели

    $\pi = s_0, s_1, s_2, \ldots$, такая что s_0=s и для всех i≥0 верно, что $R(s_i, s_{i+1})$.
    - Бесконечная ветвь в дереве вычислений модели M
- π^i – суффикс пути π, начинающегося в состоянии s_i.
- M, s ⊨ φ – φ выполняется в состоянии s на модели Крипке M
  - φ — формула состояния
- M, π ⊨ φ – φ выполняется на протяжении пути π в модели M
  - φ — формула пути.
- Обозначение модели Крипке может быть опущено.

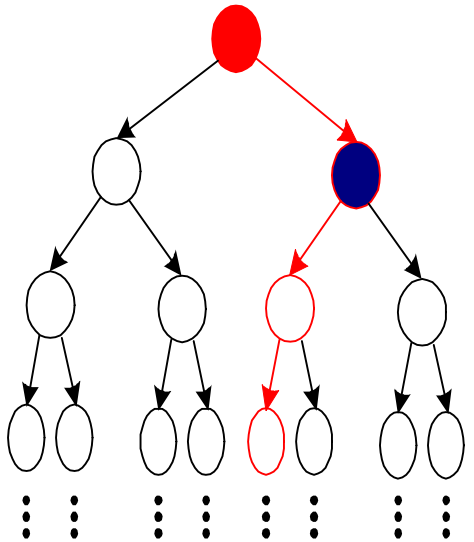
1 Логика деревьев вычислений CTL*

- Отношение $\models$ определяется индуктивно следующим образом
  - φ, φ_1 и φ_2 — формулы состояния
  - ψ, ψ_1 и ψ_2 — формулы пути:
- $M, s \models p \Leftrightarrow p \in L(s)$
- $M, s \models \neg\varphi \Leftrightarrow M, s \not\models \varphi$
- $M, s \models \varphi_1 \wedge \varphi_2 \Leftrightarrow M, s \models \varphi_1$ и $M, s \models \varphi_2$
- $M, s \models \varphi_1 \vee \varphi_2 \Leftrightarrow M, s \models \varphi_1$ или $M, s \models \varphi_2$
- $M, s \models \mathbf{E}\ \psi \Leftrightarrow$ в M существует такой путь π из состояния s, что $M, \pi \models \psi$
- $M, s \models \mathbf{A}\ \psi \Leftrightarrow$ в M для любого пути π из состояния s верно $M, \pi \models \psi$



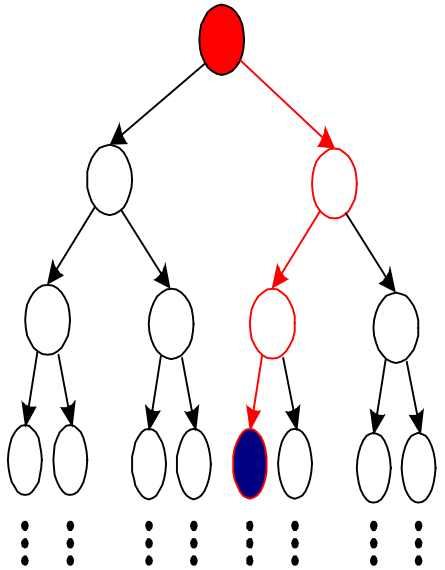
1 Логика деревьев вычислений CTL*

- Отношение $\models$ определяется индуктивно следующим образом
  - φ, φ_1 и φ_2 — формулы состояния
  - ψ, ψ_1 и ψ_2 — формулы пути:
- M, $\pi \models \psi \Leftrightarrow$ для первого состояния s пути π в M верно M, s $\models \psi$
- M, $\pi \models \neg \psi \Leftrightarrow$ M, $\pi \not\models \psi$
- M, $\pi \models \psi_1 \wedge \psi_2 \Leftrightarrow$ M, $\pi \models \psi_1$ и M, $\pi \models \psi_2$
- M, $\pi \models \psi_1 \vee \psi_2 \Leftrightarrow$ M, $\pi \models \psi_1$ или M, $\pi \models \psi_2$
- M, $\pi \models \mathbf{X}\psi \Leftrightarrow$ M, $\pi^2 \models \psi$
- M, $\pi \models \mathbf{F}\psi \Leftrightarrow$ существует такое $k \geq 0$, что M, $\pi^k \models \psi$
- M, $\pi \models \mathbf{G}\psi \Leftrightarrow$ для любого $k \geq 0$ верно, что M, $\pi^k \models \psi$
- M, $\pi \models \psi_1 \mathbf{U} \psi_2 \Leftrightarrow$ существует такое $k \geq 0$, что M, $\pi^k \models \psi_2 \Leftrightarrow$ и для всех $0 \leq j < k$ верно M, $\pi^j \models \psi_1$
- M, $\pi \models \psi_1 \mathbf{R} \psi_2 \Leftrightarrow$ для всех $k \geq 0$, если для каждого $j < k$ верно M, $\pi^j \not\models \psi_1$, то M, $\pi^k \models \psi_2$



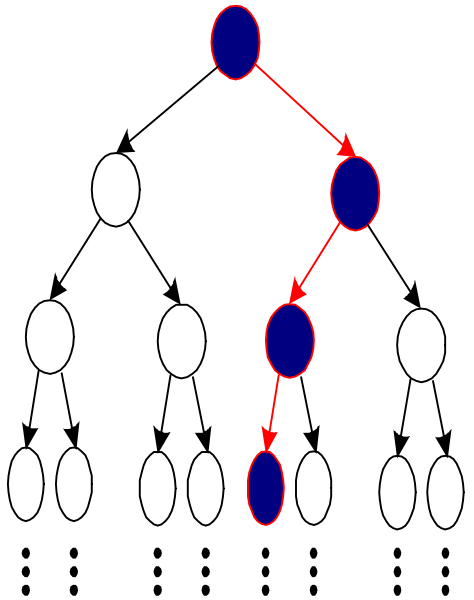
1 Логика деревьев вычислений CTL*

- Отношение $\models$ определяется индуктивно следующим образом
  - φ, φ_1 и φ_2 – формулы состояния
  - ψ, ψ_1 и ψ_2 – формулы пути:
- M, $\pi \models \psi \Leftrightarrow$ для первого состояния s пути π в M верно M, s $\models \psi$
- M, $\pi \models \neg \psi \Leftrightarrow$ M, $\pi \not\models \psi$
- M, $\pi \models \psi_1 \wedge \psi_2 \Leftrightarrow$ M, $\pi \models \psi_1$ и M, $\pi \models \psi_2$
- M, $\pi \models \psi_1 \vee \psi_2 \Leftrightarrow$ M, $\pi \models \psi_1$ или M, $\pi \models \psi_2$
- M, $\pi \models \mathbf{X}\psi \Leftrightarrow$ M, $\pi^2 \models \psi$
- M, $\pi \models \mathbf{F}\psi \Leftrightarrow$ существует такое $k \geq 0$, что M, $\pi^k \models \psi$
- M, $\pi \models \mathbf{G}\psi \Leftrightarrow$ для любого $k \geq 0$ верно, что M, $\pi^k \models \psi$
- M, $\pi \models \psi_1 \mathbf{U} \psi_2 \Leftrightarrow$ существует такое $k \geq 0$, что M, $\pi^k \models \psi_2 \Leftrightarrow$ и для всех $0 \leq j < k$ верно M, $\pi^j \models \psi_1$
- M, $\pi \models \psi_1 \mathbf{R} \psi_2 \Leftrightarrow$ для всех $k \geq 0$, если для каждого $j < k$ верно M, $\pi^j \not\models \psi_1$, то M, $\pi^k \models \psi_2$



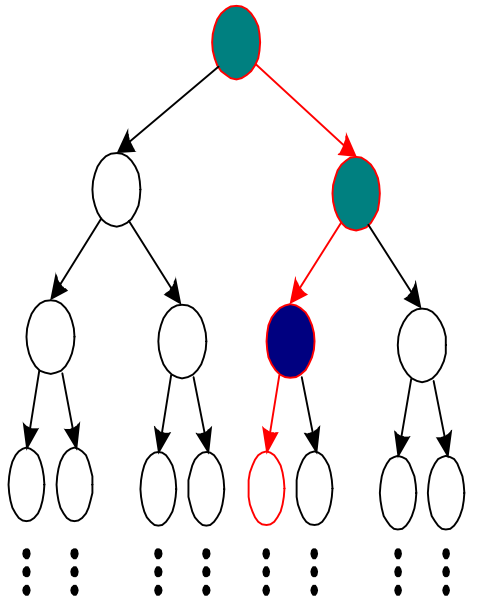
1 Логика деревьев вычислений CTL*

- Отношение $\models$ определяется индуктивно следующим образом
  - φ, φ_1 и φ_2 — формулы состояния
  - ψ, ψ_1 и ψ_2 — формулы пути:
- $M, \pi \models \psi \Leftrightarrow$ для первого состояния s пути π в M верно $M, s \models \psi$
- $M, \pi \models \neg \psi \Leftrightarrow M, \pi \not\models \psi$
- $M, \pi \models \psi_1 \wedge \psi_2 \Leftrightarrow M, \pi \models \psi_1$ и $M, \pi \models \psi_2$
- $M, \pi \models \psi_1 \vee \psi_2 \Leftrightarrow M, \pi \models \psi_1$ или $M, \pi \models \psi_2$
- $M, \pi \models \mathbf{X}\psi \Leftrightarrow M, \pi^2 \models \psi$
- $M, \pi \models \mathbf{F}\psi \Leftrightarrow$ существует такое $k \geq 0$, что $M, \pi^k \models \psi$
- $M, \pi \models \mathbf{G}\psi \Leftrightarrow$ для любого $k \geq 0$ верно, что $M, \pi^k \models \psi$
- $M, \pi \models \psi_1 \mathbf{U} \psi_2 \Leftrightarrow$ существует такое $k \geq 0$, что $M, \pi^k \models \psi_2 \Leftrightarrow$ и для всех $0 \leq j < k$ верно $M, \pi^j \models \psi_1$
- $M, \pi \models \psi_1 \mathbf{R} \psi_2 \Leftrightarrow$ для всех $k \geq 0$, если для каждого $j < k$ верно $M, \pi^j \not\models \psi_1$, то $M, \pi^k \models \psi_2$



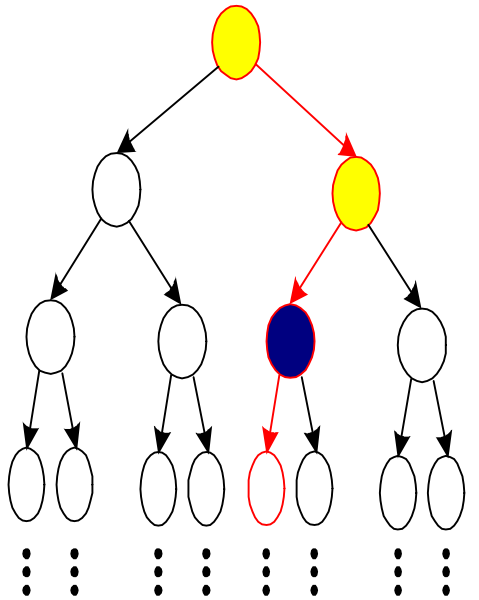
1 Логика деревьев вычислений CTL*

- Отношение $\models$ определяется индуктивно следующим образом
  - φ, φ_1 и φ_2 — формулы состояния
  - ψ, ψ_1 и ψ_2 — формулы пути:
- M, $\pi \models \psi \Leftrightarrow$ для первого состояния s пути π в M верно M, s $\models \psi$
- M, $\pi \models \neg \psi \Leftrightarrow$ M, $\pi \not\models \psi$
- M, $\pi \models \psi_1 \wedge \psi_2 \Leftrightarrow$ M, $\pi \models \psi_1$ и M, $\pi \models \psi_2$
- M, $\pi \models \psi_1 \vee \psi_2 \Leftrightarrow$ M, $\pi \models \psi_1$ или M, $\pi \models \psi_2$
- M, $\pi \models \mathbf{X}\psi \Leftrightarrow$ M, $\pi^2 \models \psi$
- M, $\pi \models \mathbf{F}\psi \Leftrightarrow$ существует такое $k \geq 0$, что M, $\pi^k \models \psi$
- M, $\pi \models \mathbf{G}\psi \Leftrightarrow$ для любого $k \geq 0$ верно, что M, $\pi^k \models \psi$
- M, $\pi \models \psi_1 \mathbf{U} \psi_2 \Leftrightarrow$ существует такое $k \geq 0$, что M, $\pi^k \models \psi_2 \Leftrightarrow$ и для всех $0 \leq j < k$ верно M, $\pi^j \models \psi_1$
- M, $\pi \models \psi_1 \mathbf{R} \psi_2 \Leftrightarrow$ для всех $k \geq 0$, если для каждого $j < k$ верно M, $\pi^j \not\models \psi_1$, то M, $\pi^k \models \psi_2$



