Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 1

Что такое и зачем нужна формальная верификация

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Что такое формальная верификация

Наиболее популярный подход к отладке[1] вычислительной системы,[2]
известный каждому разработчику, — это тестирование:
система выполняется в разных обстоятельствах
(*на наборе тестовых сценариев*),
результаты выполнения сравниваются с ожидаемыми,
и при расхождении ищутся и устраняются причины

Зачастую[3] получить версию системы, подходящую для отладки, —
это трудоёмко, дорого или просто невозможно,
и тогда применяется имитационное моделирование:
составление и выполнение *модели* разрабатываемой системы

В большинстве случаев нельзя исследовать систему или её модель
во **всевозможных** обстоятельствах, поэтому после тестирования
и моделирования **ошибки всё равно остаются**

1 Обнаружению и устранению ошибок
2 Например, программы, микросхемы, распределённой системы и многого другого
3 При разработке микросхем, распределённых систем, больших программных и программно-аппаратных комплексов и не только

Что такое формальная верификация

> The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness.[1]
>
> Dijkstra. The humble programmer.
> Turing award speech. 1972.

Хотя с этим утверждением можно до некоторой степени поспорить, но уже в 1970-х годах обозначилась необходимость строго обосновывать отсутствие ошибок проектирования и программирования в вычислительных системах

Формальная верификация — это подход к проверке правильности вычислительных систем, устроенный в целом так:

- Придумывается формальная спецификация: набор требований, означающих правильность функционирования системы и записанных на формальном языке
- Строго математически обосновывается (или опровергается) утверждение о том, что система удовлетворяет спецификации

1 Существенно повысить уверенность в правильности программы можно только предоставив убедительное обоснование правильности

Какие бывают формальные спецификации

Формальной спецификацией может быть, например:

- «эталонная» правильная система

Тогда формальная верификация представляет собой проверку эквивалентности заданной системы и эталонной

Пример

Пусть имеется программа π_1, заведомо правильно строго обосновывать (*хотя, быть может, и неэффективно*) сортирующая массив целых чисел

Кто-то написал другую (*эффективную, но совершенно непонятную*) программу π_2 и утверждает, что она тоже сортирует массив целых чисел

Чтобы проверить справедливость этого утверждения, достаточно убедиться, что программа π_2 эквивалентна программе π_1

Какие бывают формальные спецификации

Формальной спецификацией может быть, например:

- «эталонная» правильная система

Другой пример

Цифровая микросхема при разработке, как правило, проходит через много уровней абстракции, например (*если чуть упростить реальность*):

Системный уровень (∼ SystemC): высокоуровневое алгоритмическое описание

↓

Поведенческий уровень (∼ Verilog):
способ пересылки логических значений смежду схемными регистрами

↓

Логический уровень (∼ AIG): соединение триггеров и логических вентилей

↓

Транзисторный уровень: принципиальная схема соединения МОП-транзисторов

↓

Топологический уровень: размещение КМОП-транзисторов на кристалле

Схема на предыдущем уровне служит эталоном для следующего, и описание схемы на следующем уровне должно быть эквивалентно её описанию на предыдущем уровне

Какие бывают формальные спецификации

Формальной спецификацией может быть, например:

- набор логических формул, выражающих свойство правильности системы на достаточно простом языке

Для проверки правильности системы не обязательно иметь готовую эталонную систему, достаточно

- выбрать подходящий формальный язык спецификаций,
- представить фразу «система работает правильно» формулой и
- применить известный метод проверки соответствия системы формуле

Спецификации в таком подходе зачастую основываются на

- логике предикатов (*полагаю, что все тут знают, что это такое, но всё равно позже коротко напомню*) и
- темпоральных логиках (*позволяющих описывать взаимосвязи событий, происходящих в системе в разные моменты времени*)

Этому подходу и этим языкам будет посвящена подавляющая часть курса

Что такое формальная верификация

Формальная верификация естественным образом дополняет тестирование и имитационное моделирование,

- предлагая «доказательство без выполнения» вместо «выполнения без доказательства» и
- покрывая те случаи, в которых тестирование и моделирование не могут должным образом повысить уверенность в том, что система работает «достаточно правильно»

Два основных популярных подхода к формальной верификации вычислительных систем — это

- дедуктивный анализ и
- проверка моделей (другие названия: проверка на модели; верификация моделей программ; model checking)

Дедуктивный анализ

Дедуктивный подход к формальной верификации обычно устроен так

Для системы задаётся формальная семантика:
описание поведения в строгих математических терминах

Формулируется **теорема о корректности системы**: теорема в обычном математическом понимании, утверждающая правильность системы относительно формальной семантики и формальной спецификации

Теорема о корректности **доказывается** каким-либо общепризнанным способом, в том числе

- вручную, как обычно всё доказывается в математических статьях и книгах, или
- с использованием средств автоматизированного доказательства теорем (пруверов)

Подробный рассказ о методах формальной верификации в курсе начнётся с краткого введения в дедуктивную верификацию последовательных программ

Проверка моделей (model checking)

Общая схема метода проверки моделей:

1. Спецификация системы представляется в виде формулы φ некоторого логического языка

2. Для системы строится модель M, представляющая собой
   - описание того, какие возможности пошагового выполнения имеет система, и вместе с этим
   - интерпретацию формул выбранного языка спецификаций

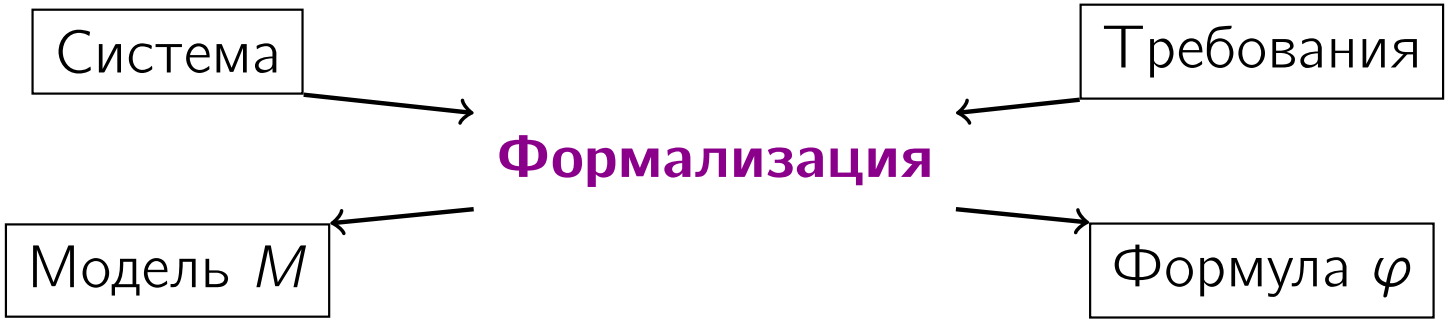
3. Проверяется выполнимость формулы φ на модели M:
$$M \models \varphi?$$

Этому подходу посвящена бóльшая часть курса

Проверка моделей (model checking)

Проверка моделей ...	**Дедуктивный анализ ...**
обычно алгоритмически разрешима	обычно алгоритмически неразрешим
обычно применяется для моделей и спецификаций простого вида — например, формула несложного пропозиционального языка и конечный автомат	ограничивает устройство моделей и спецификаций в основном только возможностями ручного доказательства пользователем
используется в основном для случаев, когда проверить выполнимость формулы на модели можно автоматически	хотя до некоторой степени и автоматизирован, но в основном выполняется вручную и требует соответствующих навыков в области построения доказательств
ввиду автоматического решения предпочтительна там, где может быть применена	может применяться для случаев, с которыми не справляется проверка моделей

Основные этапы применения проверки моделей



В начале применения метода проверки моделей строятся

- модель по реальной исследуемой системе и
- формула по (неформальной) спецификации этой системы

Моделирование системы
иногда оказывается простым и даже автоматическим,
а иногда требует большого труда и продвинутых знаний
(например, знаний о методах абстракции моделей
для уменьшения их размера — это будет обсуждаться в лекциях)

Разработать спецификацию и соответствующую формулу —
непростое дело: во многом ручное, и иногда непросто понять,
правильна ли формула; если ошибочна, то верификация бессмысленна
Этому всему будете учиться, выполняя обязательные домашние задания

Основные этапы применения проверки моделей

В идеальном случае верификация модели относительно формулы выполняется автоматически с использованием несложных алгоритмов алгебры и дискретной математики, таких как

проверка выполнимости булевых формул,

проверка достижимости вершин в графе,

проверка достижимости компонент сильной связности в графе,

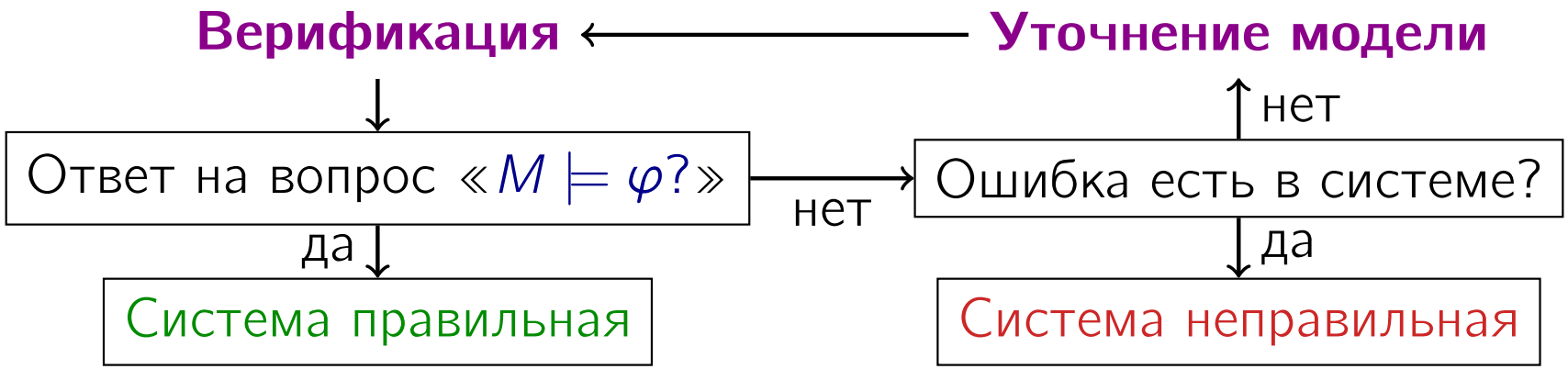
построение схем по булевым формулам,

решение систем линейных неравенств,

...

О применении таких алгоритмов в верификации будет идти речь в лекциях

Основные этапы применения проверки моделей



На практике зачастую бывает недостаточно
ответить на вопрос «$M \models \varphi?$»:
если ответ отрицательный, то в зависимости от того,
содержится ли найденная ошибка в системе, это может означать
как то, что система оказалась неправильной,
так и то, что модель следует уточнить

Существуют методы автоматического уточнения модели по найденной ошибке, *и быть может, поговорим и про них ближе к концу курса*

Зачем нужна формальная верификация?

Когда-то мир обходился и без формальной верификации вычислительных систем

Но так жить становится всё труднее

Программно-аппаратные системы повсеместно используются в критических областях (медицина, энергетика, транспорт, военные комплексы, телекоммуникации, банковский сектор, ...)

Если в такой системе возникнет ошибка, то это может привести к серьёзным последствиям и катастрофе
как при ошибочном продолжении работы,
так и при экстренном отключении

Помочь избежать такой катастрофы может
строгое обоснование отсутствия ошибок

Зачем нужна формальная верификация?

Когда-то мир обходился и без формальной верификации вычислительных систем

Но так жить становится всё труднее

Даже вне критических областей цена ошибки может быть высока:

- вычислительные системы становятся
  всё сложнее, объёмнее и дороже, а значит,
  повышается вероятность ошибок и трудоёмкость их исправления
- с возрастанием сложности систем
  усложняются и методы их разработки, и вместе с этим
  разнообразие ошибок и процессы их устранения
  без необходимости возврата к начальным этапам разработки
- в некоторых системах (таких как микросхема на кристалле)
  исправить ошибку «локально» в готовом изделии
  вовсе невозможно, исправление сопряжено
  с полным перевыпуск изделия со всеми сопутствующими затратами

Зачем нужна формальная верификация?

Когда-то мир обходился и без формальной верификации вычислительных систем

Но так жить становится всё труднее

При этом в современных вычислительных системах есть классы ошибок, которые практически невозможно обнаружить более «привычными» программисту методами, такими как тестирование

Например, это ошибки в распределённых системах, скрытые в распределённом взаимодействии компонентов

(*А в современном мире большинство систем являются распределёнными*)

Зачем нужна формальная верификация?