1 Логика деревьев вычислений CTL*

- Отношение $\models$ определяется индуктивно следующим образом
  - φ, φ_1 и φ_2 — формулы состояния
  - ψ, ψ_1 и ψ_2 — формулы пути:
- M, $\pi \models \psi \Leftrightarrow$ для первого состояния s пути π в M верно M, s $\models \psi$
- M, $\pi \models \neg \psi \Leftrightarrow$ M, $\pi \not\models \psi$
- M, $\pi \models \psi_1 \wedge \psi_2 \Leftrightarrow$ M, $\pi \models \psi_1$ и M, $\pi \models \psi_2$
- M, $\pi \models \psi_1 \vee \psi_2 \Leftrightarrow$ M, $\pi \models \psi_1$ или M, $\pi \models \psi_2$
- M, $\pi \models \mathbf{X}\psi \Leftrightarrow$ M, $\pi^2 \models \psi$
- M, $\pi \models \mathbf{F}\psi \Leftrightarrow$ существует такое $k \geq 0$, что M, $\pi^k \models \psi$
- M, $\pi \models \mathbf{G}\psi \Leftrightarrow$ для любого $k \geq 0$ верно, что M, $\pi^k \models \psi$
- M, $\pi \models \psi_1 \mathbf{U} \psi_2 \Leftrightarrow$ существует такое $k \geq 0$, что M, $\pi^k \models \psi_2 \Leftrightarrow$ и для всех $0 \leq j < k$ верно M, $\pi^j \models \psi_1$
- M, $\pi \models \psi_1 \mathbf{R} \psi_2 \Leftrightarrow$ для всех $k \geq 0$, если для каждого $j < k$ верно M, $\pi^j \not\models \psi_1$, то M, $\pi^k \models \psi_2$



1 Логика деревьев вычислений CTL*

- Операторов $\neg$, $\vee$, **X**, **U**, **E** достаточно, чтобы выразить любую формулу CTL*:
  - $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$
  - $\varphi \mathbf{R} \psi \equiv \neg(\neg\varphi \ \mathbf{U} \ \neg\psi)$
  - $\mathbf{F}\varphi \equiv \text{True} \ \mathbf{U} \ \neg\varphi$
  - $\mathbf{G}\varphi \equiv \neg\mathbf{F} \ \neg\varphi$
  - $\mathbf{A}\varphi \equiv \neg\mathbf{E} \ \neg\varphi$

2. CTL и LTL

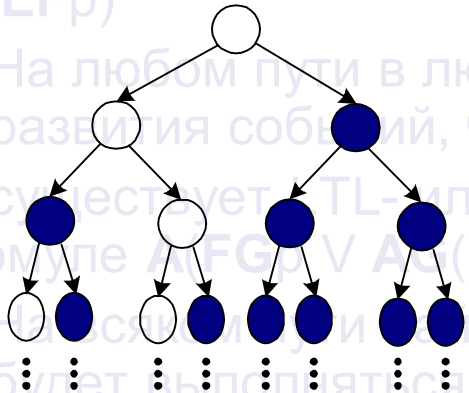
- Подмножества CTL*
  - CTL — логика ветвящегося времени
  - LTL — логика линейного времени.
- Отношение к ветвлению в дереве вычислений.
  - Ветвящееся время
    - каждый темпоральный оператор описывает пути вычисления, заданные ему своим квантором
  - Линейное время
    - темпоральные операторы описывает события на протяжении единственного пути вычисления

2. CTL и LTL

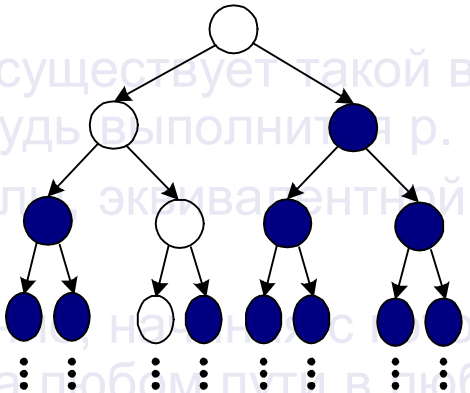
- Логика деревьев вычислений CTL
  - фрагмент CTL*
    - каждый темпоральный оператор **X**, **F**, **G**, **U** и **R** следует непосредственно за квантором пути
  - Если φ, φ_1 и φ_2 — формулы пути, то **AX**φ, **AF**φ, **AG**φ, φ_1 **AU**φ_2 и φ_1 **AR**φ_2 — формулы пути, для **E** аналогично.
- Линейная темпоральная логика LTL
  - фрагмент CTL*
    - состоит из всех формул вида **A**ψ, где ψ — формула пути, в которой все формулы состояния — это атомарные высказывания
  - Если $p \in AP$, то p — формула пути
  - Если ψ, ψ_1 и ψ_2 — формулы пути, то $\neg\psi$, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, **X**ψ, **F**ψ, **G**ψ, ψ_1**U**ψ_2 и ψ_1**R**ψ_2 — формулы пути

2. CTL и LTL

- Разные выразительные возможности.
  - Не существует CTL-формулы, эквивалентной LTL-формуле **A**(**FG**p).
    - На всяком пути найдется состояние, начиная с которого p будет выполняться всегда
  - Не существует LTL-формулы, эквивалентной CTL-формуле **AG**(**EF**p)
    - На любом пути в любой момент существует такой вариант развития событий, что когда-нибудь выполнится p.
  - Не существует LTL- или CTL-формулы, эквивалентной CTL*-формуле **A**(**FG**p)∨**A**(**G**(**EF**p))
    - На всяком пути найдется состояние, начиная с которого p будет выполняться всегда или на любом пути в любой момент существует такой вариант развития событий, что когда-нибудь выполнится p.



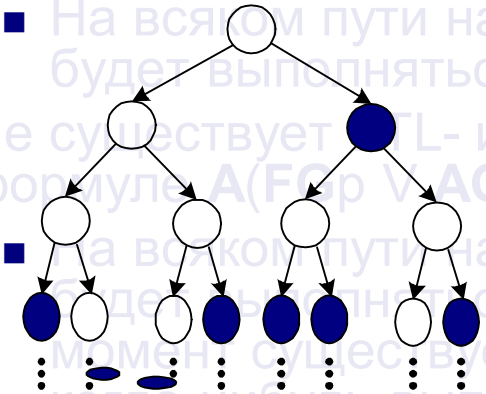
A(FGp**)**



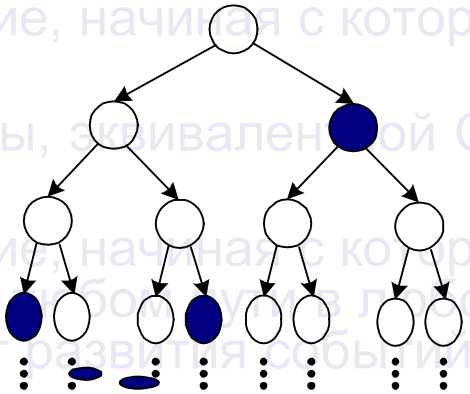
AF(AGp**)**

2. CTL и LTL

- Разные выразительные возможности.
  - Не существует LTL-формулы, эквивалентной CTL-формуле **AG**(**EF**p)
    - На любом пути в любой момент существует такой вариант развития событий, что когда-нибудь выполнится p.
  - Не существует CTL-формулы, эквивалентной LTL-формуле **A**(**FG**p).
    - На всяком пути найдется состояние, начиная с которого p будет выполняться всегда
  - Не существует LTL- или CTL-формулы, эквивалентной CTL*-формуле **A**(**FG**p ∨ **AG**(EFp))
    - На всяком пути найдется состояние, начиная с которого p будет выполняться всегда или на любом пути в любой момент существует такой вариант развития событий, что когда-нибудь выполнится p.



AG(**EF**p)



AG¬*G*¬p

EFp=¬*A*¬Fp= ¬*AG*¬p

2. CTL и LTL

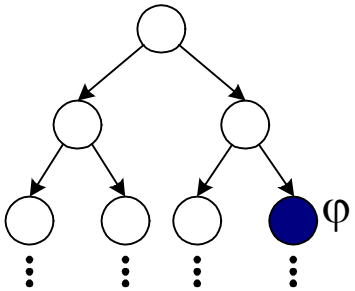
- Разные выразительные возможности.
  - Не существует CTL-формулы, эквивалентной LTL-формуле **A**(**FG**p).
    - На всяком пути найдется состояние, начиная с которого p будет выполняться всегда
  - Не существует LTL-формулы, эквивалентной CTL-формуле **AG**(**EF**p)
    - На любом пути в любой момент существует такой вариант развития событий, что когда-нибудь выполнится p.
  - Не существует LTL- или CTL-формулы, эквивалентной CTL*-формуле **A**(**FG**p V **AG**(EFp))
    - На всяком пути найдется состояние, начиная с которого p будет выполняться всегда или на любом пути в любой момент существует такой вариант развития событий, что когда-нибудь выполнится p.

2. CTL и LTL

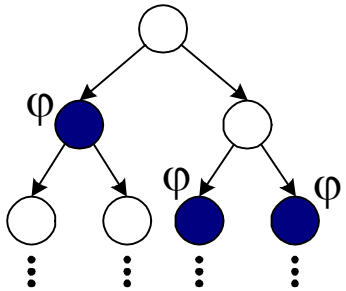
- Основные операторы CTL:
  - **AX** и **EX**
  - **AF** и **EF**
  - **AG** и **EG**
  - **AU** и **EU**
  - **AR** и **ER**
- Достаточно трех операторов EX, EG и EU:
  - $\mathbf{AX}\varphi \equiv \neg\mathbf{EX}\neg\varphi$
  - $\mathbf{EF}\varphi \equiv \mathbf{E}[\ \mathrm{True}\ \mathbf{U}\varphi]$
  - $\mathbf{AG}\varphi \equiv \neg\mathbf{EF}\neg\varphi$
  - $\mathbf{AF}\varphi \equiv \neg\mathbf{EG}\neg\varphi$
  - $\mathbf{A}[\varphi\mathbf{U}\psi] \equiv \neg\mathbf{E}[\neg\psi\mathbf{U}(\neg\varphi\wedge\neg\psi)] \wedge \neg\mathbf{EG}\neg\psi$
  - $\mathbf{A}[\varphi\mathbf{R}\psi] \equiv \neg\mathbf{E}[\neg\varphi\mathbf{R}\neg\psi]$
  - $\mathbf{E}[\varphi\mathbf{R}\psi] \equiv \neg\mathbf{A}[\neg\varphi\mathbf{R}\neg\psi]$

2. CTL и LTL

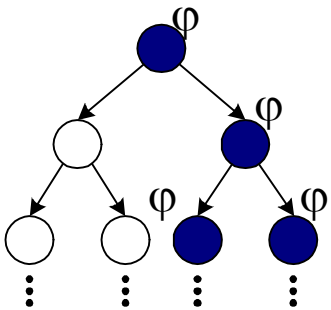
- Семантика четырех часто встречающихся операторов.
  - Дерево вычислений с корнем s_0



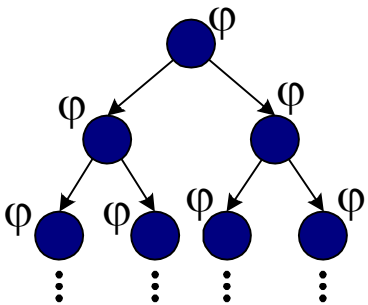
$M, s_0 \models \mathbf{EF}\varphi$



$M, s_0 \models \mathbf{AF}\varphi$



$M, s_0 \models \mathbf{EG}\varphi$



$M, s_0 \models \mathbf{AG}\varphi$

2. CTL и LTL

Типичные CTL-формулы

- **EF** (Start $\wedge$ $\neg$Ready)
  - можно достичь такого состояния, в котором условие Start выполняется, а Ready — нет
- **AG** (Req $\rightarrow$ **AF** Ack)
  - если получен запрос, то он рано или поздно будет подтвержден
- **AG** (**AF** DeviceEnabled)
  - условие DeviceEnabled выполняется бесконечно часто на каждом пути вычисления
- **AG** (**EF** Restart)
  - из любого состояния достижимо состояние Restart.

2. CTL и LTL

Композиционные доказательства, абстракция.

- Допускаются только универсальные кванторы пути
  - Вариант логики CTL* – ACTL*
  - Вариант логики CTL – ACTL
  - Неявное присутствие экзистенциальных кванторов пути
    - использование отрицания
    - допускаются только формулы в негативной нормальной форме
      - отрицания применяются только к атомарным высказываниям
    - $\wedge$, $\vee$, **U** и **R**
  - **AF**(**AG**p) и **AG**(**AX**p) – ACTL- формулы, невыразимые в логике LTL
    - Логики ACTL и LTL несравнимы
      - ACTL $\subset$ CTL
      - **AG**(**EF**p), **AG**($\neg$**AF**p) $\notin$ ACTL
    - LTL $\subset$ ACTL*

3. Справедливость

- Необходимо исследовать только некоторые выделенные пути вычисления
  - Справедливые пути
- Асинхронная схема с арбитром
  - Рассматриваются только такие срабатывания, в которых арбитр не игнорирует бесконечно долго ни один из ее запросов.
- Коммуникационные протоколы
  - Только надежные каналы связи
  - Ни одно сообщение не может непрерывно пересылаться, но при этом ни разу не быть принятым
- Такие свойства выразимы в CTL*, но не в CTL
- Изменить семантику CTL

3. Справедливость

- Новая семантика CTL– справедливая семантика
- Ограничение справедливости
  - произвольное множество состояний
  - формула
- Справедливость как множество состояний
  - На справедливом пути хотя бы один элемент из каждого ограничения справедливости должен встречаться бесконечно часто
- Справедливость как формула
  - На справедливом пути каждое ограничение выполняется бесконечно часто
- Действие кванторов пути ограничивается только справедливыми путями

3. Справедливость

- Справедливая модель Крипке M = (S, R, L, F)
  - S, R и L — те же
  - $F \subseteq 2^S$ — множество ограничений справедливости (обобщенные условия допущения Бюхи).
- $\pi = s_0, s_1, \ldots$ — путь в M
  - $\inf(\pi) = \{s \mid s = s_i$ для бесконечно многих $i\}$
- π — справедливый путь
  - если для всякого $P \in F$ верно $\inf(\pi) \cap P \neq \varnothing$
- $M, s \models_F \varphi$
  - формула состояния φ истинна в состоянии s на справедливой модели Крипке M.
- $M, \pi \models_F \psi$
  - формула пути ψ истинна на пути π в справедливой модели M

3. Справедливость

- Семантика CTL* на справедливых моделях Крипке аналогична семантике CTL* на обычных моделях Крипке
  - Изменения только в пп. 1, 5 и 6 первоначального определения семантики
- $M, s \models p \Leftrightarrow p \in L(s)$ и имеется справедливый путь, исходящий из s
- $M, s \models \mathbf{E}\ \psi \Leftrightarrow$ в M существует такой справедливый путь π из состояния s, что $M, \pi \models \psi$
- $M, s \models \mathbf{A}\ \psi \Leftrightarrow$ в M для любого справедливого пути π из состояния s верно $M, \pi \models \psi$

$M, s \models p \Leftrightarrow p \in L(s)$
$M, s \models \mathbf{E}\ \psi \Leftrightarrow$ в M существует такой путь π из состояния s, что $M, \pi \models \psi$
$M, s \models \mathbf{A}\ \psi \Leftrightarrow$ в M для любого пути π из состояния s верно $M, \pi \models \psi$

3. Справедливость

Пример использования справедливости

- Коммуникационный протокол с надежными каналами
  - Для каждого канала есть ограничение справедливости, выражающее требование надежности канала
    - Справедливость для канала i
      - множество состояний, т. ч. $\neg send_i \vee recieve_i$
  - Путь вычисления считается справедливым тогда и только тогда, когда для каждого канала бесконечно часто выполняется одно из двух свойств:
    - либо сообщение не послано
    - либо сообщение доставлено

Проверка на модели для CTL

Задача проверки моделей

- Задача проверки моделей
  - Дано
    - модель Крипке M = (S, R, L)
      - программная система с конечным числом состояний
    - формула (темпоральной) логики φ
      - спецификация
  - Найти
    - в множестве S подмножество всех состояний, в которых выполняется φ, т.е. семантику φ, множество $\{ s \in S \mid M,s \models \varphi \}$.
  - Часто некоторые состояния параллельной системы выделяются как начальные состояния.
    - Система удовлетворяет спецификации φ, если все начальные состояния принадлежат семантике φ

Задача проверки моделей

- Первые алгоритмы решения задачи проверки моделей
  - явное представление моделей Крипке в виде размеченных ориентированных графов
    - вершины – состояния из S,
    - дуги графа – отношение переходов R,
    - разметка вершин – функция $L : S \to 2^{AP}$.

1. Проверка моделей для CTL

- Задана модель Крипке M = (S,R,L) и CTL- формула φ.
- Алгоритм приписывает каждому состоянию s подформулы φ, истинные в s. Множество label(s).
  - Сначала label(s) = L(s)
  - На шаге i обрабатываются подформулы, в которых глубина вложенности CTL-операторов равна i-1.
  - После обработки подформулу добавляют к разметке всех тех состояний, в которых она истинна.
- После завершения работы алгоритма $M,s \models \varphi \Leftrightarrow \varphi \in$ label(s).

1. Проверка моделей для CTL

- Всякую CTL-формулу можно записать, используя $\neg$, $\vee$, **EX**, **EG** и **EU**.
- Достаточно уметь анализировать формулы вида:
  - $p \in AP$, $\neg\varphi$, $\varphi_1 \vee \varphi_2$, **EX**φ, **EG**φ и **E**φ_1**U**φ_2.
    - $\neg\varphi$ — помечаем все состояния, не помеченные формулой φ.
    - $\varphi_1 \vee \varphi_2$ — помечаем все состояния, помеченные либо φ_1, либо φ_2.
    - **EX**φ — помечаем все состояния, один из последователей которых помечен φ.

1. Проверка моделей для CTL

- Анализ формулы вида $\psi = \mathbf{E}\varphi_1\mathbf{U}\varphi_2$
  - Выделить все состояния, помеченные φ_2
  - Будем отступать из них назад, используя отношение, обратное отношению переходов R
    - Найти все состояния, через которые проходят пути, каждое состояние которых помечено φ_1.
  - Все такие состояния пометим ψ.
- Процедура CheckEU добавляет $\mathbf{E}\varphi_1\mathbf{U}\varphi_2$ к множеству label(s) для каждого состояния s, удовлетворяющего $\mathbf{E}\varphi_1\mathbf{U}\varphi_2$,
  - при этом φ_1 и φ_2 корректно обработаны
    - для каждого состояния s верно, что

      $\varphi_1 \in \text{label}(s) \Leftrightarrow s \models \varphi_1$, и $\varphi_2 \in \text{label}(s) \Leftrightarrow s \models \varphi_2$
- Временная сложность процедуры $O(|S|+|R|)$

1. Проверка моделей для CTL

```
procedure CheckEU(f1,f2)
        T := { s | f2∈label(s)};
        for all s∈T do label{s) := label(s)∪{E[f1 U f2]};
        while T≠∅ do
                choose s∈T;
                T := T\{s};
                for all t such that R(t,s) do
                        if E[f1 U f2]∉label(t) and f1∈label(t) then
                                label(t) := label(t)∪{E[f1 U f2]};
                                T:=T∪{t};
                        end if
                end for all
        end while
end procedure
```

Процедура разметки состояний для формулы **E**[f_1 **U** f_2]

1. Проверка моделей для CTL

- Анализ формулы вида **EG**φ
  - разбиение графа на нетривиальные сильно связные компоненты
- Сильно связная компонента (SCC) C
  - наибольший подграф, любая вершина которого достижима из всякой другой вершины этого подграфа по ориентированному пути, целиком содержащемуся в C.
- Компонента C нетривиальна
  - когда она содержит более одной вершины или в точности одну вершину с циклом-петлей, проходящим через нее
- Пусть модель M' получена из M стягиванием по φ, т.е. удалением из S всех состояний, в которых не выполняется φ и соответствующего сужения R и L.
  - Модель M' = (S',R',L'), в которой
    S' = $\{s \in S \mid M,s \models \varphi\}$, R' = $R|_{S' \times S'}$ и L'=$L|_{S'}$
  - R' может не быть тотальным
  - Состояния, из которых не исходит ни одного перехода, могут быть удалены без уменьшения общности

1. Проверка моделей для CTL

- Обоснование алгоритма
- **Лемма 1**. M,s ⊨ **EG**φ тогда и только тогда, когда
  - состояние s содержится в множестве S';
  - в M' есть путь из s в некоторую вершину t, содержащуюся в нетривиальной сильно связной компоненте C графа (S', R').
- **Доказательство**.
  - ⇨Пусть M,s ⊨ **EG**φ. Ясно, что s ∈ S'.
  - Пусть бесконечный путь π
    - исходит из s и
    - φ выполняется в каждом состоянии π.
  - Так как модель M конечна, путь $\pi = \pi_0\pi_1$, где
    - π_0 — конечный начальный отрезок
    - π_1 — бесконечный суффикс π такой, что каждое состояние в π_1 встречается в нем бесконечно часто.
  - π_0 содержится в S'.
  - Пусть C — множество состояний, через которые проходит π_1.
  - C содержится в S'.

1. Проверка моделей для CTL

- Покажем, что между любой парой состояний из C проходит путь, целиком лежащий в C.
  - Пусть $s_1, s_2 \in C$.
  - Выберем некоторое вхождение i состояния s_1 в π_1.
  - По определению пути π_1 одно из вхождений j состояния s_2 есть на пути π_1 еще дальше, т.е. i<j.
  - По определению C отрезок пути между i и j целиком лежит в C.
  - Это путь из s_1 в s_2, лежащий в C.
- C либо является сильно связной компонентой, либо содержится в одной из таких компонент.
  - В любом случае оба условия леммы выполняются.

1. Проверка моделей для CTL

- состояние s содержится в множестве S';
- в M' есть путь из s в некоторую вершину t, содержащуюся в нетривиальной сильно связной компоненте C графа (S', R').