Понять, насколько неприятными могут быть последствия даже небольших ошибок, достаточно лишь «слегка» поискать примеры в открытом доступе:[1]

- Отдаляется «светлое будущее человечества»
  - 1962, Маринер 1; 1988, Фобос-1; 1996, Ариан-5; 1999, Mars Climate Orbiter; 2005, зонд Гюйгенс; ...
- Корпорации получают неожиданные огромные убытки
  - 1994, Intel, сотни млн. \$; 2012, Knight Capital Group, 440 млн. \$; 2009, банк UBS, чудом избежали убытка 31 млрд. \$
- Умирают люди
  - 1985–1987, 2000, аппараты радиотерапии Therac-25 и схожий
  - 1995, Боинг-757 врезался в гору
  - 2003, ошибка в системе «свой-чужой» ЗРК MIM-104 Patriot
- Страдает быт людей с широким спектром последствий
  - 2003, массовое отключение электростанций в нескольких штатах США и провинциях Канады от нескольких часов до нескольких дней

1 См., например, `https://en.wikipedia.org/wiki/List_of_software_bugs`

Зачем нужна формальная верификация?

На все эти примеры можно возразить:
«Но где гарантии, что умелое применение формальной верификации помогло бы избежать этих ошибок?»

Двойственные положительные примеры, увы,
найти и заметить намного сложнее:

- Если ошибка в программе привела к серьёзным проблемам,
  то это видно, и об этом говорят
- Если же отсутствие ошибок из-за умелого их поиска
  привело к нормальной работе программы,
  то это незаметно, и об этом не говорят
- Нередко средства, методы и результаты формальной верификации
  умалчиваются из коммерческих соображений
  и соображений безопасности

Тем не менее известны и такие примеры

Зачем нужна формальная верификация?

Некоторые достижения методов формальной верификации

1988

Использование дедуктивного анализа позволило университету Оксфорда совместно с компанией Inmos разработать микропроцессор т.н. транспьютера и язык программирования этого микропроцессора Occam, существенно ускорив разработку (по сравнению с альтернативными проектами), обнаружив неоднозначность в стандарте IEEE и ошибку в подсхеме FPU у конкурентов

Работа была удостоена награды «Queen's award for technological achievement»

Зачем нужна формальная верификация?

Некоторые достижения методов формальной верификации

1988

Компании GEC Alsthom, MATRA Transport и RATP (автономное агентство парижского транспорта) завершили проект по компьютеризации системы управления парижским метро (RER) на языке Modula-2

Использование дедуктивного анализа для обоснования правильности отдельных модулей системы позволило избежать тестирования отдельных модулей и ограничиться только глобальным тестированием

Позже по той же методике была выполнена полная автоматизация одной из линий парижского метро

Зачем нужна формальная верификация?

Некоторые достижения методов формальной верификации

1992

National Westminster Bank и компания Platform Seven завершили проект по созданию электронной платёжной системы для smart-cards

Использование дедуктивного анализа позволило обнаружить и исправить уязвимости и получить доказательство соответствия требованиям 200-страничного стандарта безопасности

Зачем нужна формальная верификация?

Некоторые достижения методов формальной верификации

1992–настоящее время

Продолжается проект системы автоматического управлени подвижным барьером для защиты Роттердама от наводнений

Метод проверки моделей (в частности, средство Spin) применяется для обнаружения ошибок в спецификациях и коде на языке C

Это средство будет изучаться в курсе

Зачем нужна формальная верификация?

Некоторые достижения методов формальной верификации

1993

При помощи методов формальной верификации обнаружены ошибки в протоколе обеспечения когерентности кэшей FutureBus+ , принятого в качестве стандарта IEEE для шины высокопроизводительных компьютеров

Для выявления ошибок использовалась проверка моделей (средство SMV)

В курсе будет рассматриваться потомок этого средства, NuSMV

Зачем нужна формальная верификация?

Некоторые достижения методов формальной верификации

2009

Группой австралийский исследователей строго доказана корректность *nix-микроядра L4 (seL4) в предположениях об устройстве и надёжности аппаратуры, на которой запускается это ядро

Для доказательства использовался дедуктивный анализ со средством автоматизированного построения доказательств Isabelle/HOL

«We can predict precisely how the kernel will behave in every possible situation»

Этого средства в курсе не будет, увы

Зачем нужна формальная верификация?

Некоторые достижения методов формальной верификации

2003. Тони Хоар инициировал «grand challenge for computing research»: создание верифицирующего компилятора

2006. Создан и пополняется «The Verified Software Repository» (VSR)

Подразделения формальных методов верификации имеются во многих крупных компаниях, занимающихся программами и аппаратурой: Microsoft, Intel, Cisco, IBM, Cadence, Mentor Graphics, ...

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 2

Логика предикатов
(напоминание)

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Сигнатура

Сигнатура логики предикатов $\langle Const, Func, Pred \rangle$ состоит из

- множества констант $Const$
- множества функциональных символов $Func$
  (символов операций)
- множества предикатных символов $Pred$
  (символов отношений)

Каждый функциональный и предикатный символ имеет вид $s^{(n)}$, где $n \in \mathbb{N} = \{1, 2, \dots\}$ — местность символа

Местность часто опускается в записи символа s

Var — счётное множество переменных

Термы и формулы

Форма Бэкуса-Наура (БНФ), задающая синтаксис
термов (выражений) и формул (условий, или булевых выражений):

$$\begin{aligned} t &::= \mathrm{x} \mid \mathbf{c} \mid \mathbf{f}(t_1, \ldots, t_n), \\ \varphi &::= \mathrm{P}(t_1, \ldots, t_n) \mid (\neg\varphi) \mid (\varphi \,\&\, \varphi) \mid (\varphi \vee \varphi) \mid \\ &\quad (\varphi \to \varphi) \mid (\exists \mathrm{x}\varphi) \mid (\forall \mathrm{x}\varphi) \mid \mathbb{t} \mid \mathbb{f}, \end{aligned}$$

где φ — **формула**, $t, t_1, \ldots, t_n$ — **термы**,
$\mathrm{x} \in \mathrm{Var}$, $\mathbf{c} \in \mathit{Const}$ и $\mathbf{f}^{(n)} \in \mathit{Func}$, $\mathrm{P}^{(n)} \in \mathit{Pred}$

Примеры

(сигнатура: $\langle\{\mathbf{3}\}, \{+^{(2)}, \cdot^{(2)}\}, \{=^{(2)}\}\rangle$)

- $\mathrm{x} + \mathbf{3} \cdot \mathrm{y}$ — терм в инфиксной записи
- $+(\mathrm{x}, \cdot(\mathbf{3}, \mathrm{y}))$ — терм в функциональной записи
- $\mathrm{x} + \mathbf{3} \cdot \mathrm{y} = \mathbf{2} + \mathrm{z}$ — формула в инфиксной записи
- $=(+(\mathrm{x}, \cdot(\mathbf{3}, \mathrm{y})), +(\mathbf{2}, \mathrm{z}))$ — формула в функциональной записи

Term — множество всех термов (заданной сигнатуры)

Термы и формулы

Приоритеты логических операций в порядке убывания:

$\exists$, $\forall$ и $\neg$; затем $\&$; затем $\vee$; затем $\to$

Ещё немного формул:

$$\forall \mathrm{x}\ \exists \mathrm{y}\ (\mathrm{y} = \mathrm{x} + 1)$$

$$\exists \mathrm{x}\ \forall \mathrm{y}\ \mathrm{P}(\mathrm{x}, \mathrm{y}) \to \forall \mathrm{y}\ \exists \mathrm{x}\ \mathrm{P}(\mathrm{x}, \mathrm{y})$$

Свободные и связанные переменные

Квантор связывает ту переменную, которая следует за ним

Область действия внешнего квантора
в формулах вида $\forall \mathtt{x}\, \varphi$ и $\exists \mathtt{x}\, \varphi$ — это подформула φ

Связанное вхождение переменной в формулу — это
вхождение переменной в область действия квантора,
связывающего эту переменную

Свободное вхождение переменной — это
вхождение, не являющееся связанным

Свободная переменная формулы — это
переменная, имеющая свободное вхождение в формулу

Пример:

$$\exists \mathtt{y}(\forall \mathtt{x} \neg \mathrm{P}(\mathtt{y}, \mathbf{f}(\mathtt{x}, \mathtt{y})) \to \mathrm{R}(\mathtt{x}))$$

Свободные и связанные переменные

Квантор связывает ту переменную, которая следует за ним

Область действия внешнего квантора
в формулах вида $\forall \mathtt{x}\, \varphi$ и $\exists \mathtt{x}\, \varphi$ — это подформула φ

Связанное вхождение переменной в формулу — это
вхождение переменной в область действия квантора,
связывающего эту переменную

Свободное вхождение переменной — это
вхождение, не являющееся связанным

Свободная переменная формулы — это
переменная, имеющая свободное вхождение в формулу

Пример:

$$\exists \mathtt{y}(\forall \mathtt{x} \neg \mathrm{P}(\mathtt{y}, \mathbf{f}(\mathtt{x}, \mathtt{y})) \to \mathrm{R}(\mathtt{x}))$$

Переменная $\mathtt{y}$ связана квантором $\exists$

Свободные и связанные переменные

Квантор связывает ту переменную, которая следует за ним

Область действия внешнего квантора
в формулах вида $\forall \mathtt{x}\, \varphi$ и $\exists \mathtt{x}\, \varphi$ — это подформула φ

Связанное вхождение переменной в формулу — это
вхождение переменной в область действия квантора,
связывающего эту переменную

Свободное вхождение переменной — это
вхождение, не являющееся связанным

Свободная переменная формулы — это
переменная, имеющая свободное вхождение в формулу

Пример:

$$\exists \mathtt{y}(\forall \mathtt{x} \neg \mathrm{P}(\mathtt{y}, \mathbf{f}(\mathtt{x}, \mathtt{y})) \to \mathrm{R}(\mathtt{x}))$$

Переменная $\mathtt{x}$ связана квантором $\forall$

Свободные и связанные переменные

Квантор связывает ту переменную, которая следует за ним

Область действия внешнего квантора
в формулах вида $\forall \mathrm{x}\, \varphi$ и $\exists \mathrm{x}\, \varphi$ — это подформула φ

Связанное вхождение переменной в формулу — это
вхождение переменной в область действия квантора,
связывающего эту переменную

Свободное вхождение переменной — это
вхождение, не являющееся связанным

Свободная переменная формулы — это
переменная, имеющая свободное вхождение в формулу

Пример:

$$\exists \mathrm{y}(\forall \mathrm{x} \neg \mathrm{P}(\mathrm{y}, \mathbf{f}(\mathrm{x}, \mathrm{y})) \to \mathrm{R}(\mathrm{x}))$$

Область действия квантора $\exists$

Свободные и связанные переменные

Квантор связывает ту переменную, которая следует за ним

Область действия внешнего квантора
в формулах вида $\forall \mathrm{x}\, \varphi$ и $\exists \mathrm{x}\, \varphi$ — это подформула φ

Связанное вхождение переменной в формулу — это
вхождение переменной в область действия квантора,
связывающего эту переменную

Свободное вхождение переменной — это
вхождение, не являющееся связанным

Свободная переменная формулы — это
переменная, имеющая свободное вхождение в формулу

Пример:

$$\exists \mathrm{y}(\forall \mathrm{x} \neg \mathrm{P}(\mathrm{y}, \mathbf{f}(\mathrm{x}, \mathrm{y})) \to \mathrm{R}(\mathrm{x}))$$

Область действия квантора $\forall$

Свободные и связанные переменные

Квантор связывает ту переменную, которая следует за ним

Область действия внешнего квантора
в формулах вида $\forall \mathtt{x}\, \varphi$ и $\exists \mathtt{x}\, \varphi$ — это подформула φ

Связанное вхождение переменной в формулу — это
вхождение переменной в область действия квантора,
связывающего эту переменную

Свободное вхождение переменной — это
вхождение, не являющееся связанным

Свободная переменная формулы — это
переменная, имеющая свободное вхождение в формулу

Пример:

$$\exists \mathtt{y}(\forall \mathtt{x} \neg \mathrm{P}(\mathtt{y}, \mathbf{f}(\mathtt{x}, \mathtt{y})) \to \mathrm{R}(\mathtt{x}))$$

Связанные вхождения переменной $\mathtt{y}$

Свободные и связанные переменные

Квантор связывает ту переменную, которая следует за ним

Область действия внешнего квантора
в формулах вида $\forall \mathrm{x}\, \varphi$ и $\exists \mathrm{x}\, \varphi$ — это подформула φ

Связанное вхождение переменной в формулу — это
вхождение переменной в область действия квантора,
связывающего эту переменную