⇦ Пусть соблюдены условия леммы

- Рассмотрим путь π_0 из s в t.
- Пусть π_1 конечный нетривиальный путь из t в t.
  - Такой путь π_1 существует, так как состояние t лежит в нетривиальной сильно связной компоненте.
- Все состояния в бесконечном пути $\pi = \pi_0\pi_1^{\omega}$ удовлетворяют φ.
- Поскольку π — один из допустимых путей, исходящих из s в модели M, то верно $M,s \models \mathbf{EG}\varphi$. ■

1. Проверка моделей для CTL

- Алгоритм для ψ = **EG**φ следует из леммы 1
  1. Построить стягиванием по φ модель Крипке M' = (S',R',L')
  2. Разбить граф (S',R') на сильно связные компоненты
     - Алгоритм Тарьяна, сложность O(|S'| + |R'|)
  3. Выделить состояния, принадлежащие сильно связным компонентам
  4. Вернуться назад, используя R^{-1}
     - определяются все состояния, лежащие на путях с состояниями, помеченными φ
- Временная сложность O(|S| + |R|)
- Процедура CheckEG добавляет **EG**φ к множеству label(s) для каждого s, в котором выполняется **EG**φ, при условии, что формула φ уже была корректно обработана.

1. Проверка моделей для CTL

```
procedure CheckEG(f)
        S' := { s | f ∈ label{s)};
        SCC := {C | C - нетривиальная SCC в S'};
        T := ∪C∈SCC{ s | s ∈ C };
        for all s ∈ T do label(s) := label(s) ∪ {EGf};
        while T ≠ ∅ do
                choose s ∈ T;
                T:= T\{s};
                for all t such that t ∈ S' and R(t,s) do
                        if EGf ∉ label(t) then
                                label(t) := label(t) ∪ {EGf};
                                T:= T ∪ {t};
                        end if
                end for all
        end while
end procedure
```

- Процедура разметки состояний для формулы **EG**φ

1. Проверка моделей для CTL

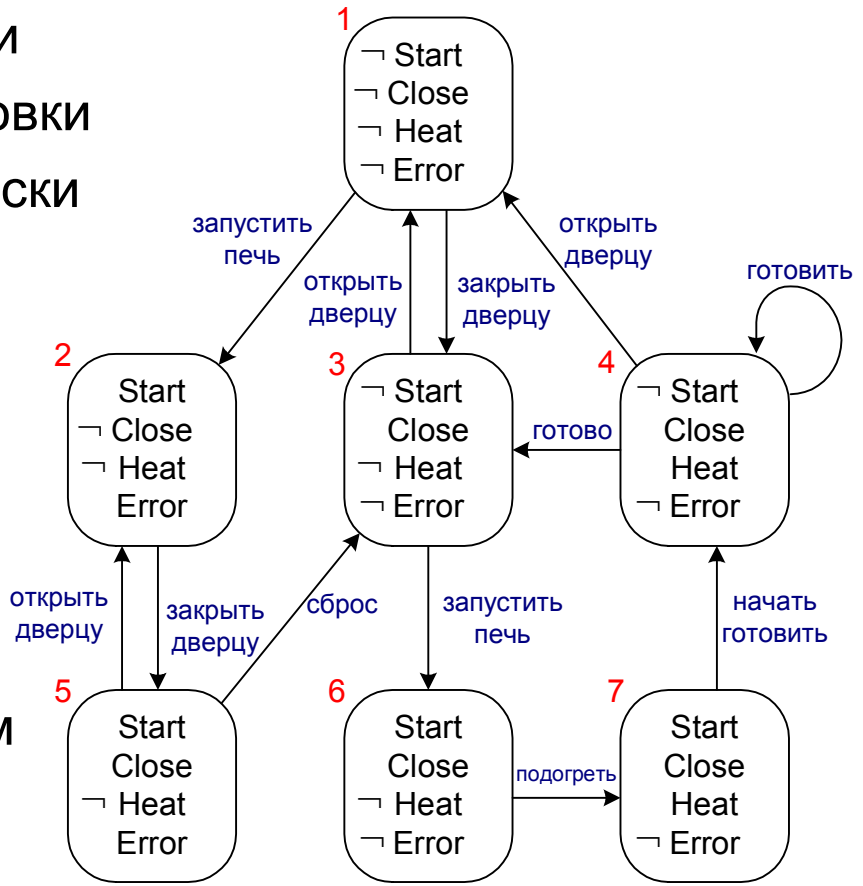
- Для произвольной CTL-формулы φ
  - применить алгоритм разметки состояний ко всем подформулам φ
    - начинать с самых коротких (наиболее глубоко вложенных)
    - продвигаться наружу, пока не доберёмся до φ
  - при анализе некоторой подформулы φ, все её подформулы обработаны
- Временная сложность алгоритма — $O(|\varphi|(|S| + |R|))$
  - Каждая подформула — $O(|S| + |R|)$
  - Формула φ содержит не более $|\varphi|$ подформул

- **Теорема 1**. Существует алгоритм проверки выполнимости произвольной CTL-формулы φ в состоянии s на модели M = (S, R, L) за время $O(|\varphi|(|S| + |R|))$.

1. Проверка моделей для CTL

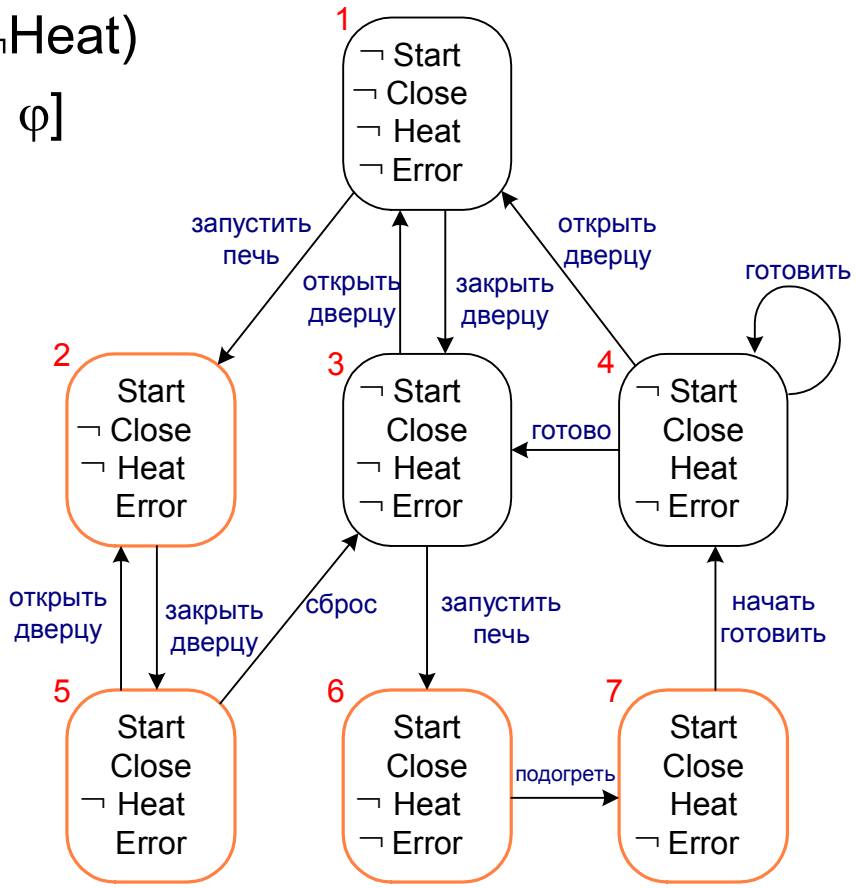
Пример работы алгоритма верификации моделей для CTL

- Поведение микроволновой печи
- Модель Крипке для микроволновки
  - Пометки в состояниях — списки
    - истинных атомарных высказываний
    - ложных атомарных высказываний
  - Пометки на дугах
    - действия, приводящие к переходам
      - не входят в модель Крипке



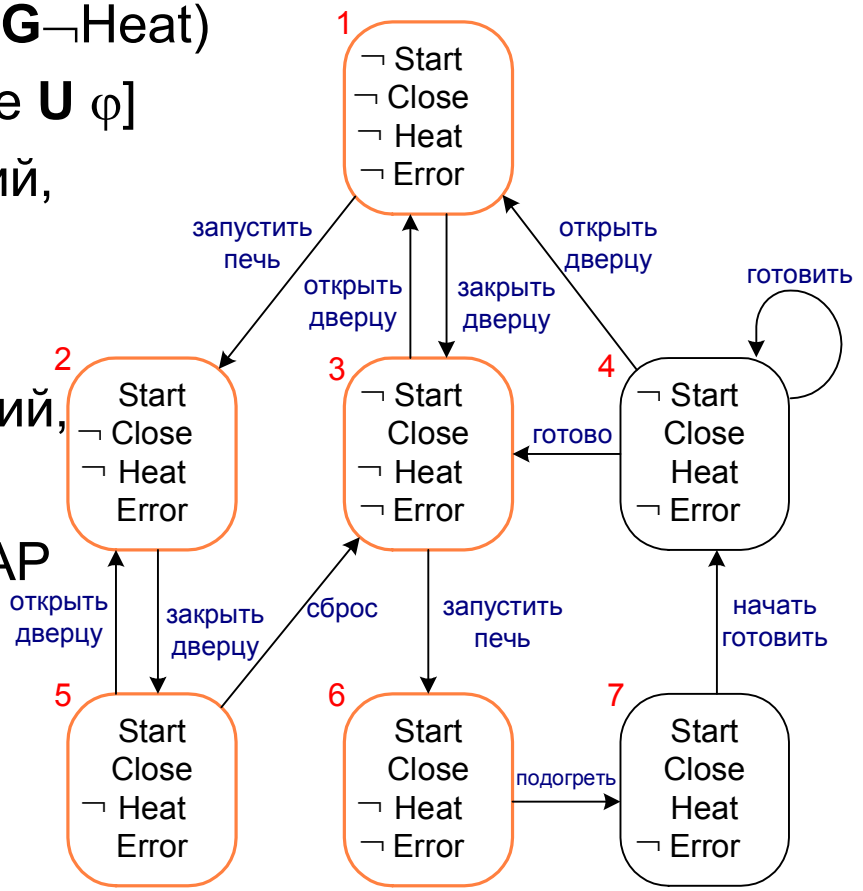
1. Проверка моделей для CTL

- Проверить CTL-формулу **AG**(Start → **AF** Heat)
  - эквивалентна ¬ **EF**(Start ∧ **EG**¬Heat)
    - **EF**φ – сокращение **E**[true **U** φ]
- Вычислить множества состояний, удовлетворяющих атомарным формулам
- S(ψ) – множество всех состояний, помеченных подформулой ψ.
- Вычисление S(p) для всех p ∈ AP занимает время O(|S|)
  - S(Start) = [2,5,6,7]
  - S{¬Heat) = [1,2,3,5,6]



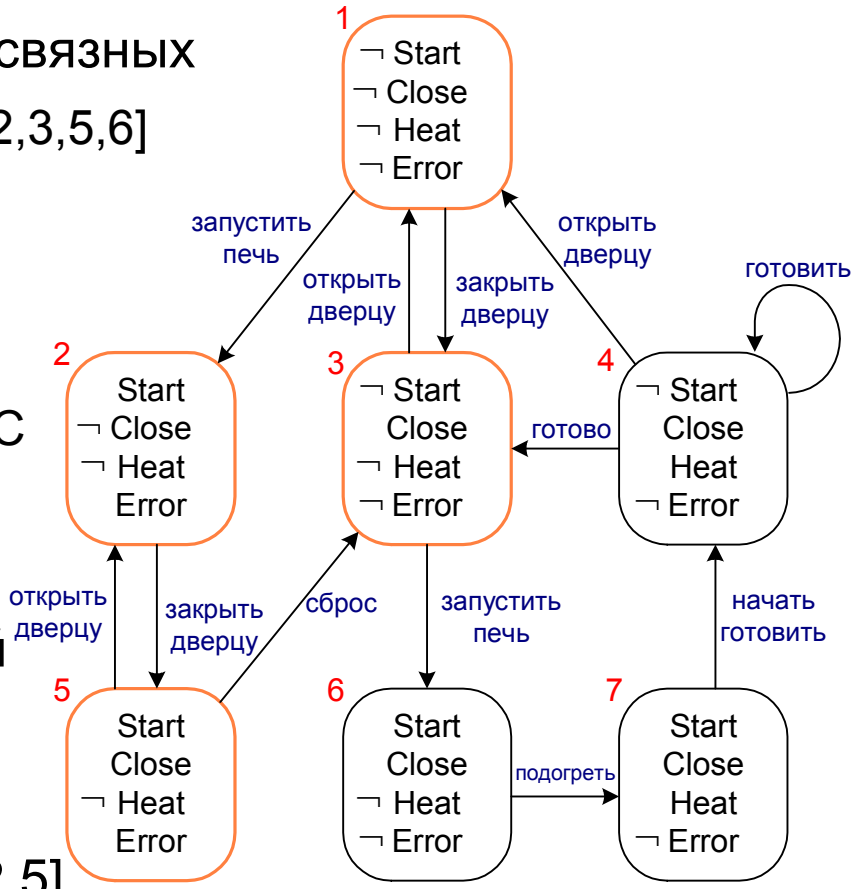
1. Проверка моделей для CTL

- Проверить CTL-формулу **AG**(Start → **AF** Heat)
  - эквивалентна ¬ **EF**(Start ∧ **EG**¬Heat)
    - **EF**φ – сокращение **E**[true **U** φ]
- Вычислить множества состояний, удовлетворяющих атомарным формулам
- S(ψ) – множество всех состояний, помеченных подформулой ψ.
- Вычисление S(p) для всех p ∈ AP занимает время O(|S|)
  - S(Start) = [2,5,6,7]
  - S{¬Heat) = [1,2,3,5,6]



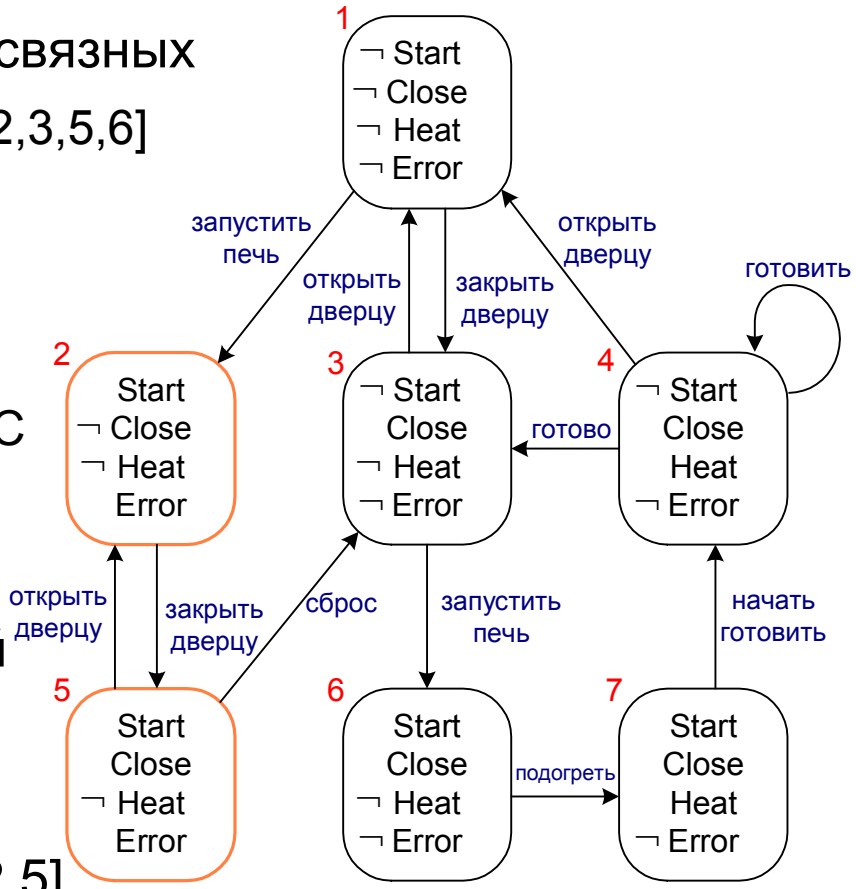
1. Проверка моделей для CTL

- Вычисление S(**EG**¬Heat)
  - Найти множество всех сильно связных компонент в S' = S(¬Heat) = [1,2,3,5,6]
    - SCC = {{1,2,3,5}}
  - T – множество состояний, удовлетворяющих **EG**¬Heat
    - Объединение множеств из SCC
      - T = {1,2,3,5}
  - Ни из какого другого состояния из S' нельзя достичь состояний из T путем, проходящим в S'
- S(**EG**¬Heat) = [1,2,3,5]
- Вычислить S(Start ∧ **EG**¬Heat) = [2,5]



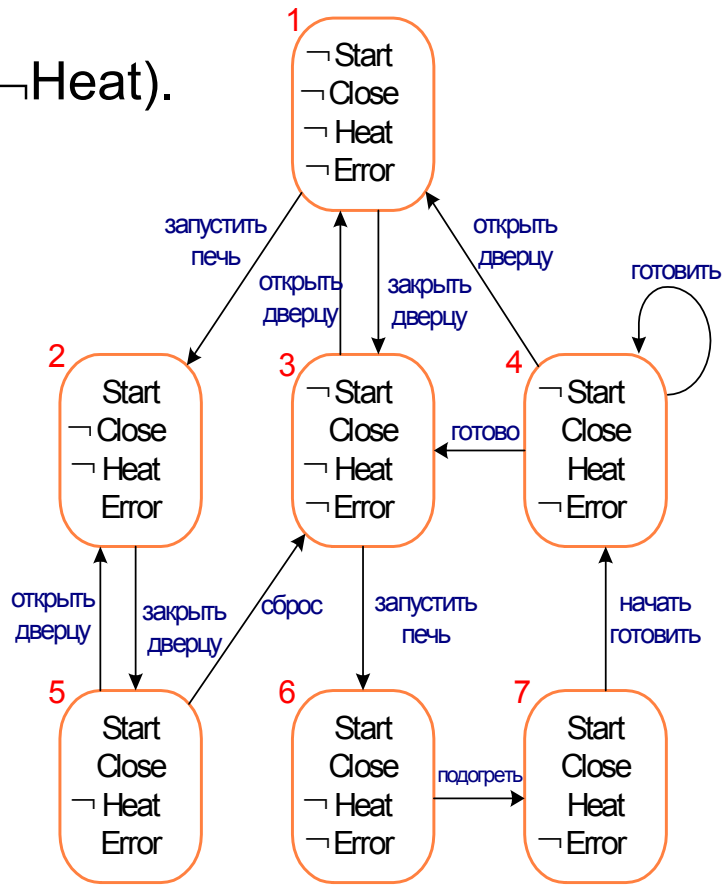
1. Проверка моделей для CTL

- Вычисление S(**EG**¬Heat)
  - Найти множество всех сильно связных компонент в S' = S(¬Heat) = [1,2,3,5,6]
    - SCC = {{1,2,3,5}}
  - T – множество состояний, удовлетворяющих **EG**¬Heat
    - Объединение множеств из SCC
      - T = {1,2,3,5}
  - Ни из какого другого состояния из S' нельзя достичь состояний из T путем, проходящим в S'
- S(**EG**¬Heat) = [1,2,3,5]
- Вычислить S(Start ∧ **EG**¬Heat) = [2,5]



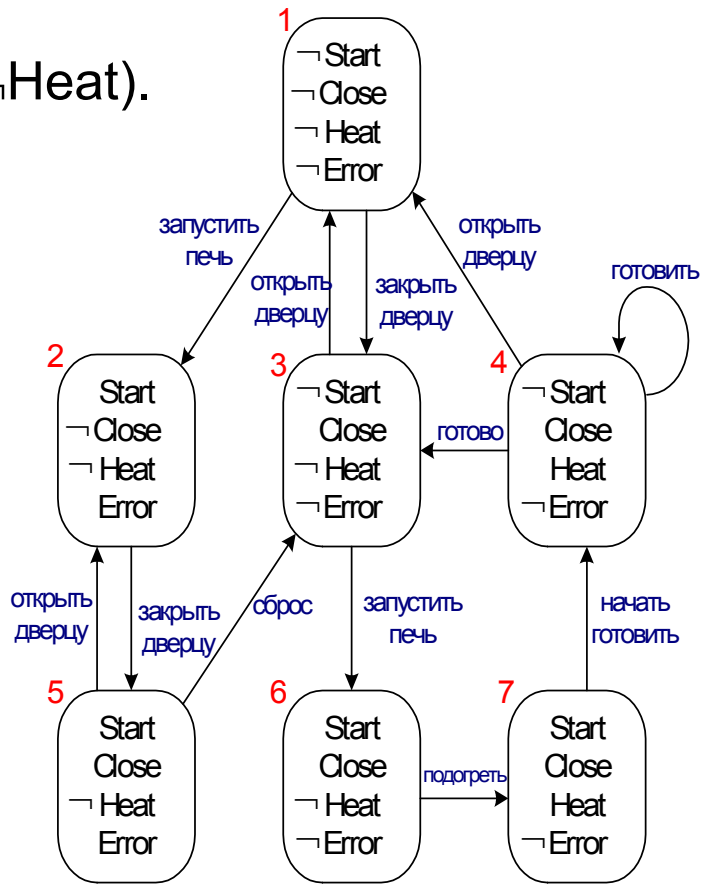
1. Проверка моделей для CTL

- Вычисление S(**EF**(Start ∧ **EG**¬Heat))
  - Начальное значение T = S(Start ∧ **EG**¬Heat).
  - Пометить все состояния, в которых верно **EF**(Start ∧ **EG**¬Heat)
    - Использовать обратное отношение переходов
- S(**EF**(Start ∧ **EG**¬Heat)) = [1,2,3,4,5,6,7]
- Вычислить S(¬(**EF**(Start ∧ **EG**¬Heat)) = ∅
- Начальное состояние 1 не содержится в этом множестве
  - Система, соответствующая представленной модели Крипке, не удовлетворяет заданной спецификации



1. Проверка моделей для CTL

- Вычисление S(**EF**(Start ∧ **EG**¬Heat))
  - Начальное значение T = S(Start ∧ **EG**¬Heat).
  - Пометить все состояния, в которых верно **EF**(Start ∧ **EG**¬Heat)
    - Использовать обратное отношение переходов
- S(**EF**(Start ∧ **EG**¬Heat)) = [1,2,3,4,5,6,7]
- Вычислить S(¬(**EF**(Start ∧ **EG**¬Heat)) = ∅
- Начальное состояние 1 не содержится в этом множестве
  - Система, соответствующая представленной модели Крипке, не удовлетворяет заданной спецификации



2. Ограничения справедливости

- Обобщение алгоритма проверки моделей для CTL с учётом ограничений справедливости.
- Пусть есть справедливая модель Крипке M = (S,R,L,F)
  - F = $\{P_1,\ldots, P_k\}$ — ограничения справедливости
- Сильно связная компонента C графа M – справедлива относительно F
  - когда для каждого $P_i \in F$ существует состояние $t_i \in C \cap P_i$

2. Ограничения справедливости

- Алгоритм для проверки **EG**φ на справедливой модели.
- Обоснование алгоритма – лемма, аналогичная лемме 1
- Пусть модель M' получена из M справедливым стягиванием по φ
  - удалить из S всех состояния, в которых φ не выполняется справедливо.
- M' = (S',R',L',F')
  - $S' = \{s \in S \mid M,s \models_F \varphi\}$, $R' = R|_{S' \times S'}$ и $L'=L|_{S'}$
- **Лемма 2**. $M,s \models_F$ **EG**φ верно тогда и только тогда, когда
  - состояние s содержится в множестве S'
  - в M' есть путь из s в вершину t, содержащуюся в нетривиальной справедливой сильно связной компоненте C графа (S', R').
- **Доказательство** леммы аналогично доказательству леммы 1

2. Ограничения справедливости

- Процедура CheckFairEG(f) добавляет **EG**φ к пометке состояния s для каждого такого s, что M,s $\models_F$ **EG**φ
- Считаем, что состояния помечены φ корректно в справедливой семантике
  - $\varphi \in$ labels(s) тогда и только тогда, когда M,s $\models_F \varphi$
- Процедура CheckFairEG аналогична процедуре CheckEG
  - Отличие заключается в том, что SCC состоит только справедливых компонент
    - Сложность – O((|S| + |R|)|F|)
      - определить справедливость компоненты
        - Убедиться, что в компоненте есть состояния из каждого ограничения справедливости

2. Ограничения справедливости

- Проверка остальных CTL-формул на справедливых моделях
  - вспомогательное атомарное высказывание **fair**, истинное в состоянии тогда и только тогда, когда существует справедливый путь, исходящий из этого состояния.
  - **fair** = **EG**true в справедливой семантике
- Разметить состояния модели новым высказыванием **fair**
  - CheckFairEG(true).