Свободное вхождение переменной — это
вхождение, не являющееся связанным

Свободная переменная формулы — это
переменная, имеющая свободное вхождение в формулу

Пример:

$$\exists \mathrm{y}(\forall \mathrm{x} \neg \mathrm{P}(\mathrm{y}, \mathbf{f}(\mathrm{x}, \mathrm{y})) \to \mathrm{R}(\mathrm{x}))$$

Связанное вхождение переменной x

Свободные и связанные переменные

Квантор связывает ту переменную, которая следует за ним

Область действия внешнего квантора
в формулах вида $\forall \mathrm{x}\, \varphi$ и $\exists \mathrm{x}\, \varphi$ — это подформула φ

Связанное вхождение переменной в формулу — это
вхождение переменной в область действия квантора,
связывающего эту переменную

Свободное вхождение переменной — это
вхождение, не являющееся связанным

Свободная переменная формулы — это
переменная, имеющая свободное вхождение в формулу

Пример:

$$\exists \mathrm{y}(\forall \mathrm{x} \neg \mathrm{P}(\mathrm{y}, \mathbf{f}(\mathrm{x}, \mathrm{y})) \to \mathrm{R}(\mathrm{x}))$$

Свободное вхождение переменной x

Интерпретации, выполнимость, истинность

Интерпретация (сигнатуры $\langle \mathit{Const}, \mathit{Func}, \mathit{Pred} \rangle$) состоит из

- предметной области D
  - (это произвольное непустое множество предметов, на котором задана интерпретация)
- оценки констант
  - оценка константы $\mathbf{c}$ — это предмет $\overline{\mathbf{c}} \in D$
- оценки функциональных символов
  - оценка функционального символа $\mathbf{f}^{(k)}$ — это функция $\overline{\mathbf{f}} : D^k \to D$
- оценки предикатных символов
  - оценка предикатного символа $\mathrm{P}^{(k)}$ — это предикат $\overline{\mathrm{P}} : D^k \to \{\mathbb{t}, \mathbb{f}\}$

Интерпретации, выполнимость, истинность

Оценка переменных множества V, $V \subseteq \mathrm{Var}$, в интерпретации на D — это отображение вида $\sigma : V \to D$

Связка оценки переменных — это запись вида x/d, где $\mathrm{x} \in \mathrm{Var}$ и $d \in D$

Связка x/d означает, что переменная x оценивается предметом d

Оценку σ переменных конечного множества $V = \{\mathrm{x}_1, \ldots, \mathrm{x}_n\}$
можно представить в виде конечного множества связок,
в котором вместо фигурных скобок обычно изображаются квадратные:

$$\sigma = [\mathrm{x}_1/\sigma(\mathrm{x}_1), \ldots, \mathrm{x}_n/\sigma(\mathrm{x}_n)]$$

Записью $\sigma[\mathrm{x} \leftarrow d]$, где $\mathrm{x} \in \mathrm{Var}$, $d \in D$ и σ — оценка переменных V, будем обозначать оценку переменных $V \cup \{\mathrm{x}\}$, отличающуюся от σ только тем, что x оценивается предметом d:

- $\sigma[\mathrm{x} \leftarrow d](\mathrm{x}) = d$
- Для остальных переменных y верно $\sigma[\mathrm{x} \leftarrow d](\mathrm{y}) = \sigma(\mathrm{y})$

Интерпретации, выполнимость, истинность

Если все переменные терма t принадлежат множеству переменных V, то значение терма на оценке σ переменных V ($t\sigma$) в интерпретации $\mathcal{I}$ — это предмет, задающийся так:

- $\mathbf{c}\sigma = \overline{\mathbf{c}}$
- $\mathrm{x}\sigma = \sigma(\mathrm{x})$
- $\mathbf{f}(t_1, \ldots, t_n)\sigma = \overline{\mathbf{f}}(t_1\sigma, \ldots, t_n\sigma)$

Например, если предметная область интерпретации $\mathcal{I}_{ar}$ — все целые числа ($\mathbb{Z}$) и все символы сигнатуры оцениваются естественно (будем называть такую интерпретацию целочисленной арифметической), то

$$\overline{\mathbf{2}}\,\overline{+}\,\overline{\mathbf{2}} \quad \equiv \quad 4$$

Интерпретации, выполнимость, истинность

Если все свободные переменные формулы φ
принадлежат множеству переменных V,
то выполнимость формулы φ в интерпретации $\mathcal{I}$
на оценке σ переменных V ($\mathcal{I} \models \varphi\sigma$) задаётся так:

- Обязательно верно $\mathcal{I} \models \mathbb{t}\sigma$ и $\mathcal{I} \not\models \mathbb{f}\sigma$
- $\mathcal{I} \models \mathrm{P}(t_1, \ldots, t_k)\sigma \Leftrightarrow \overline{P}(t_1\sigma, \ldots, t_k\sigma) = \mathbb{t}$
- $\mathcal{I} \models (\neg\psi)\sigma \Leftrightarrow \mathcal{I} \not\models \psi\sigma$
- $\mathcal{I} \models (\psi_1 \,\&\, \psi_2)\sigma \Leftrightarrow \mathcal{I} \models \psi_1\sigma$ и $\mathcal{I} \models \psi_2\sigma$
- $\mathcal{I} \models (\psi_1 \vee \psi_2)\sigma \Leftrightarrow$ верно хотя бы одно из двух: $\mathcal{I} \models \psi_1\sigma$; $\mathcal{I} \models \psi_2\sigma$
- $\mathcal{I} \models (\psi_1 \to \psi_2)\sigma \Leftrightarrow \mathcal{I} \models (\neg\psi_1 \vee \psi_2)\sigma$
- $\mathcal{I} \models (\exists \mathrm{x}\, \psi)\sigma \Leftrightarrow$ существует предмет d, такой что $\mathcal{I} \models \psi\sigma[\mathrm{x} \leftarrow d]$
- $\mathcal{I} \models (\forall \mathrm{x}\, \psi)\sigma \Leftrightarrow$ для любого предмета d верно $\mathcal{I} \models \psi\sigma[\mathrm{x} \leftarrow d]$

Формула φ истинна в интерпретации $\mathcal{I}$ ($\mathcal{I} \models \varphi$), если для любой оценки σ свободных переменных формулы φ верно $\mathcal{I} \models \varphi\sigma$

Интерпретации, выполнимость, истинность

Примеры

$$
\begin{aligned}
\mathcal{I}_{ar} &\models (\mathrm{x} = \mathrm{x})[\mathrm{x}/1] \\
\mathcal{I}_{ar} &\models (\mathrm{x} = \mathrm{x})[\mathrm{x}/2, \mathrm{y}/5] \\
\mathcal{I}_{ar} &\models \mathrm{x} = \mathrm{x} \\
\mathcal{I}_{ar} &\models (\mathrm{x} = \mathbf{1})[\mathrm{x}/1] \\
\mathcal{I}_{ar} &\not\models (\mathrm{x} = \mathbf{1})[\mathrm{x}/2] \\
\mathcal{I}_{ar} &\not\models \mathrm{x} = \mathbf{1} \\
\mathcal{I}_{ar} &\models (\mathrm{y} = \mathrm{x} + \mathbf{1})[\mathrm{x}/3, \mathrm{y}/4] \\
\mathcal{I}_{ar} &\models \exists \mathrm{y}\ (\mathrm{y} = \mathrm{x} + \mathbf{1}) \\
\mathcal{I}_{ar} &\models \forall \mathrm{x}\ \exists \mathrm{y}\ (\mathrm{y} = \mathrm{x} + \mathbf{1}) \\
\mathcal{I}_{ar} &\not\models \exists \mathrm{y}\ (\mathrm{x} = \mathbf{2} * \mathrm{y})
\end{aligned}
$$

Подстановки

Подстановка — это отображение $\theta : \mathrm{Var} \to \mathrm{Term}$

Связка подстановки — это запись вида $\mathtt{x}/t$, где $\mathtt{x} \in \mathrm{Var}$ и $t \in \mathrm{Term}$

Связка $\mathtt{x}/t$ означает, что при применении подстановки
на место $\mathtt{x}$ подставляется терм t

Область подстановки θ — это множество всех переменных $\mathtt{x}$,
для которых верно неравенство $\theta(\mathtt{x}) \neq \mathtt{x}$

Подстановка считается конечной, если конечна её область

Конечную подстановку θ с областью $V = \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$
можно представить в виде конечного множества связок:

$$\theta = \{\mathtt{x}_1/\theta(\mathtt{x}_1), \ldots, \mathtt{x}_n/\theta(\mathtt{x}_n)\}$$

Подстановки

$E\theta$ — это результат применения подстановки θ к выражению E, то есть выражение, получающееся из E заменой

- каждого вхождения каждой переменной $\mathtt{x}$ на соответствующий терм $\theta(\mathtt{x})$, если E — терм
- каждого свободного вхождения каждой свободной переменной $\mathtt{x}$ на соответствующий терм $\theta(\mathtt{x})$, если E — формула

Например,

$$\begin{aligned}(\mathtt{x}+\mathtt{x}*\mathtt{y}*\mathtt{z})\{\mathtt{x}/\mathtt{y},\mathtt{y}/\mathtt{z}+\mathbf{2}\} &\equiv \mathtt{y}+\mathtt{y}*(\mathtt{z}+\mathbf{2})*\mathtt{z}\\ (\mathtt{x}=\mathtt{y}\vee\exists\mathtt{x}\,(\mathtt{x}=\mathtt{y}))\{\mathtt{x}/\mathbf{1},\mathtt{y}/\mathbf{2}\} &\equiv \mathbf{1}=\mathbf{2}\vee\exists\mathtt{x}\,(\mathtt{x}=\mathbf{2})\end{aligned}$$

Композиция подстановок θ и η — это подстановка $\theta\eta$, такая что для любой переменной $\mathtt{x}$ верно равенство

$$\mathtt{x}(\theta\eta)=(\mathtt{x}\theta)\eta$$

Например,

$$\{\mathtt{x}/\mathbf{f}(\mathtt{x},\mathbf{c}),\ \mathtt{y}/\mathbf{g}(\mathtt{u}),\ \mathtt{z}/\mathtt{y}\}\{\mathtt{x}/\mathbf{g}(\mathtt{y}),\mathtt{y}/\mathtt{z},\mathtt{u}/\mathbf{c}\}=\{\mathtt{x}/\mathbf{f}(\mathbf{g}(\mathtt{y}),\mathbf{c}),\ \mathtt{y}/\mathbf{g}(\mathbf{c}),\ \mathtt{u}/\mathbf{c}\}$$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 3

Общие принципы дедуктивной верификации программ

Модельные императивные программы:
синтаксис,
операционная семантика

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Принципы дедуктивной верификации программ

1. Программа π вычисляет отношение R_π между данными, которые подаются на вход и получаются на выходе

2. Текст программы π — это формальное описание отношения R_π

3. Спецификация Φ программы — это формальное описание отношения R_Φ между данными программы
   - Отношение, описываемое спецификацией, — это требования, которым должно удовлетворять отношение, вычисляемое программой

4. Формальная верификация программы π относительно спецификации Φ — это строгое доказательство того, что программа π удовлетворяет требованиям Φ, то есть доказательство включения $R_\pi \subseteq R_\Phi$

Принципы дедуктивной верификации программ

Чтобы уметь формально верифицировать программы, нужно:

1. Строго описать,
   - какие записи мы считаем программами (синтаксис программ)
   - как программы преобразуют входные данные в выходные (семантику программ)

2. Выбрать формальный язык описания требований к программам

3. Предложить метод проверки того, удовлетворяет ли заданная программа предъявленным к ней требованиям

Синтаксис программ

Синтаксис императивных программ (заданной сигнатуры σ) зададим следующей БНФ:

$$
\begin{array}{lll}
\pi & ::= & stmt \mid stmt\ \pi \\
stmt & ::= & \emptyset \mid \qquad \text{(пустая команда)} \\
 & & \mathtt{x} := t; \mid \qquad \text{(присваивание)} \\
 & & \textbf{if } C \textbf{ then } \pi \textbf{ else } \pi \textbf{ fi} \mid \qquad \text{(ветвление)} \\
 & & \textbf{while } C \textbf{ do } \pi \textbf{ od} \qquad \text{(цикл)}
\end{array}
$$

Здесь:

- π — программа
- $stmt$ — команда программы (или, по-другому, инструкция)
- $\mathtt{x} \in \mathrm{Var}$
- t — выражение: произвольный терм
- C — условие: произвольная формула без $\forall$ и $\exists$

Синтаксис программ

Пример: реализация алгоритма Эвклида
вычисления наибольшего общего делителя чисел в переменных $\mathtt{x}$, $\mathtt{y}$

while $\neg(\mathtt{x} = \mathtt{y})$ **do**
  if $\mathtt{x} > \mathtt{y}$ **then**
    $\mathtt{x} := \mathtt{x} - \mathtt{y};$
  else
    $\mathtt{y} := \mathtt{y} - \mathtt{x};$
  fi
od

Виды семантики программ

Два основных способа определения семантики программ:

денотационный и операционный

Денотационный:

- Значение каждой части программы — это денотация (денотат, denotation), особый математический объект
- Денотация программы задаётся как композиция денотаций её составных частей
- Денотационной семантикой не определяется способ вычисления денотации на входных данных
- Хорошо подходит для описания значения функциональных программ
- Денотации функциональных программ — это функции

Виды семантики программ

Два основных способа определения семантики программ:

денотационный и операционный

Операционный:

- Вычисление программы — это последовательное изменение текущего состояния вычисления
- Программой задаётся отношение переходов, описывающее то, какое состояние будет следующим в вычислении относительно произвольного текущего состояния
- Значение программы — это функция преобразования входных данных в выходные, определяемая на основе *рефлексивно-транзитивного замыкания* отношения переходов
- Хорошо подходит для описания значения императивных программ

Операционная семантика программ

Состояние управления — это произвольная программа

Состояние данных — это произвольная оценка всех переменных Var

Состояние вычисления — это пара $\langle \pi \mid \sigma \rangle$, где
π — состояние управления и σ — состояние данных

Σ — так будем обозначать множество всех состояний данных

В примерах будем записывать состояния данных
как оценки только «заслуживающих внимания» переменных,
считая, что значения остальных переменных неважны

Операционная семантика программ

Отношение переходов $\xrightarrow{\mathcal{I}}$ на множестве состояний вычисления в интерпретации $\mathcal{I}$ определяется так:

- $\langle \mathrm{x} := t; \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \sigma[\mathrm{x} \leftarrow t\sigma] \rangle$
- Если $\mathcal{I} \models C\sigma$, то $\langle \textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi} \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \pi_1 \mid \sigma \rangle$
- Если $\mathcal{I} \not\models C\sigma$, то $\langle \textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi} \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \pi_2 \mid \sigma \rangle$
- Если $\mathcal{I} \models C\sigma$, то $\langle \textbf{while } C \textbf{ do } \pi \textbf{ od} \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \pi \textbf{ while } C \textbf{ do } \pi \textbf{ od} \mid \sigma \rangle$
- Если $\mathcal{I} \not\models C\sigma$, то $\langle \textbf{while } C \textbf{ do } \pi \textbf{ od} \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \emptyset \mid \sigma \rangle$
- Если $\langle \pi_1 \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \pi_1' \mid \sigma' \rangle$, то $\langle \pi_1 \; \pi_2 \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \pi_1' \; \pi_2 \mid \sigma' \rangle$
- $\langle \emptyset \; \pi \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \pi \mid \sigma \rangle$

Операционная семантика программ

Трасса программы π на оценке σ в интерпретации $\mathcal{I}$ — это последовательность состояний вычисления вида

$$\langle \pi \mid \sigma \rangle \xrightarrow{\mathcal{I}} \langle \pi_1 \mid \sigma_1 \rangle \xrightarrow{\mathcal{I}} \langle \pi_2 \mid \sigma_2 \rangle \xrightarrow{\mathcal{I}} \dots$$

Вычисление программы π на оценке σ в интерпретации $\mathcal{I}$ — это максимальная по длине трасса π на σ в $\mathcal{I}$

Результат конечного вычисления программы π
на оценке σ в интерпретации $\mathcal{I}$ — это
оценка в последнем состоянии вычисления π на σ в $\mathcal{I}$

Все бесконечные вычисления считаются не имеющими результата

Операционная семантика программ

Рефлексивно-транзитивное замыкание отношения $R \subseteq X \times X$ — это наименьшее по включению отношение $R^* \subseteq X \times X$, удовлетворяющее следующим свойствам:

- $R \subseteq R^*$
- $\{(x, x) \mid x \in X\} \subseteq R^*$
- Если $(x, y) \in R^*$ и $(y, z) \in R^*$, то $(x, z) \in R^*$

Утверждение. **Для любых программы π, интерпретации $\mathcal{I}$ и состояний данных σ, $\overline{\sigma}$ верно следующее:**

$$\langle \pi \mid \sigma \rangle \xrightarrow{\mathcal{I}}{}^{*} \langle \emptyset \mid \overline{\sigma} \rangle \quad \Leftrightarrow \quad \overline{\sigma} \text{ — результат вычисления } \pi \text{ в } \mathcal{I} \text{ на } \sigma$$

Программой π в интерпретации $\mathcal{I}$ реализуется отношение $R_\pi^{\mathcal{I}} \subseteq \Sigma \times \Sigma$, задающееся так:

$$(\sigma, \overline{\sigma}) \in R_\pi^{\mathcal{I}} \quad \Leftrightarrow \quad \langle \pi \mid \sigma \rangle \xrightarrow{\mathcal{I}}{}^{*} \langle \emptyset \mid \overline{\sigma} \rangle$$

Пример вычисления программы

$\pi = \textbf{while } \neg(\mathrm{x} = \mathrm{y}) \textbf{ do if } \mathrm{x} > \mathrm{y} \textbf{ then } \mathrm{x} := \mathrm{x} - \mathrm{y}; \textbf{ else } \mathrm{y} := \mathrm{y} - \mathrm{x}; \textbf{ fi od}$

Вычисление π на $[\mathrm{x}/2, \mathrm{y}/4]$ в $\mathcal{I}_{ar}$:

$$\langle \pi \mid [\mathrm{x}/2, \mathrm{y}/4] \rangle$$
$$\mathcal{I}_{ar} \downarrow \quad \text{т.к. } \mathcal{I}_{ar} \models \neg(\mathrm{x} = \mathrm{y})[\mathrm{x}/2, \mathrm{y}/4]$$
$$\langle \textbf{if } \mathrm{x} > \mathrm{y} \textbf{ then } \mathrm{x} := \mathrm{x} - \mathrm{y}; \textbf{ else } \mathrm{y} := \mathrm{y} - \mathrm{x}; \textbf{ fi } \pi \mid [\mathrm{x}/2, \mathrm{y}/4] \rangle$$
$$\mathcal{I}_{ar} \downarrow \quad \text{т.к. } \mathcal{I}_{ar} \not\models (\mathrm{x} > \mathrm{y})[\mathrm{x}/2, \mathrm{y}/4]$$
$$\langle \mathrm{y} := \mathrm{y} - \mathrm{x}; \pi \mid [\mathrm{x}/2, \mathrm{y}/4] \rangle$$
$$\mathcal{I}_{ar} \downarrow \quad \text{т.к. } [\mathrm{x}/2, \mathrm{y}/4][\mathrm{y} \leftarrow (\mathrm{y} - \mathrm{x})[\mathrm{x}/2, \mathrm{y}/4]] = [\mathrm{x}/2, \mathrm{y}/2]$$
$$\langle \emptyset\ \pi \mid [\mathrm{x}/2, \mathrm{y}/2] \rangle$$
$$\mathcal{I}_{ar} \downarrow$$
$$\langle \pi \mid [\mathrm{x}/2, \mathrm{y}/2] \rangle$$
$$\mathcal{I}_{ar} \downarrow \quad \text{т.к. } \mathcal{I}_{ar} \not\models \neg(\mathrm{x} = \mathrm{y})[\mathrm{x}/2, \mathrm{y}/2]$$
$$\langle \emptyset \mid [\mathrm{x}/2, \mathrm{y}/2] \rangle$$

Результат этого вычисления: $[\mathrm{x}/2, \mathrm{y}/2]$

Следовательно, $([\mathrm{x}/2, \mathrm{y}/4], [\mathrm{x}/2, \mathrm{y}/2]) \in R_{\pi}^{\mathcal{I}_{ar}}$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 4

Дедуктивная верификация программ:
постановка задачи,
логика Хоара

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Задача дедуктивной верификации программ

Неформальная постановка

Программа частично корректна относительно требований,
выраженных в виде предусловия φ и постусловия ψ,
если для любых входных данных, удовлетворяющих φ,
результат любого конечного вычисления программы удовлетворяет ψ

Программа тотально корректна, если

- она частично корректна и
- любое её вычисление на входных данных,
  удовлетворяющих предусловию φ, конечно

Задача верификации императивных программ
состоит в проверке тотальной или частичной корректности
заданной программы относительно заданных предусловия и постусловия

Задача дедуктивной верификации программ

Формальная постановка

Парой формул логики предикатов φ, ψ в интерпретации $\mathcal{I}$ зададим отношение $R^{\mathcal{I}}_{\varphi,\psi} = \{(\sigma, \overline{\sigma}) \mid \sigma, \overline{\sigma} \in \Sigma,\ \mathcal{I} \models \varphi\sigma,\ \mathcal{I} \models \psi\overline{\sigma}\}$

Для интерпретации $\mathcal{I}$ и формул φ и ψ, называющихся соответственно предусловием и постусловием, говорят, что программа π

- частично корректна в $\mathcal{I}$ относительно φ и ψ, если $R^{\mathcal{I}}_{\pi} \subseteq R^{\mathcal{I}}_{\varphi,\psi}$, и
- тотально корректна в $\mathcal{I}$ относительно φ и ψ, если
  - она частично корректна в $\mathcal{I}$ относительно φ и ψ и
  - вычисление π в $\mathcal{I}$
    на любой оценке σ, для которой верно $\mathcal{I} \models \varphi\sigma$, конечно

Утверждение. **Для любых программы π, интерпретации $\mathcal{I}$, предусловия φ и постусловия ψ верно следующее:**
программа π частично корректна в $\mathcal{I}$ относительно φ и ψ $\Leftrightarrow$
для любых состояний данных σ и $\overline{\sigma}$,
таких что $\mathcal{I} \models \varphi\sigma$ и $\langle \pi \mid \sigma \rangle \xrightarrow{\mathcal{I}}{}^{*} \langle \emptyset \mid \overline{\sigma} \rangle$, верно $\mathcal{I} \models \psi\overline{\sigma}$

Задача дедуктивной верификации программ

Формальная постановка

Триплет Хоара (или, по-другому, тройка Хоара) — это запись вида $\{\varphi\}\pi\{\psi\}$, где

- φ и ψ — формулы логики предикатов, называемые соответственно предусловием и постусловием, и
- π — программа

Триплет Хоара $\{\varphi\}\pi\{\psi\}$ истинен в интерпретации $\mathcal{I}$ ($\mathcal{I} \models \{\varphi\}\pi\{\psi\}$), если программа π частично корректна в $\mathcal{I}$ относительно φ и ψ

Логика Хоара

Для доказательства частичной корректности программ будем использовать систему правил[1] вида:

$$\frac{\Phi}{\Psi_1}, \quad \frac{\Phi}{\varphi}, \quad \frac{\Phi}{\Psi_1, \Psi_2}, \quad \frac{\Phi}{\varphi, \Psi_1, \psi}$$

(φ, ψ — формулы логики предикатов;
Φ, Ψ_1, Ψ_2 — триплеты Хоара)

Содержательно каждое из правил прочитывается так:

если истинны все триплеты и формулы, записанные под чертой,
то триплет Φ истинен

Правила можно прочитать и немного по-другому:

если **доказана** истинность всех триплетов и формул под чертой,
то **доказана** и истинность триплета Φ

1 Hoare C.A.R. An axiomatic basis for computer programming. 1969

Логика Хоара

Вот эти правила:

$$R_{\emptyset}\colon \frac{\{\varphi\}\emptyset\{\varphi\}}{\Uparrow}$$

$$R_{:=}\colon \frac{\{\varphi\{x/t\}\}\mathrm{x} := t;\{\varphi\}}{\Uparrow}$$

(*если подстановка* $\{\mathrm{x}/t\}$ *правильна для* φ)

$$R_{inf}\colon \frac{\{\varphi\}\pi\{\psi\}}{\varphi \to \varphi', \{\varphi'\}\pi\{\psi'\}, \psi' \to \psi}$$

$$R_{seq}\colon \frac{\{\varphi\}\pi_1\ \pi_2\{\psi\}}{\{\varphi\}\pi_1\{\chi\}, \{\chi\}\pi_2\{\psi\}}$$

$$R_{\textbf{if}}\colon \frac{\{\varphi\}\textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi}\{\psi\}}{\{\varphi \,\&\, C\}\pi_1\{\psi\}, \{\varphi \,\&\, \neg C\}\pi_2\{\psi\}}$$

$$R_{\textbf{while}}\colon \frac{\{\varphi\}\textbf{while } C \textbf{ do } \pi \textbf{ od}\{\varphi \,\&\, \neg C\}}{\{\varphi \,\&\, C\}\pi\{\varphi\}}$$

Подстановка $\{\mathrm{x}/t\}$ правильна для формулы φ, если ни одно свободное вхождение x не входит в область действия квантора, связывающего какую-либо переменную терма t

Формула φ в правиле $R_{\textbf{while}}$ называется инвариантом цикла

Логика Хоара

Теорема (о корректности правил вывода логики Хоара)

Для любой интерпретации $\mathcal{I}$

и любого из правил $R_{\emptyset}$, $R_{:=}$, R_{inf}, R_{seq}, $R_{\textbf{if}}$, $R_{\textbf{while}}$

$$\left(\frac{\Phi}{\Psi_1}, \quad \frac{\Phi}{\varphi}, \quad \frac{\Phi}{\Psi_1, \Psi_2}, \quad \frac{\Phi}{\varphi, \Psi_1, \psi}\right)$$

верно: если $\mathcal{I} \models \Psi_1$, $\mathcal{I} \models \Psi_2$, $\mathcal{I} \models \varphi$ **и** $\mathcal{I} \models \psi$, **то** $\mathcal{I} \models \Phi$

Доказательство.