- Проверка $M, s \models_F p$ для $p \in AP$
  - проверить $M, s \models p \wedge \mathbf{fair}$
- Проверка $M, s \models_F \mathbf{EX}\varphi$
  - проверить $M, s \models \mathbf{EX}(\varphi \wedge \mathbf{fair})$
- Проверка $M, s \models_F \mathbf{E}\varphi_1\mathbf{U}\varphi_2$
  - проверить $M, s \models \mathbf{E}\varphi_1\mathbf{U}(\varphi_2 \wedge \mathbf{fair})$

2. Ограничения справедливости

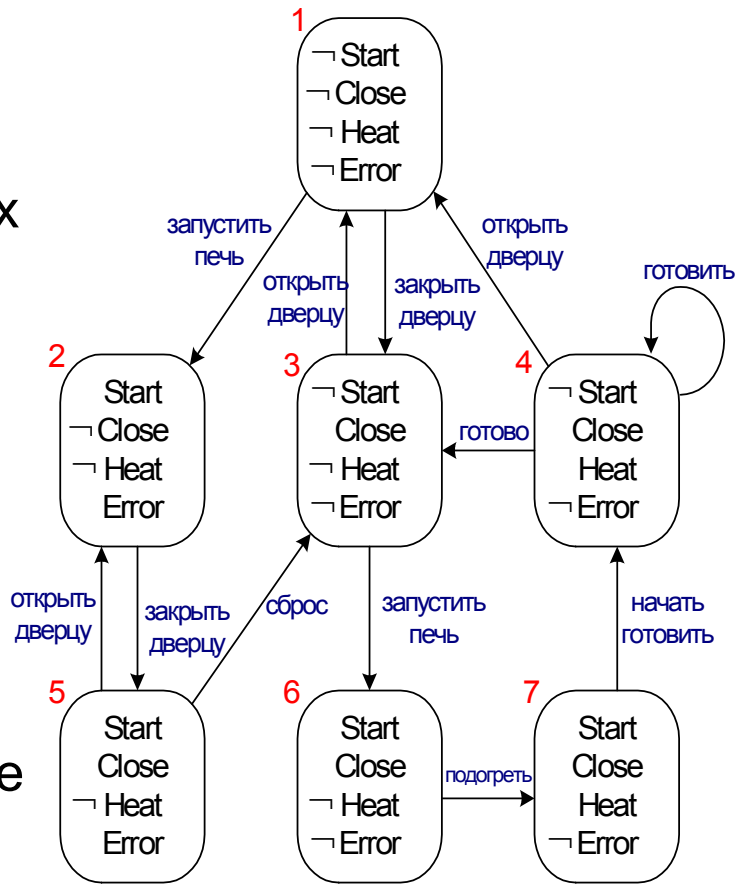
- Временная сложность алгоритма — $O(|\varphi|(|S| + |R|)|F|)$
  - Каждая подформула — $O((|S| + |R|)|F|)$
  - Формула φ содержит не более $|\varphi|$ подформул

- **Теорема 1**. Существует алгоритм проверки выполнимости произвольной CTL-формулы φ в состоянии s на модели M = (S, R, L, F) в справедливой семантике за время $O(|\varphi|(|S| + |R|)|F|)$.

2. Ограничения справедливости

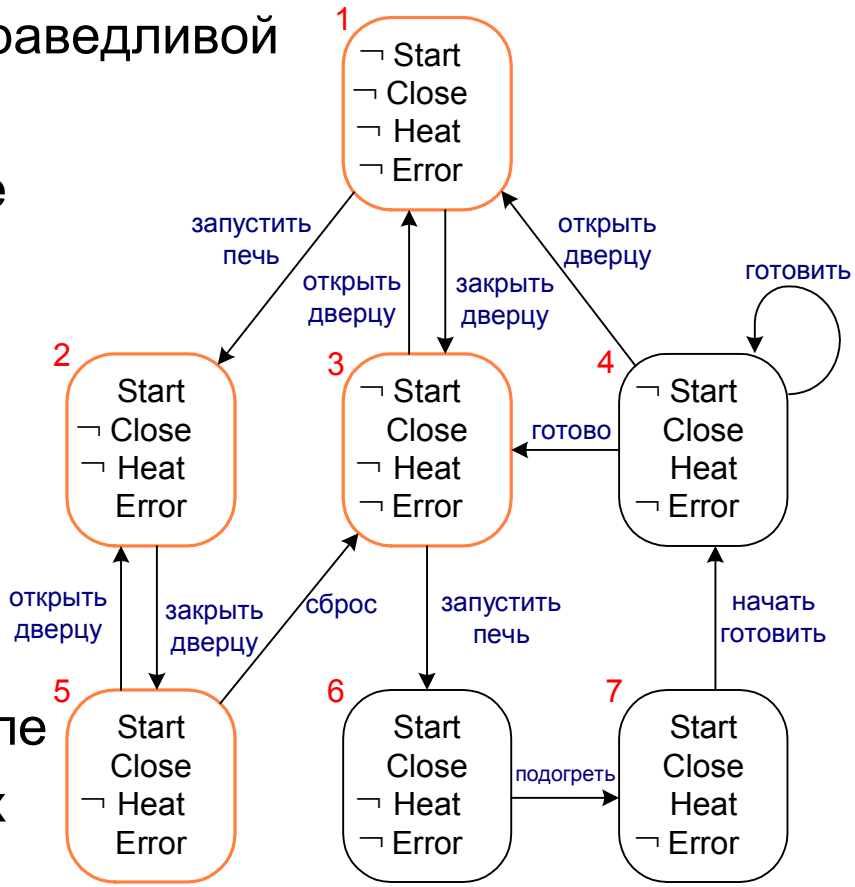
Справедливый пример с микроволновкой

- **AG**(Start → **AF** Heat)
  - Если включить, то станет греть
- Рассмотрим только те пути, на которых пользователь правильно работает с микроволновкой бесконечно часто
  - бесконечно часто должно выполняться условие Start ∧ Close ∧¬Error
  - F = {P}, где P = {s | s ⊨ Start ∧ Close ∧¬Error}.
- Множества S(Start) и S(¬Heat) такие же
  - S(Start) = [2,5,6,7]
  - S{¬Heat) = [1,2,3,5,6]



2. Ограничения справедливости

- Множество сильно связных компонент над S' = S(¬Heat)
  - SCC = {1,2,3,5} не является справедливой
    - не содержит состояний, удовлетворяющих формуле Start ∧ Close ∧¬Error
- ⇒ S(**EG**¬Heat) = ∅
- ⇒ S(**EF**(Start ∧ **EG**¬Heat)) = ∅
- ⇒ S(¬**EF**(Start ∧ **EG**¬Heat))
  = {1,2,3,4,5,6,7}



- Программа удовлетворяет формуле **AG**(Start → **AF** Heat) при заданных ограничениях справедливости.

Проверка на модели для LTL

3. Табличная проверка моделей для LTL

- Модель Крипке M = (S,R,L), $s \in S$, $\mathbf{A}\varphi \in$ LTL.
  - φ — упрощенная формула пути
    - атомарные высказывания и их булевы комбинации — формулы состояния
- Задача проверки на модели
  - верно ли, что $M,s \models \mathbf{A}\varphi$
    - $\Leftrightarrow M,s \models \neg\mathbf{E}\neg\varphi$
      - достаточно уметь проверять $\mathbf{E}\varphi$, где φ — упрощенная формула пути.
        - PSPACE-полная проблема в общем случае

3. Табличная проверка моделей для LTL

- Задача проверки моделей для формул **E**φ, где φ — упрощенная формула пути NP-полна.
  - Пусть есть граф G = (V, A), где V = {v_1,… ,v_n}.
  - Покажем, что задача проверки наличия в графе G гамильтонова пути сводима к задаче проверки M,s $\models \varphi$
    - M — конечная модель Крипке
    - s — состояние в M
    - φ — формула

      E[**F**$p_1 \wedge \ldots \wedge$ **F**$p_n \wedge$ **G**$(p_1 \rightarrow$ **XG**$\neg p_1) \wedge \ldots \wedge$ **G**$(p_n \rightarrow$ **XG**$\neg p_n)$]
      - p_1,…, p_n – атомарные высказывания

3. Табличная проверка моделей для LTL

- Построим модель M по графу G
  - каждое высказывание p_i истинно в v_i ($1 \le i \le n$), ложно в остальных вершинах
  - вершина-источник u_1, из которой достижимы все вершины v_i
    - задает единственное начальное состояние, позволяющее начать гамильтонов путь из любой вершины графа
  - вершина-сток u_2, которая достижима из всех вершин v_i
    - для тотальности отношения достижимости в модели
- Модель Крипке M — (U,B,L)
  - $U = V \cup \{u_1,u_2\}$, где $u_1,u_2 \notin V$
  - $B = A \cup \{ (u_1,v_i) \mid v_i \in V \} \cup \{ (v_i,u_2) \mid v_i \in V \} \cup \{(u_2,u_2)\}$
  - L — означивание высказываний в состояниях
    - p_i истинно в v_i для $1 \le i \le n$
    - p_j ложно в v_i для $1 \le i,j \le n$, $i \ne j$
    - p_i ложно в u_1,u_2 для $1 \le i \le n$

3. Табличная проверка моделей для LTL

- $M,u_1 \models \varphi$ верно тогда и только тогда
  - существует ориентированный бесконечный путь в M,
    - начинающийся в u_1
    - проходящий по всем $v_i \in V$ ровно один раз
    - оканчивающийся циклом-петлей, проходящим через u_2
- Формула φ имеет тот же размер, что и граф G

$$\mathbf{E}[\mathbf{F}p_1 \wedge \ldots \wedge \mathbf{F}p_n \wedge \mathbf{G}(p_1 \rightarrow \mathbf{XG}\neg p_1) \wedge \ldots \wedge \mathbf{G}(p_n \rightarrow \mathbf{XG}\neg p_n)]$$

3. Табличная проверка моделей для LTL

- Пусть длина проверяемой формулы гораздо меньше размера модели Крипке
- Будет ли сложность столь же высока?
- Сложность экспоненциальна относительно длины формулы и линейна относительно размера модели (Лихтенштейн, Пнуели)
- Алгоритм Лихтенштейна-Пнуели неявно использует табличную конструкцию
- Таблица — граф, строящийся по данной формуле
  - из графа можно извлечь модель для этой формулы в том и только том случае, когда формула выполнима
- Алгоритм проверки выполнимости формулы φ на модели M
  - строит композицию таблицы и модели,
    - чтобы выяснить, существует ли
      - такое вычисление в модели, которое одновременно
      - является путем в таблице