Подробно рассмотрим только правило $R_{:=}$: $\dfrac{\{\varphi\{x/t\}\}\mathtt{x} := t;\{\varphi\}}{\mathbb{t}}$

Рассмотрим произвольное состояние данных σ, такое что $\mathcal{I} \models \varphi\{x/t\}\sigma$, и соотношение $\langle \mathtt{x} := t; \mid \sigma\rangle \xrightarrow{\mathcal{I}} \langle \pi \mid \overline{\sigma}\rangle$

По операционной семантике программ, верно $\pi = \emptyset$ и $\overline{\sigma} = \sigma[\mathtt{x} \leftarrow t\sigma]$

Следовательно, верно $\mathcal{I} \models \varphi\overline{\sigma}$ и $\mathcal{I} \models \{\varphi\{x/t\}\}\mathtt{x} := t;\{\varphi\}$

Корректность остальных правил доказывается аналогично ▼

Логика Хоара

Зачем нужна теорема о корректности[1]

Истинность триплета $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$ доказана,
если построен успешный вывод этого триплета,
то есть конечный вывод вида

$$\dfrac{\{\varphi\}\pi\{\psi\}}{\begin{array}{ccccccc}\ldots & \dfrac{\ldots}{\chi} & \ldots & \dfrac{\ldots}{\chi' \quad \dfrac{\{\varphi'\}\pi'\{\psi'\}}{\ldots} \quad \chi''} & \ldots\end{array}},$$

где

- под каждым триплетом Хоара в выводе стоит черта
- под каждой чертой записаны формулы и триплеты Хоара согласно правилам
- каждая формула, располагающаяся в выводе вне триплетов (как χ, χ', χ'' на рисунке), истинна в интерпретации $\mathcal{I}$

1 Подробнее о выводе в логике Хоара рассказывается в курсе «Математическая логика и логическое программирование» бакалавриата

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 5

Дедуктивная верификация программ:
аннотированные программы

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Аннотированные программы: определения

Доказательство корректности программ, основанное логике Хоара, можно сделать более наглядным, если “встроить” информацию о применении правил вывода непосредственно в текст программы

Аннотация — это запись вида $\{\varphi\}$, где φ — произвольная формула

Аннотированная программа — это программа, в которой до и после каждой команды могут располагаться последовательности аннотаций

Аннотация может расцениваться как

- предусловие команды, следующей за аннотацией
- постусловие команды, предшествующей аннотации
- составная часть триплета, использующегося в выводе исходного триплета

Аннотированные программы: определения

Аннотированная программа $\widetilde{\pi}$ правильно отвечает триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, если выполняются следующие условия:

1. При удалении всех аннотаций из $\widetilde{\pi}$ получается программа π
2. $\widetilde{\pi}$ начинается с аннотации $\{\varphi\}$ и оканчивается аннотацией $\{\psi\}$
3. Перед каждой командой и после каждой команды в $\widetilde{\pi}$ располагается хотя бы одна аннотация
4. Аннотации перед каждой пустой командой и после неё равны:
$$\{\chi\}\emptyset\{\chi\}$$
5. Аннотации перед каждым присваиванием и после него соотносятся так же, как и в правиле $R_{:=}$:
$$\{\chi\{\mathrm{x}/t\}\}\mathrm{x} := t;\{\chi\},$$
где подстановка $\{\mathrm{x}/t\}$ правильна для формулы χ

Аннотированные программы: определения

Аннотированная программа $\widetilde{\pi}$ правильно отвечает триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, если выполняются следующие условия:

6. Каждое ветвление в $\widetilde{\pi}$ аннотировано следующим образом:
$$\{\chi_1\}$$
$$\textbf{if } C \textbf{ then } \{\chi_1 \,\&\, C\}\widetilde{\pi}_1\{\chi_2\} \textbf{ else } \{\chi_1 \,\&\, \neg C\}\widetilde{\pi}_2\{\chi_2\} \textbf{ fi}$$
$$\{\chi_2\}$$

7. каждый цикл в $\widetilde{\pi}$ аннотирован следующим образом:
$$\{\chi\}$$
$$\textbf{while } C \textbf{ do } \{\chi \,\&\, C\}\widetilde{\pi}'\{\chi\} \textbf{ od}$$
$$\{\chi \,\&\, \neg C\}$$

8. Для любых двух подряд идущих аннотаций $\{\chi_1\}\{\chi_2\}$ в $\widetilde{\pi}$ верно $\mathcal{I} \models \chi_1 \to \chi_2$

Аннотированные программы: пример

Рассмотрим такую программу π:

$$\textbf{while } \mathrm{x} \neq \mathrm{y} \textbf{ do if } \mathrm{x} > \mathrm{y} \textbf{ then } \mathrm{x} := \mathrm{x} - \mathrm{y}; \textbf{ else } \mathrm{y} := \mathrm{y} - \mathrm{x}; \textbf{ fi od}$$

Докажем, что программа π корректно реализует вычисление наибольшего общего делителя натуральных значений x и y с записью результата в x

Для этого достаточно доказать истинность триплета $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}_{ar}$, где:

$$\begin{aligned}
\varphi(\mathrm{x},\mathrm{y},\mathrm{z}): &\quad \mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \\
\psi(\mathrm{x},\mathrm{y},\mathrm{z}): &\quad \mathrm{x} = \mathrm{z} \\
\gcd(\mathrm{x},\mathrm{y},\mathrm{z}): &\quad \exists \mathrm{u}\,(\mathrm{z} \times \mathrm{u} = \mathrm{x}) \,\&\, \exists \mathrm{u}\,(\mathrm{z} \times \mathrm{u} = \mathrm{y}) \,\& \\
&\quad \forall \mathrm{w}\,(\exists \mathrm{u}\,(\mathrm{w} \times \mathrm{u} = \mathrm{x}) \,\&\, \exists \mathrm{u}\,(\mathrm{w} \times \mathrm{u} = \mathrm{y}) \to (\mathrm{w} \leq \mathrm{z}))
\end{aligned}$$

Аннотированные программы: пример

Аннотированная программа, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$, может выглядеть так:

Аннотированные программы: пример

Аннотированная программа, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$, может выглядеть так:

$\{\mathrm{x} > \mathbf{0}\,\&\,\mathrm{y} > \mathbf{0}\,\&\,\gcd(\mathrm{x},\mathrm{y},\mathrm{z})\}$
while $\mathrm{x} \neq \mathrm{y}$ **do**

if $\mathrm{x} > \mathrm{y}$ **then**

$\mathrm{x} := \mathrm{x} - \mathrm{y};$

else

$\mathrm{y} := \mathrm{y} - \mathrm{x};$

fi

od

$\{\mathrm{x} = \mathrm{z}\}$

Аннотированные программы: пример

Аннотированная программа, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$, может выглядеть так:

$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z})\}$
while $\mathrm{x} \neq \mathrm{y}$ **do**
$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y}\}$
if $\mathrm{x} > \mathrm{y}$ **then**

$\mathrm{x} := \mathrm{x} - \mathrm{y};$

else

$\mathrm{y} := \mathrm{y} - \mathrm{x};$

fi
$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z})\}$
od
$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z}) \,\&\, \neg(\mathrm{x} \neq \mathrm{y})\}$
$\{\mathrm{x} = \mathrm{z}\}$

Аннотированные программы: пример

Аннотированная программа, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$, может выглядеть так:

$$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z})\}$$
while $\mathrm{x} \neq \mathrm{y}$ **do**
$$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y}\}$$
if $\mathrm{x} > \mathrm{y}$ **then**
$$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y} \,\&\, \mathrm{x} > \mathrm{y}\}$$
$$\mathrm{x} := \mathrm{x} - \mathrm{y};$$
else
$$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y} \,\&\, \neg(\mathrm{x} > \mathrm{y})\}$$
$$\mathrm{y} := \mathrm{y} - \mathrm{x};$$
fi
$$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z})\}$$
od
$$\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x}, \mathrm{y}, \mathrm{z}) \,\&\, \neg(\mathrm{x} \neq \mathrm{y})\}$$
$$\{\mathrm{x} = \mathrm{z}\}$$

Аннотированные программы: пример

Аннотированная программа, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$, может выглядеть так:

$$
\begin{array}{l}
\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad \textbf{while } \mathrm{x} \neq \mathrm{y} \textbf{ do} \\
\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y}\} \\
\quad\quad \textbf{if } \mathrm{x} > \mathrm{y} \textbf{ then} \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y} \,\&\, \mathrm{x} > \mathrm{y}\} \\
\\
\quad\quad\quad \mathrm{x} := \mathrm{x} - \mathrm{y}; \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad\quad \textbf{else} \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y} \,\&\, \neg(\mathrm{x} > \mathrm{y})\} \\
\\
\quad\quad\quad \mathrm{y} := \mathrm{y} - \mathrm{x}; \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad\quad \textbf{fi} \\
\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad \textbf{od} \\
\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \neg(\mathrm{x} \neq \mathrm{y})\} \\
\{\mathrm{x} = \mathrm{z}\}
\end{array}
$$

Аннотированные программы: пример

Аннотированная программа, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$, может выглядеть так:

$$
\begin{array}{l}
\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad \textbf{while } \mathrm{x} \neq \mathrm{y} \textbf{ do} \\
\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y}\} \\
\quad\quad \textbf{if } \mathrm{x} > \mathrm{y} \textbf{ then} \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y} \,\&\, \mathrm{x} > \mathrm{y}\} \\
\quad\quad\quad \{\mathrm{x} - \mathrm{y} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x} - \mathrm{y},\mathrm{y},\mathrm{z})\} \\
\quad\quad\quad\quad \mathrm{x} := \mathrm{x} - \mathrm{y}; \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad\quad \textbf{else} \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y} \,\&\, \neg(\mathrm{x} > \mathrm{y})\} \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} - \mathrm{x} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y} - \mathrm{x},\mathrm{z})\} \\
\quad\quad\quad\quad \mathrm{y} := \mathrm{y} - \mathrm{x}; \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad\quad \textbf{fi} \\
\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad \textbf{od} \\
\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \neg(\mathrm{x} \neq \mathrm{y})\} \\
\{\mathrm{x} = \mathrm{z}\}
\end{array}
$$

Аннотированные программы: пример

Аннотированная программа, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$, может выглядеть так:

$$
\begin{array}{l}
\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad \textbf{while } \mathrm{x} \neq \mathrm{y} \textbf{ do} \\
\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y}\} \\
\quad\quad \textbf{if } \mathrm{x} > \mathrm{y} \textbf{ then} \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y} \,\&\, \mathrm{x} > \mathrm{y}\} \\
\quad\quad\quad \{\mathrm{x} - \mathrm{y} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x} - \mathrm{y},\mathrm{y},\mathrm{z})\} \\
\quad\quad\quad\ \mathrm{x} := \mathrm{x} - \mathrm{y}; \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad\quad \textbf{else} \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \mathrm{x} \neq \mathrm{y} \,\&\, \neg(\mathrm{x} > \mathrm{y})\} \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} - \mathrm{x} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y} - \mathrm{x},\mathrm{z})\} \\
\quad\quad\quad\ \mathrm{y} := \mathrm{y} - \mathrm{x}; \\
\quad\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad\quad \textbf{fi} \\
\quad\quad \{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z})\} \\
\quad \textbf{od} \\
\{\mathrm{x} > \mathbf{0} \,\&\, \mathrm{y} > \mathbf{0} \,\&\, \gcd(\mathrm{x},\mathrm{y},\mathrm{z}) \,\&\, \neg(\mathrm{x} \neq \mathrm{y})\} \\
\{\mathrm{x} = \mathrm{z}\}
\end{array}
$$

Аннотированные программы и логика Хоара

Теорема (о корректности аннотации программ)
Если существует аннотированная программа $\widetilde{\pi}$, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, то $\mathcal{I} \models \{\varphi\}\pi\{\psi\}$

Доказательство.

Достаточно показать, как по аннотированной программе $\widetilde{\pi}$ можно построить успешный вывод триплета $\{\varphi\}\pi\{\psi\}$:

Фрагмент программы $\widetilde{\pi}$ вида

$$\{\chi\}\emptyset\{\chi\}$$

соответствует фрагменту вывода

$$R_{\emptyset}\colon \frac{\{\chi\}\emptyset\{\chi\}}{\mathbb{t}}$$

Аннотированные программы и логика Хоара

Теорема (о корректности аннотации программ)
Если существует аннотированная программа $\widetilde{\pi}$, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, то $\mathcal{I} \models \{\varphi\}\pi\{\psi\}$

Доказательство.

Достаточно показать, как по аннотированной программе $\widetilde{\pi}$ можно построить успешный вывод триплета $\{\varphi\}\pi\{\psi\}$:

Фрагмент программы $\widetilde{\pi}$ вида

$$\{\chi\{x/t\}\}\mathtt{x} := t;\{\chi\}$$

соответствует фрагменту вывода

$$R_{:=}\colon \frac{\{\chi\{x/t\}\}\mathtt{x} := t;\{\chi\}}{\mathbb{t}}$$

Аннотированные программы и логика Хоара

Теорема (о корректности аннотации программ)
Если существует аннотированная программа $\widetilde{\pi}$, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, то $\mathcal{I} \models \{\varphi\}\pi\{\psi\}$

Доказательство.

Достаточно показать, как по аннотированной программе $\widetilde{\pi}$ можно построить успешный вывод триплета $\{\varphi\}\pi\{\psi\}$:

Фрагмент программы $\widetilde{\pi}$ вида

$$\{\chi\}\{\chi'\}\widetilde{\pi}'\{\chi''\}$$

соответствует фрагменту вывода

$$R_{inf}\colon \frac{\{\chi\}\pi'\{\chi''\}}{\chi \to \chi', \{\chi'\}\pi'\{\chi''\}, \chi'' \to \chi''}$$

Аннотированные программы и логика Хоара

Теорема (о корректности аннотации программ)
Если существует аннотированная программа $\widetilde{\pi}$, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, то $\mathcal{I} \models \{\varphi\}\pi\{\psi\}$

Доказательство.

Достаточно показать, как по аннотированной программе $\widetilde{\pi}$ можно построить успешный вывод триплета $\{\varphi\}\pi\{\psi\}$:

Фрагмент программы $\widetilde{\pi}$ вида

$$\{\chi\}\widetilde{\pi}'\{\chi'\}\{\chi''\}$$

соответствует фрагменту вывода

$$R_{inf}\colon \frac{\{\chi\}\pi'\{\chi''\}}{\chi \to \chi, \{\chi\}\pi'\{\chi'\}, \chi' \to \chi''}$$

Аннотированные программы и логика Хоара

Теорема (о корректности аннотации программ)
Если существует аннотированная программа $\widetilde{\pi}$, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, то $\mathcal{I} \models \{\varphi\}\pi\{\psi\}$

Доказательство.

Достаточно показать, как по аннотированной программе $\widetilde{\pi}$ можно построить успешный вывод триплета $\{\varphi\}\pi\{\psi\}$:

Фрагмент программы $\widetilde{\pi}$ вида

$$\{\chi\}\widetilde{\pi}_1\{\chi'\}\ \widetilde{\pi}_2\{\chi''\}$$

соответствует фрагменту вывода

$$R_{seq}\colon \frac{\{\chi\}\pi_1\ \pi_2\{\chi''\}}{\{\chi\}\pi_1\{\chi'\}, \{\chi'\}\pi_2\{\chi''\}}$$

Аннотированные программы и логика Хоара

Теорема (о корректности аннотации программ)
Если существует аннотированная программа $\widetilde{\pi}$, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, то $\mathcal{I} \models \{\varphi\}\pi\{\psi\}$

Доказательство.

Достаточно показать, как по аннотированной программе $\widetilde{\pi}$ можно построить успешный вывод триплета $\{\varphi\}\pi\{\psi\}$:

Фрагмент программы $\widetilde{\pi}$ вида

$$\{\chi\}\textbf{if } C \textbf{ then } \{\chi \,\&\, C\}\widetilde{\pi}_1\{\chi'\} \textbf{ else } \{\chi \,\&\, \neg C\}\widetilde{\pi}_2\{\chi'\} \textbf{ fi}\{\chi'\}$$

соответствует фрагменту вывода

$$R_{\textbf{if}}\colon \frac{\{\chi\}\textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi}\{\chi'\}}{\{\chi \,\&\, C\}\pi_1\{\chi'\}, \{\chi \,\&\, \neg C\}\pi_2\{\chi'\}}$$

Аннотированные программы и логика Хоара

Теорема (о корректности аннотации программ)
Если существует аннотированная программа $\widetilde{\pi}$, правильно отвечающая триплету $\{\varphi\}\pi\{\psi\}$ в интерпретации $\mathcal{I}$, то $\mathcal{I} \models \{\varphi\}\pi\{\psi\}$

Доказательство.

Достаточно показать, как по аннотированной программе $\widetilde{\pi}$ можно построить успешный вывод триплета $\{\varphi\}\pi\{\psi\}$:

Фрагмент программы $\widetilde{\pi}$ вида

$$\{\chi\}\textbf{while } C \textbf{ do } \{\chi \,\&\, C\}\widetilde{\pi}'\{\chi\} \textbf{ od}\{\chi \,\&\, \neg C\}$$

соответствует фрагменту вывода

$$R_{\textbf{while}}\colon \frac{\{\chi\}\textbf{while } C \textbf{ do } \pi' \textbf{ od}\{\chi \,\&\, \neg C\}}{\{\chi \,\&\, C\}\pi'\{\chi\}}$$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 6

Дедуктивная верификация программ:
возможности автоматизации,
слабейшее предусловие

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Насколько просто **автоматизировать** проверку корректности программ?

Иллюстрация, показывающая, насколько это непростая задача:
доказательство истинности триплета

$$\{\mathtt{x} > \mathbf{0} \,\&\, \mathtt{y} > \mathbf{0} \,\&\, \mathtt{z} > \mathbf{0} \,\&\, \mathtt{n} > \mathbf{2}\}$$
$$\textbf{if } \mathtt{x}^{\mathtt{n}} + \mathtt{y}^{\mathtt{n}} = \mathtt{z}^{\mathtt{n}} \textbf{ then } \mathtt{u} := \mathbf{0}; \textbf{ else } \mathtt{u} := \mathbf{1}; \textbf{ fi}$$
$$\{\mathtt{u} > \mathbf{0}\}$$

в $\mathcal{I}_{ar}$ — это доказательство Великой теоремы Ферма

Более того, проблема корректности программ в общем случае и даже в ряде достаточно простых случаев **неразрешима**:

- программы с операцией $+$ и отношением $<$ в интерпретации $\mathcal{I}_{ar}$ полны по Тьюрингу
- любое нетривиальное семантическое свойство частично-рекурсивных функций, а значит, и машин Тьюринга, неразрешимо (*Rice. Classes of recursively enumerable sets and their decision problems. 1953*)

Слабейшее предусловие (weakest precondition)
для программы π и постусловия ψ в интерпретации $\mathcal{I}$ — это формула $wpr(\pi, \psi, \mathcal{I})$, для которой выполнены следующие условия:

1. $\mathcal{I} \models \{wpr(\pi, \psi, \mathcal{I})\}\pi\{\psi\}$
2. Для любой формулы χ, такой что $\mathcal{I} \models \{\chi\}\pi\{\psi\}$, верно $\mathcal{I} \models \chi \to wpr(\pi, \psi, \mathcal{I})$

Теорема. $\mathcal{I} \models \{\varphi\}\pi\{\psi\} \quad \Leftrightarrow \quad \mathcal{I} \models \varphi \to wpr(\pi, \psi, \mathcal{I})$

Доказательство. Напрямую следует из определений слабейшего предусловия и истинности триплета ▼

Будем писать $wpr(\pi, \psi)$ вместо $wpr(\pi, \psi, \mathcal{I})$, если известно, что слабейшее предусловие не зависит существенно от интерпретации $\mathcal{I}$.

Для полной автоматизации проверки корректности программ достаточно иметь два алгоритма:

1. Алгоритм проверки истинности формул логики предикатов
   - Этот алгоритм зависит от выбора интерпретации программ и в общем случае может и не существовать
2. Алгоритм вычисления слабейшего предусловия для произвольных программ и постусловий
   - С этим алгоритмом дела обстоят немного лучше:

Теорема.

- $wpr(\mathtt{x} := t; , \psi) = \psi\{x/t\}$
- $wpr(\pi_1\ \pi_2, \psi) = wpr(\pi_1, wpr(\pi_2, \psi))$
- $wpr(\textbf{if}\ C\ \textbf{then}\ \pi_1\ \textbf{else}\ \pi_2\ \textbf{fi}, \psi) =$
  $$C\ \&\ wpr(\pi_1, \psi) \vee \neg C\ \&\ wpr(\pi_2, \psi)$$

Доказательство. Самостоятельно

То есть проверка правильности императивных программ без циклов настолько же трудна, насколько и проверка истинности формул

$$R_{\textbf{while}}\colon \frac{\{\varphi\}\textbf{while } C \textbf{ do } \pi \textbf{ od}\{\varphi \,\&\, \neg C\}}{\{\varphi \,\&\, C\}\pi\{\varphi\}}$$

Для применения правила $R_{\textbf{while}}$ (а также для аннотации цикла, и для вычисления слабейшего предусловия цикла) необходимо предварительно найти подходящую формулу φ —

инвариант цикла

Автоматическая генерация инвариантов циклов, как правило, является **основной проблемой** автоматизации проверки правильности программ

Эта проблема осложняется тем, что

1. неочевидно даже существование свойства данных, выражаемого инвариантом цикла
2. если такое свойство существует, то никто не гарантирует, что это свойство можно записать в виде формулы в используемой сигнатуре

Математические методы верификации схем и программ

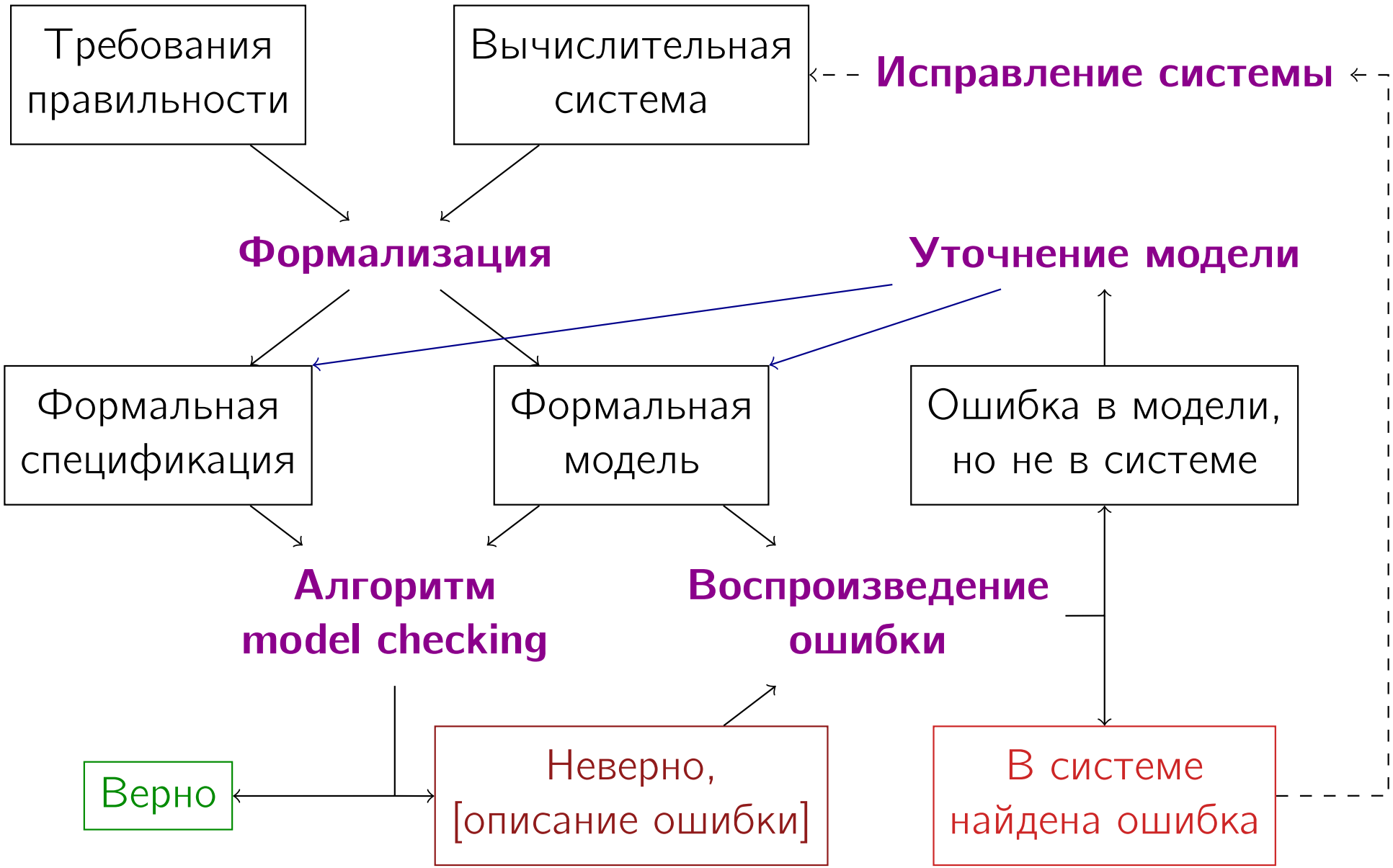
mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