3. Табличная проверка моделей для LTL

- Пусть задана модель Крипке M = (S, R, L), и пусть φ — упрощенная формула пути
- Достаточно рассмотреть **X** и **U**
  - $\mathbf{F}\varphi = \mathrm{True}\mathbf{U}\varphi$, $\mathbf{G}\varphi = \neg\mathbf{F}\neg\varphi$ и $\varphi_1\mathbf{R}\varphi_2 = \neg[\neg\varphi_1 \mathrm{U} \neg\varphi_2]$.
- Замыкание $CL(\varphi)$ формулы φ содержит формулы, значения истинности которых влияют на истинностное значение формулы φ.
- Замыкание формулы φ — наименьшее множество формул, содержащее φ и такое, что
  - $\neg\psi \in CL(\varphi)$ тогда и только тогда, когда $\psi \in CL(\varphi)$
  - если $\psi_1 \vee \psi_2 \in CL(\varphi)$, то $\psi_1, \psi_2 \in CL(\varphi)$
  - если $\mathbf{X}\psi \in CL(\varphi)$, то $\psi \in CL(\varphi)$
  - если $\neg\mathbf{X}\psi \in CL(\varphi)$, то $\mathbf{X}\neg\psi \in CL(\varphi)$
  - если $\psi_1\mathbf{U}\psi_2 \in CL(\varphi)$, то $\psi_1, \psi_2, \mathbf{X}(\psi_1\mathbf{U}\psi_2) \in CL(\varphi)$
- Считаем, что $\neg\neg\psi = \psi$
- Мощность множества $CL(\varphi)$ линейна относительно размера формулы φ

3. Табличная проверка моделей для LTL

- Атом — пара $A = (s_A, K_A)$, $s_A \in S$ и $K_A \subseteq CL(\varphi) \cup AP$, и при этом
  - для любого $p \in AP$ соотношение $p \in K_A \Leftrightarrow$
    - $p \in L(s_A)$
  - для каждой $\psi \in CL(\varphi)$ соотношение $\psi \in K_A \Leftrightarrow$
    - $\neg\psi \notin K_A$
  - для каждой $\psi_1 \vee \psi_2 \in CL(\varphi)$ соотношение $\psi_1 \vee \psi_2 \in K_A \Leftrightarrow$
    - $\psi_1 \in K_A$ или $\psi_2 \in K_A$
  - для каждой $\neg\mathbf{X}\psi \in CL(\varphi)$ соотношение $\neg\mathbf{X}\psi \in K_A \Leftrightarrow$
    - $\mathbf{X}\neg\psi \in K_A$
  - для каждой $\psi_1\mathbf{U}\psi_2 \in CL(\varphi)$ соотношение $\psi_1\mathbf{U}\psi_2 \in K_A \Leftrightarrow$
    - $\psi_2 \in K_A$ или $\psi_1, \mathbf{X}[\psi_1\mathbf{U}\psi_2] \in K_A$
- Атом (s_A, K_A) определен так, что K_A — наибольшее непротиворечивое множество формул, согласованных с разметкой s_A

3. Табличная проверка моделей для LTL

- Пусть множества вершин в графе G – множество атомов
  - Пара (A, B) – ребро в графе G $\Leftrightarrow$
    - $(s_A, s_B) \in R$
    - для каждой $\mathbf{X}\psi \in CL(\varphi)$ верно $\mathbf{X}\psi \in K_A \Leftrightarrow \psi \in K_B$
- Последовательность происшествий –
  - бесконечный путь π в G такой, что
    - если $\psi_1 \mathbf{U} \psi_2 \in K_A$ для некоторого вхождения атома A из π, то
      - найдется такое вхождение атома B, достижимое из этого вхождения A на пути π, что $\psi_2 \in K_B$

3. Табличная проверка моделей для LTL

Лемма 3. $M,s \models \mathbf{E}\varphi \Leftrightarrow$

- существует хотя бы одна последовательность происшествий, начинающаяся таким атомом (s, K), что $\varphi \in K$

Доказательство.

$\Leftarrow$ Пусть есть последовательность происшествий $(s_0, K_0)\ (s_1, K_1)\ldots$, с началом в атоме (s, K) = (s_0, K_0) таком, что $\varphi \in K$

- По определению последовательность состояний $\pi = s_0, s_1,\ldots$ является путем в M, исходящим из $s = s_0$
- Надо доказать, что $\pi \models \varphi$
  - Можно показать, что
    - если $\psi \in CL(\varphi)$ — произвольная формула и $i \geq 0$, то
    - $\pi^i \models \psi$ тогда и только тогда, когда $\psi \in K_i$.

3. Табличная проверка моделей для LTL

- Доказательство индукцией по структуре формулы.
  - базис индукции $\psi \in AP$
  - индуктивные переходы для ψ вида $\neg\theta$, $\theta_1 \vee \theta_2$, $\mathbf{X}\theta$ или $\theta_1 \mathbf{U} \theta_2$.

1. ψ — атомарное высказывание
   - по определению атома $\psi \in K_i \Leftrightarrow \psi \in L(s_i)$.
2. $\psi = \neg\theta$
   - $\pi^i \models \psi \Leftrightarrow \pi^i \not\models \theta \Leftrightarrow$ [и.п.] $\theta \notin K_i \Leftrightarrow$ [опр. K_i] $\psi \in K_i$.
3. $\psi = \theta_1 \vee \theta_2$
   - $\pi^i \models \psi \Leftrightarrow \pi^i \models \theta_1$ или $\pi^i \models \theta_2 \Leftrightarrow$ [и.п.] $\theta_1 \in K_i$ или $\theta_2 \in K_i \Leftrightarrow$ [опр. K_i] $\psi \in K_i$.
4. $\psi = \mathbf{X}\theta$
   - $\pi^i \models \psi \Leftrightarrow \pi^{i+1} \models \theta \Leftrightarrow$ [и.п.] $\theta \in K_{i+1} \Leftrightarrow$ [$((s_i,K_i), (s_{i+1},K_{i+1})) \in R$] $\psi \in K_i$.

3. Табличная проверка моделей для LTL

5. $\psi = \theta_1 \mathbf{U} \theta_2 \in K_i$
   - $\Rightarrow$ Пусть $\psi \in K_i$. По опр. последовательности происшествий существует такое $j \geq i$, что $\theta_2 \in K_j$
     - $\psi \in K_i \Rightarrow$ [опр. атома] если $\theta_2 \notin K_i$, то $\theta_1 \in K_j$ и $\mathbf{X}\psi \in K_i \Rightarrow$ [опр. R] $\psi \in K_{i+1} \Rightarrow$ для каждого $i \leq k < j$ верно $\theta_1 \in K_k$
     - [и.п.] $\pi^j \models \theta_2$ и для любого $i \leq k < j$ верно $\pi^k \models \theta_1 \Rightarrow \pi^i \models \psi$
   - $\Leftarrow$ Если $\pi^i \models \psi$
     - есть такое $j \geq i$, что $\pi^j \models \theta_2$ и для всякого $i \leq k < j$ верно $\pi^k \models \theta_1$
     - Выберем наименьшее из таких j.
     - [и.п.] $\theta_2 \in K_j$ и для всякого $i \leq k < j$ верно $\theta_1 \in K_k$
     - Пусть $\psi \notin K_i \Rightarrow$[$\theta_1 \in K_i$, и опр. атома] $\mathbf{X}\psi \notin K_i \Rightarrow \mathbf{X}\neg\psi \in K_i \Rightarrow$ [опр. R] $\neg\psi \in K_{i+1} \Rightarrow \psi \notin K_{i+1}$
     - Продолжая вдоль пути найдём, что $\psi \notin K_j$. Противоречит $\theta_2 \in K_j$
- Этим завершается обоснование достаточности условий леммы.

3. Табличная проверка моделей для LTL

$\Rightarrow$ Пусть $M,s \models \mathbf{E}\varphi$

- Тогда есть такой путь $\pi = s_0, s_1 \ldots$ из состояния $s = s_0$, что $\pi \models \varphi$

■ Рассмотрим множества $K_i = \{ \psi \mid \psi \in CL(\varphi)$ и $\pi^i \models \psi \}$. Тогда

1. Пара (s_i, K_i) является атомом.
   - По опр. отношения $\models$
     - Для заданного $\psi \in CL(\varphi)$ множество K_i должно содержать либо ψ, либо $\neg\psi$, но не обе
     - $\psi \in K_i \Leftrightarrow$ [опр. K_i] $\pi^i \models \psi \Leftrightarrow \pi^i \models \neg\psi \Leftrightarrow$ [опр. K_i] $\neg\psi \notin K_i$
2. Имеется переход из (s_i,K_i) в (s_{i+1},K_{i+1}).
   - $\mathbf{X}\psi \in K_i \Leftrightarrow \pi^i \models \mathbf{X}\psi \Leftrightarrow \pi^{i+1} \models \psi \Leftrightarrow$ [опр. K_{i+1}] $\psi \in K_{i+1}$
3. Последовательность пар $(s_0,K_0), (s_1,K_1)\ldots$ является последовательностью происшествий.
   - $\psi = \theta_1 \mathbf{U} \theta_2 \in K_i \Leftrightarrow \pi^i \models \psi \Rightarrow j \geq i$ верно $\pi^j \models \theta_2 \Leftrightarrow \theta_2 \in K_j$ ■

■ Нетривиальная сильно связная компонента C графа G самодостаточна

- для каждого атома A из C и для каждой формулы $\varphi_1 \mathbf{U} \varphi_2 \in K_A$ существует такой атом B из C, что $\varphi_2 \in K_B$.

3. Табличная проверка моделей для LTL

Лемма 4. Из атома (s, K) исходит некоторая последовательность происшествий $\Leftrightarrow$ в графе G есть путь из (s, K) в какую-либо самодостаточную сильно связную компоненту.

Доказательство.

$\Rightarrow$ Пусть существует последовательность происшествий, начинающаяся с (s, K)

- Пусть множество атомов C`, встречающихся в этой последовательности бесконечно часто.
  - C` $\subseteq$ C — сильно связная компонента G
- Пусть есть подформула $\psi = \theta_1 \mathbf{U} \theta_2$ и атом (s,K)$\in$C, что $\psi \in$K
- C сильно связана $\Rightarrow$ в C есть конечный путь из (s, K) в C`
- Если θ_2 встречается на этом пути, то в C есть атом, содержащий θ_2
- Если θ_2 нет на этом пути, то ψ есть в каждом атоме на этом пути $\Rightarrow$ ψ содержится в одном из атомов множества C`.
- Поскольку C` образовалось из последовательности происшествий, $\theta_2$ содержится в атоме из C`, а значит, и в одном из атомов C.

$\Leftarrow$ Следовательно, компонента C является самодостаточной.

3. Табличная проверка моделей для LTL

⇐ Пусть есть путь из (s, K) в самодостаточную сильно связную компоненту C

- Можно построить последовательность атомов из C, в которой за каждым вхождением подформулы $\theta_1 \mathbf{U} \theta_2$ будет когда-нибудь следовать вхождение подформулы θ_2
- Подформулы $\theta_1 \mathbf{U} \theta_2$, встречающиеся на пути из (s, K) в C
  - Вхождение каждой такой подформулы либо предшествует некоторому вхождению θ_2, либо повторяется во всех атомах этого пути, до тех пор пока не будет достигнута компонента C.
  - Так как C самодостаточна, всегда можно достичь некоторого вхождения подформулы θ_2.■