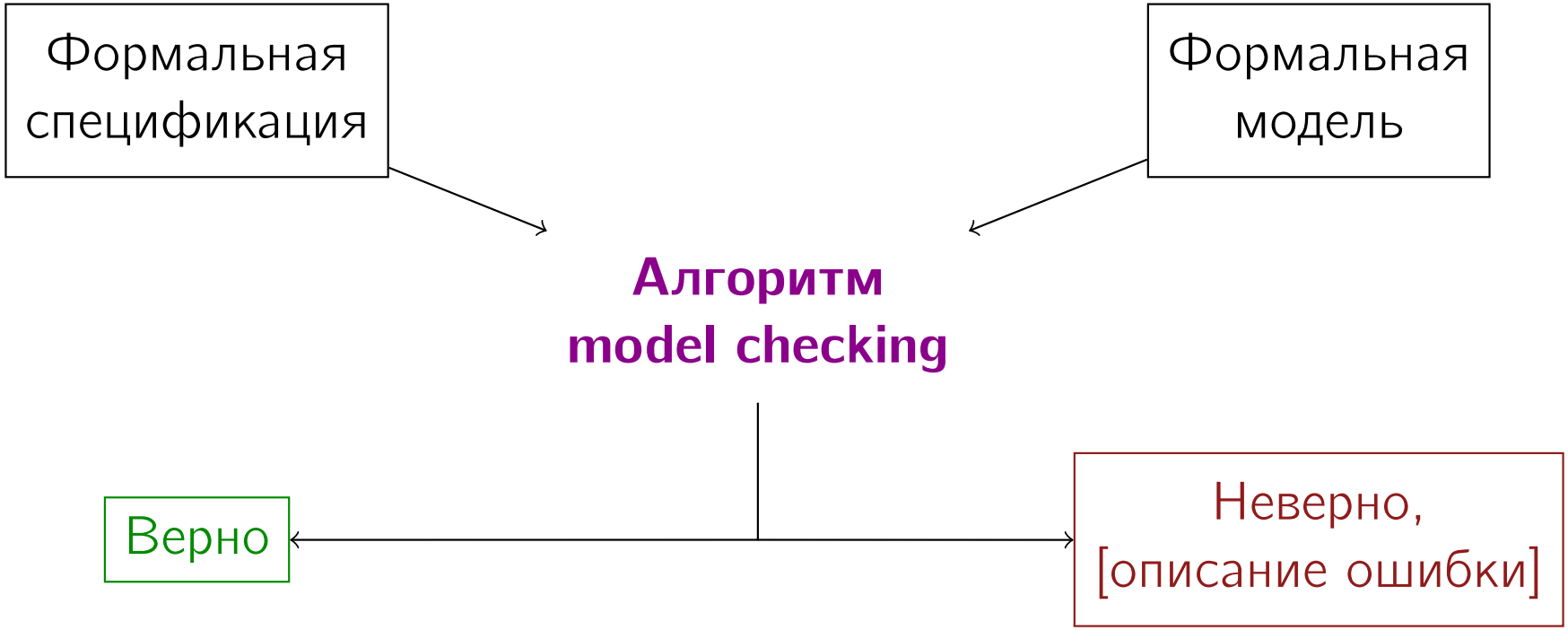
Блок 7

Общая схема метода model checking

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

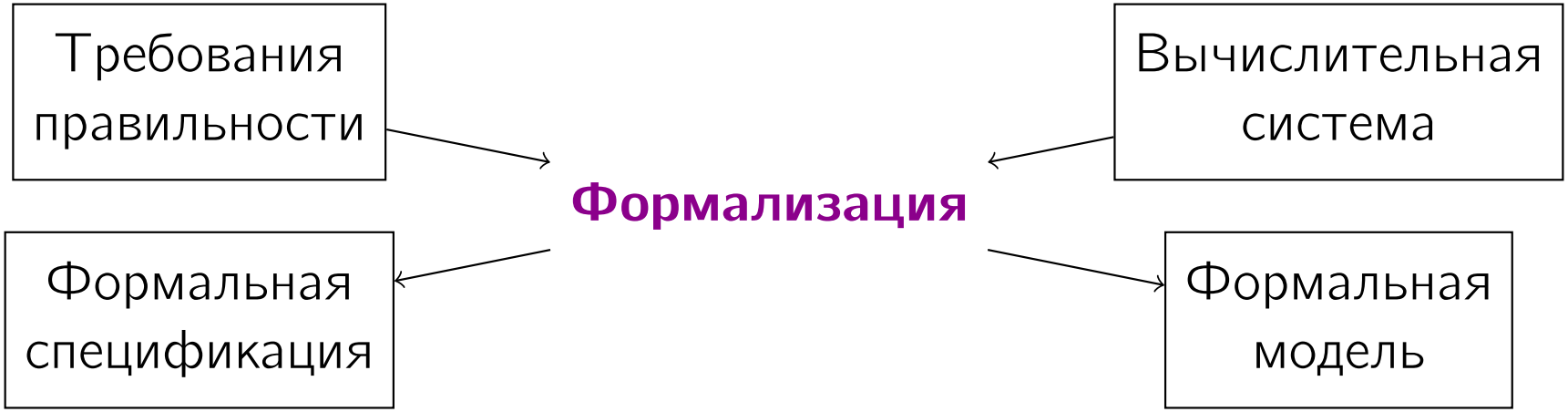
ВМК МГУ, 2023/2024, осенний семестр


Требования правильности
Вычислительная система
Исправление системы
Формализация
Уточнение модели
Формальная спецификация
Формальная модель
Ошибка в модели, но не в системе
Алгоритм model checking
Воспроизведение ошибки
Верно
Неверно, [описание ошибки]
В системе найдена ошибка



Проверка того, удовлетворяет ли модель системы её спецификации, выполняется **автоматически** и *в той или иной степени* **эффективно**

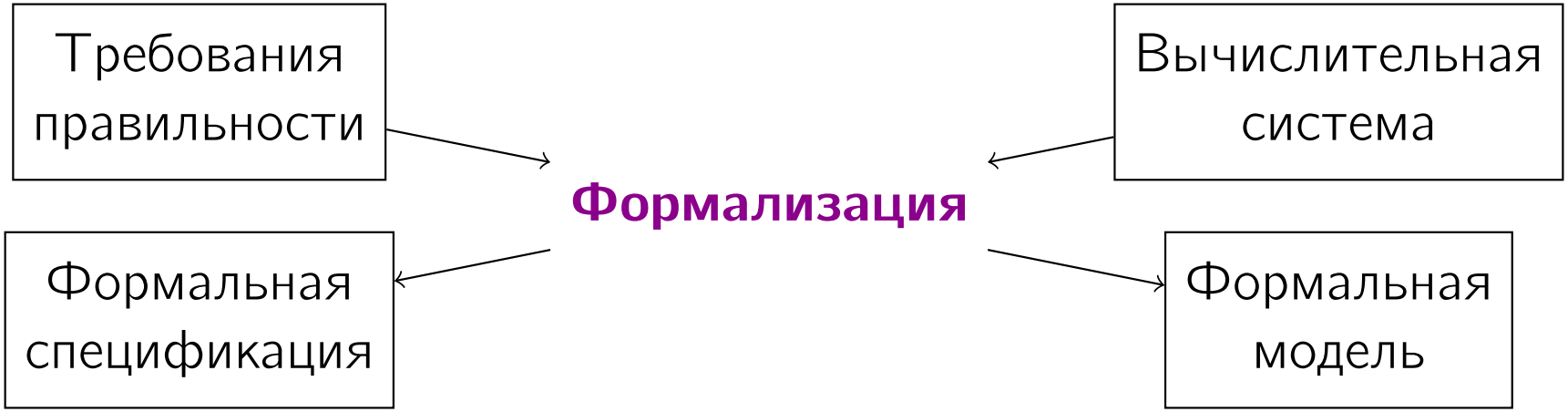
Это основное достоинство метода
и то, почему его предпочтительно применять там, где это возможно



Формализация системы и требований — это непростой ручной труд и основное препятствие к широкому использованию метода

На этом этапе:

- Определяется то, какие именно свойства системы достаточно проверить, чтобы признать систему правильной
- По системе и её свойствам (требованиям к ней) составляются модель системы и формальная спецификация



Модель системы должна быть

- настолько детальна и адекватна, чтобы результаты анализа можно было перенести на исходную систему, и при этом
- настолько проста и «очищена» от излишних деталей, чтобы можно было эффективно её анализировать

Результаты верификации модели настолько достоверны применительно к исходной системе, насколько хорошо построена модель

Плохая модель — бесполезные результаты

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 8

Модели Крипке

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Обычно модель в рамках метода model checking устроена так:

- Моделью задаётся множество состояний: «слепков» системы, в которых записаны рассматриваемые особенности системы в заданные моменты времени выполнения
- Состояния могут изменяться посредством выполнения переходов, изменяющих текущее состояние согласно выполнению заданных действий системой
- Выполнение системы в неограниченном времени соответствует вычислению модели: бесконечной последовательности состояний, получающейся из заданного состояния выполнением переходов

Вступление

Model checking применяется *в основном* для анализа систем с **конечным** числом состояний

Это один из недостатков метода, затрудняющих его широкое использование: на практике число состояний системы нередко бесконечно или конечно, но настолько велико, что можно считать его практически бесконечным

Тем не менее, существуют и важные классы систем, заведомо обладающие «разумно»-конечным числом состояний: контроллеры, драйверы, многие коммуникационные протоколы, не слишком объёмная аппаратура, ...

Вступление

Обсуждение моделей вычислительных систем начнём немного издалека, с классической головоломки про волка, козу и капусту

На левом берегу реки располагаются
волк (), коза (), капуста () и лодочник с лодкой ()

 может переправиться на противоположный берег в лодке,
взяв с собой не более одного пассажира ( ,  , )

Оставшись на берегу без  ,
 может съесть  , а  может съесть 

Как  может безопасно переправить  ,  и  на правый берег?