3. Табличная проверка моделей для LTL

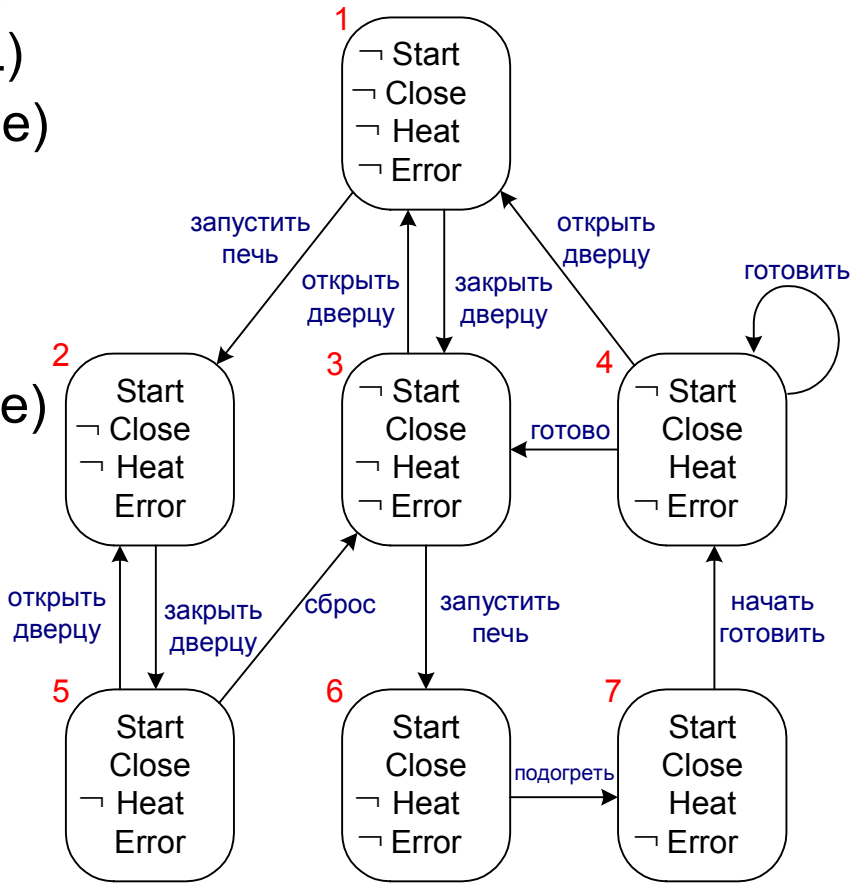
Следствие 1. $M,s \models \mathbf{E}\varphi \Leftrightarrow$

- в G найдется атом A = (s,K), такой что
  - $\varphi \in K$
  - в G существует путь из A в самодостаточную сильно связную компоненту.
- Обоснование алгоритма проверки моделей для LTL.
  - Сложность алгоритма – $O(|S|+|R|)\cdot 2^{O(|\varphi|)}$
  - Обобщение алгоритма для учёта условий справедливости
    - Лихтенштейн, Пнуели

3. Табличная проверка моделей для LTL

Пример проверки на модели для LTL

- Модель микроволновки M = (S,R,L)
  - Спецификация **A**(¬Heat **U** Close)
    - печь не может нагреваться с открытой дверцей
- Спецификация выполняется ⇔
  - ее отрицание **E**¬(¬Heat **U** Close) не выполняется.
- Пусть φ = (¬Heat) **U** Close.
  - Замыкание ¬φ : CL(¬φ) = {φ, ¬φ, **X**φ, ¬**X**φ, **X**¬φ, Heat, ¬Heat, Close, ¬Close}.
- Построим множество атомов
  - вершины графа G



3. Табличная проверка моделей для LTL

- По определению $К_A$
  - $\neg$Heat **U** Close $\in К_A \Leftrightarrow$
    - либо Close $\in К_A$
    - либо $\neg$Heat, **X**($\neg$Heat **U** Close) $\in К_A$
- $К_A$ должно быть согласовано с $L(s_A)$

- $\neg$Close, $\neg$Heat $\in$ label(1,2)
  - Формулы, ассоциированные с 1 и 2
    - $К`_1$ = {$\neg$Close, $\neg$Heat, φ, **X**φ} или
    - $К``_1$ = {$\neg$Close, $\neg$Heat, $\neg\varphi$, **X**$\neg\varphi$, $\neg$**X**φ}.
  - Атомы
    - (1,$К`_1$), (2,$К`_1$), (1,$К``_1$), (2,$К``_1$)

3. Табличная проверка моделей для LTL

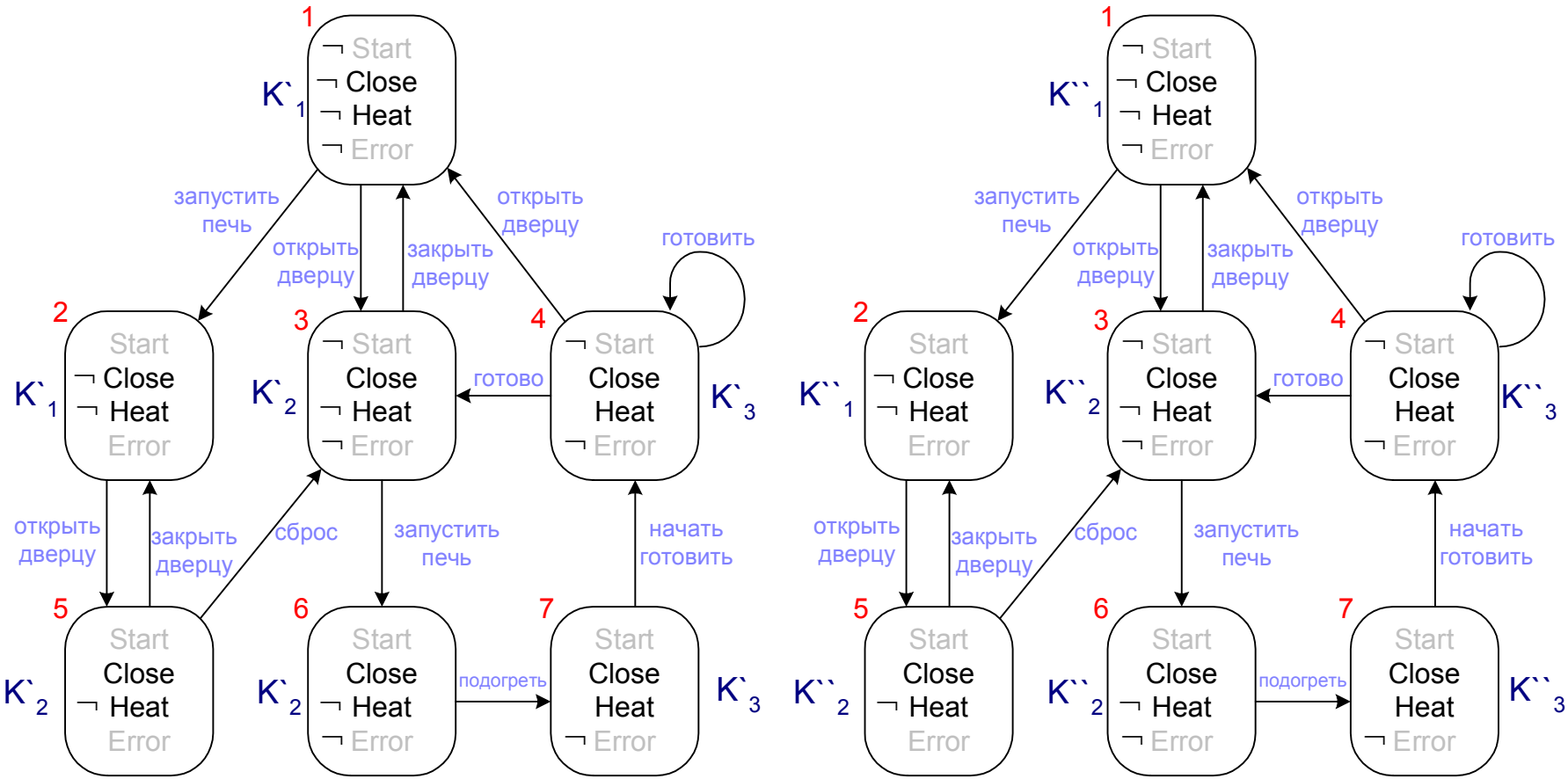
- $\neg$Heat, Close $\in$ label(3, 5, 6)
  - Формулы, ассоциированные с 3, 5, 6
    - К`$_2$ = {Close, $\neg$Heat, φ, **X**φ} или
    - К``$_2$ = {Close, $\neg$Heat, $\neg\varphi$, **X**$\neg\varphi$, $\neg$**X**φ}
- Heat, Close $\in$ label(4,7)
  - Формулы, ассоциированные с 4 и 7
    - К`$_3$ = {Close, Heat, φ, **X**φ} или
    - К``$_3$ = {Close, Heat, $\neg\varphi$, **X**$\neg\varphi$, $\neg$**X**φ}

3. Табличная проверка моделей для LTL

- Переход из атома (s_A,K_A) в атом $(s_B,K_B) \Leftrightarrow$
  - в M есть переход из s_A в s_B
  - если $\mathbf{X}\theta \in K_A$, то верно $\theta \in K_B$.
- $(1,2) \in R$, $\mathbf{X}\varphi$, $\varphi \in K`_1$, $\mathbf{X}\neg\varphi$, $\neg\varphi \in K``_1$
  - $(1,K`_1) \rightarrow (2,K`_1)$
  - $(1,K``_1) \rightarrow (2,K``_1)$
  - $(1,K`_1) \not\rightarrow (2,K``_1)$
    - $\mathbf{X}\varphi \in K`_1$, но $\varphi \notin K``_1$.
- Остальные переходы строятся аналогично

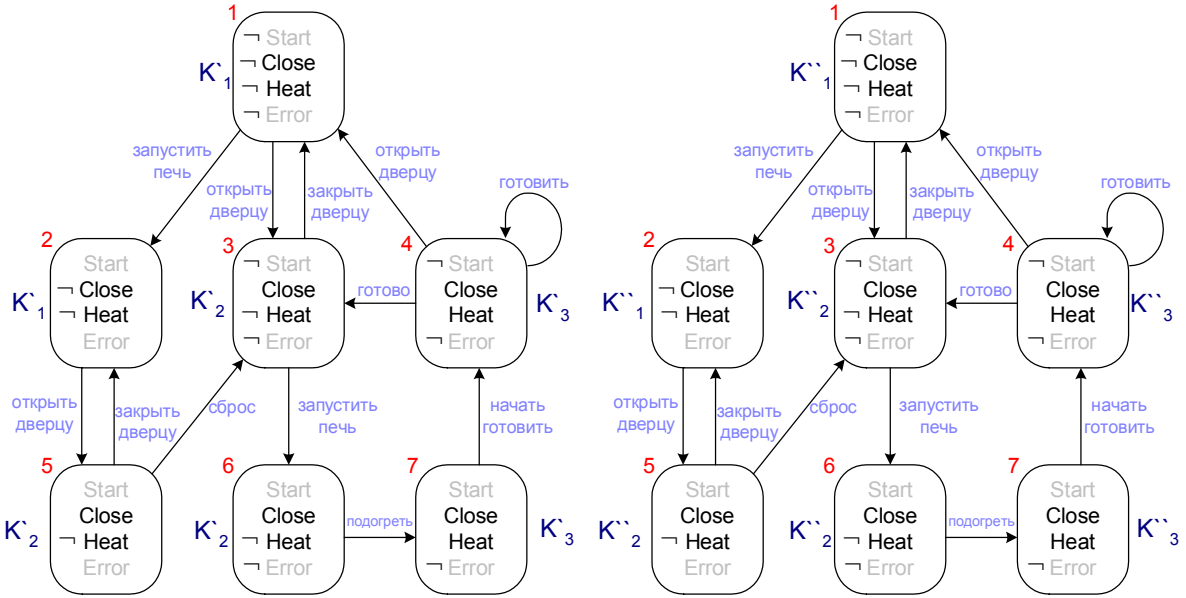
3. Табличная проверка моделей для LTL

- Остальные переходы строятся аналогично



3. Табличная проверка моделей для LTL

- По сл. 1 $\neg\varphi = \neg(\neg Heat\ \mathbf{U}\ Close)$ выполняется в $s \Leftrightarrow (s,K)$
  - $\neg\varphi \in K$
  - существует путь в графе G из (s,K) в самодостаточную сильно связную компоненту
- В построенном графе ни один из указанных атомов не лежит в начале никакого бесконечного пути
  - Ни одно состояние не удовлетворяет $\mathbf{E}\neg\varphi$
  - $\mathbf{A}\varphi$ выполняется во всех состояниях модели



4. Проверка моделей для CTL*

- Задача проверки моделей для CTL* имеет такую же сложность, что и задача проверки моделей для LTL
  - Использовать методы проверки CTL и LTL
- Алгоритм для LTL применяется к формулам вида **E**φ
  - φ — упрощенная формула пути, в которой формулами состояния могут быть только атомарные высказывания.
- Обобщить на случай формул φ, содержащих в качестве подформул произвольные формулы состояния.