Вступление

Чтобы решить эту головоломку, достаточно

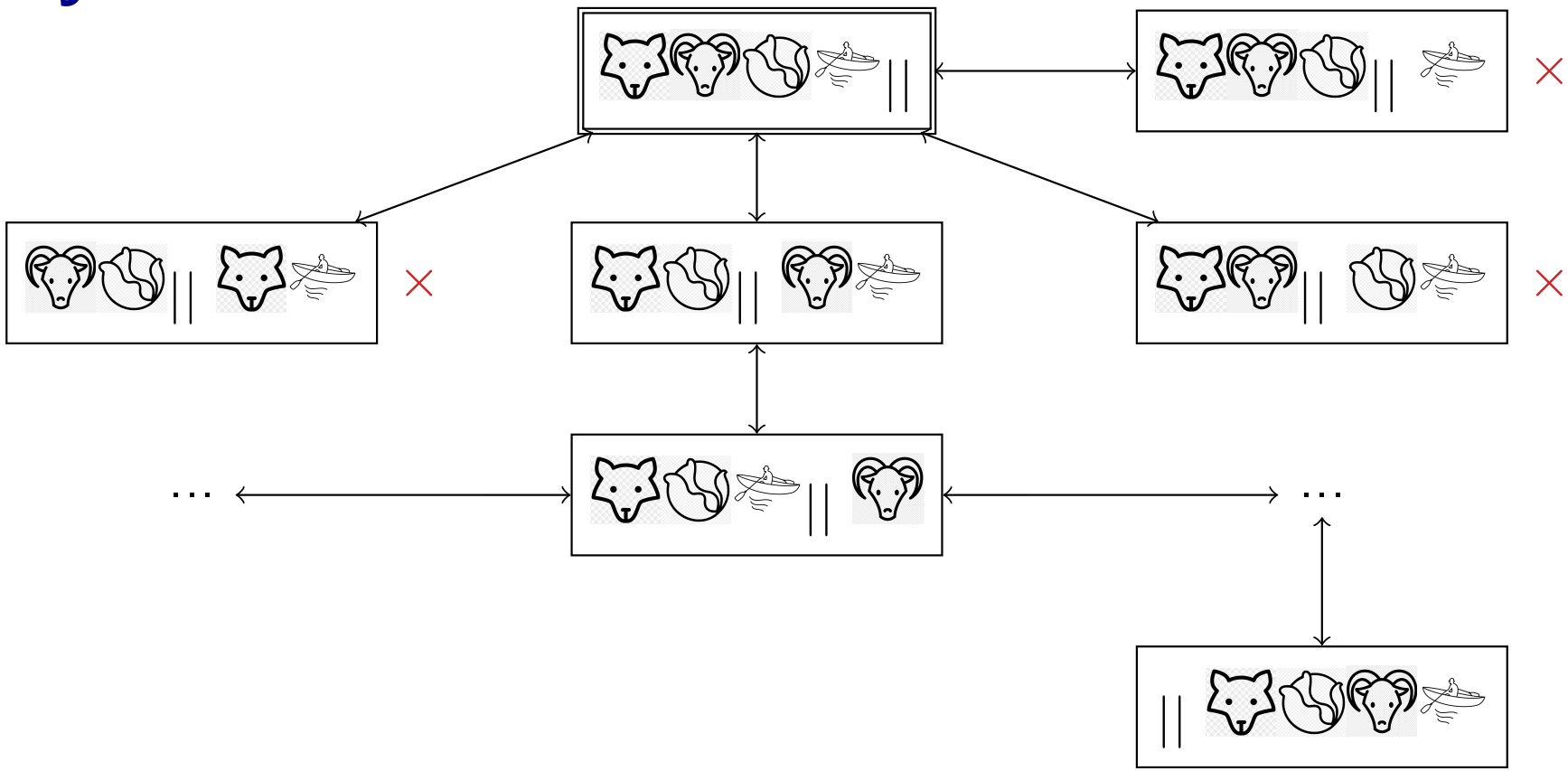
- перебрать всевзоможные варианты расположения  ,  ,  и  , получающиеся из начального расположения

  согласно всевозможным действиям 
  - это **состояния** системы

- разделить состояния на “плохие” (кто-то кого-то может съесть) и “хорошие” (остальные)
  - пометим плохие состояния символом $\times$

- посмотреть, как достичь состояния “все на правом берегу”, ни разу не встретив $\times$

Вступление



□ — **состояния** системы

▣ — **начальное** состояние

$\rightarrow$ — **переходы** системы

$\times$ — **атомарное высказывание**: свойство состояний системы, которое мы по тем или иным причинам посчитали заслуживающим рассмотрения

Модели Крипке

Для множества X записью 2^X будем обозначать множество всех подмножеств X

Модель Крипке над множеством атомарных высказываний AP — это система $M = (S, S_0, \to, L)$, где:

- S — множество состояний
- S_0 — множество начальных состояний, $S_0 \subseteq S$
- $\to \subseteq S \times S$ — тотальное отношение переходов
- $L : S \to 2^{\mathrm{AP}}$ — функция разметки

Тотальность отношения переходов означает, что для любого состояния s существует состояние s', такое что $s \to s'$

Событием будем называть произвольное множество атомарных высказываний (элемент семейства 2^{AP})

Модели Крипке

$(M = (S, S_0, \to, L)$ — *модель Крипке над* AP$)$

Соотношение $L(s) = \sigma$ можно понимать так: состояние s
обладает свойствами, отвечающими атомарным высказываниям из σ,
и не обладает остальными свойствами,
отвечающими атомарным высказываниям

Будем говорить, что модель M конечна,
если конечны множества S и AP

Модель M представляет собой особый размеченный ориентированный граф: S — это вершины, $\to$ — это дуги, остальное — это метки вершин

В связи с этим будем применять графовые обозначения
и графовую терминологию к моделям Крипке

Модели Крипке

$(M = (S, S_0, \to, L)$ — *модель Крипке над* AP$)$

Путь в модели Крипке, исходящий из начального состояния, будем называть начальным

Бесконечный начальный путь будем наывать вычислением модели

Трассой будем называть бесконечную последовательность событий

Иногда будут представлять интерес и конечные последовательности событий — будем называть их конечными трассами, или просто трассами, если конечность следует из контекста

Трассой пути $s_1 \to s_2 \to s_3 \to \ldots$ в модели Крипке будем называть трассу, состоящую из событий, помечающих состояния этого пути:

$$L(s_1), L(s_2), L(s_3), \ldots$$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 9

Особенности моделирования систем

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Моделирование программ

Рассмотрим императивную программу π, выполняющуюся в интерпретации $\mathcal{I}$ на произвольном состоянии данных множества X

Модель $M_{\pi,\mathcal{I},X} = (S, S_0, \to, L)$, отвечающая такому выполнению, может быть устроена так:

- S — это множество всех состояний вычисления программы
- $S_0 = \{\langle \pi \mid \sigma \rangle \mid \sigma \in X\}$
- $\to$ — отношение переходов программы, в которое добавлены всевозможные пары вида $\langle \emptyset \mid \sigma \rangle \to \langle \emptyset \mid \sigma \rangle$
- AP — всевозможные связки x/d оценок переменных программы
- $\mathrm{x}/d \in L(\langle \pi' \mid \sigma \rangle) \Leftrightarrow \sigma(\mathrm{x}) = d$

Путями в такой модели Крипке являются трассы программы в $\mathcal{I}$ на σ, $\sigma \in X$

Для других видов программ модель можно устроить точно так же, если определена операционная семантика

Моделирование программ

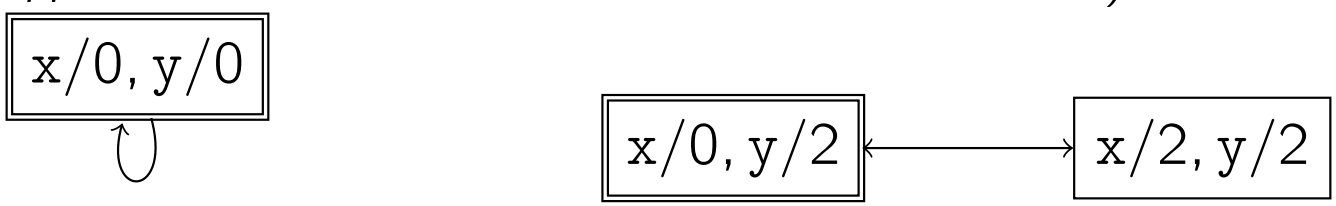
Для примера рассмотрим программу, в которой
бесконечно (в цикле) выполняется присваивание

$$\mathtt{x} := \mathtt{x} + \mathtt{y};,$$

в модели императивных программ с интерпретацией, задающей арифметику двухразрядных чисел с переполнением

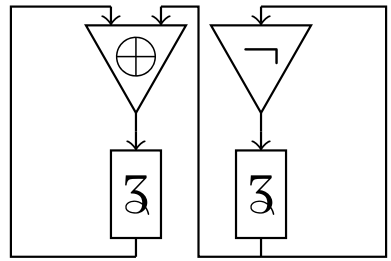
Пусть известно, что в начале работы программы $\mathtt{x} = 0$,
а значение $\mathtt{y}$ может быть любым

Тогда некоторые компоненты связности
соответствующей модели Крипке устроены так
(*можете представить себе и остальные по аналогии*):



Моделирование схем

(*кто знает термин «последовательная схема» — можете представить её вместо схемы из функциональных элементов с задержкой, СФЭЗ*)

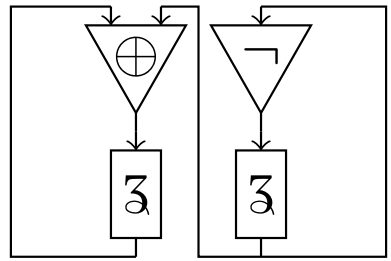


Модель $M = (S, S_0, \to, L)$, отвечающая СФЭЗ (*последовательной схеме*), может быть устроена так:

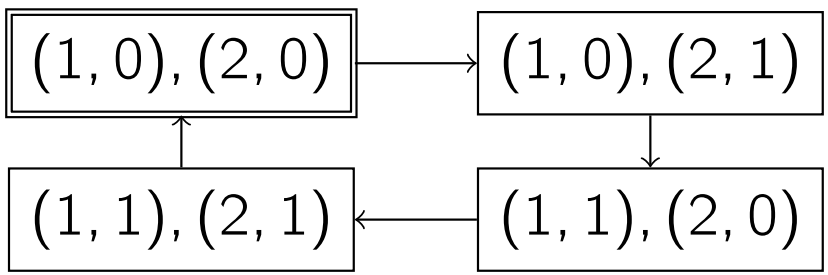
- Все элементы задержки пронумерованы: $1, 2, \ldots, n$
- $S = \{0, 1\}^n$ (*все состояния схемы*)
- $S_0 = \{(0, 0, \ldots, 0)\}$ (*состояние схемы после сброса*)
- $s \to s' \Leftrightarrow$ при переходе к следующему моменту времени (*по переднему фронту тактового сигнала*) возможна такая смена состояния схемы
- $\mathrm{AP} = \{1, 2, \ldots, n\} \times \{0, 1\}$ (*номер и состояние регистра*)
- $(i, b) \in L(b_0, b_1, \ldots, b_n) \Leftrightarrow b_i = b$

Моделирование схем

(*кто знает термин «последовательная схема» — можете представить её вместо схемы из функциональных элементов с задержкой, СФЭЗ*)



Например, модель Крипке для этой схемы может быть устроена так:



Моделирование параллелизма

Современные вычислительные системы зачастую состоят из набора компонентов, исполняющихся одновременно (параллельно) и взаимодействующих друг с другом

В зависимости от природы системы, при построении модели используется один из двух видов параллелизма (или их комбинация):

- Асинхронное исполнение (чередующееся исполнение; семантика чередующихся вычислений; interleaving): шаг вычисления системы отвечает одному шагу выполнения **одного** компонента, а остальные компоненты не выполняются
- Синхронное исполнение: шаг вычисления системы отвечает одновременному выполнению шага вычисления **всех** компонентов

Моделирование параллелизма

Параллельная композиция моделей Крипке $M = (S, S_0, \to, L)$ и $M' = (S', S'_0, \mapsto, L')$ над непересекающимися множествами атомарных высказываний — это модель $M|M' = (S \times S', S_0 \times S'_0, \rightsquigarrow, \mathcal{L})$, где:

- $\mathcal{L}(s, s') = L(s) \cup L'(s')$
- Отношение переходов $\rightsquigarrow$ определяется видом параллелизма

Синхронное исполнение характерно для аппаратных систем,
и других имеющих встроенные средства синхронизации компонентов

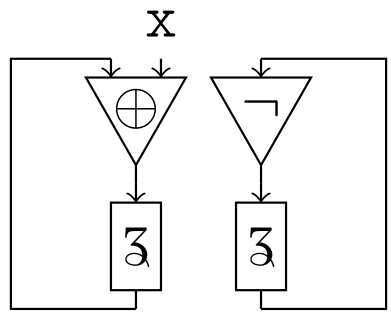
Переходы синхронной композиции моделей
(*без взаимодействия компонентов*) определяются так:

$$(s_1, s'_1) \rightsquigarrow (s_2, s'_2) \quad \Leftrightarrow \quad s_1 \to s_2 \text{ и } s'_1 \mapsto s'_2$$

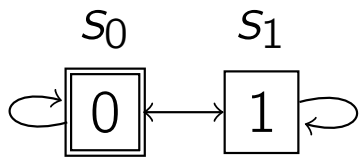
Моделирование параллелизма

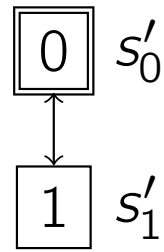
$M = (S, S_0, \to, L)$ $\quad M' = (S', S'_0, \mapsto, L')$ $\quad M|M' = (S \times S', S_0 \times S'_0, \rightsquigarrow, \mathcal{L})$

Например:

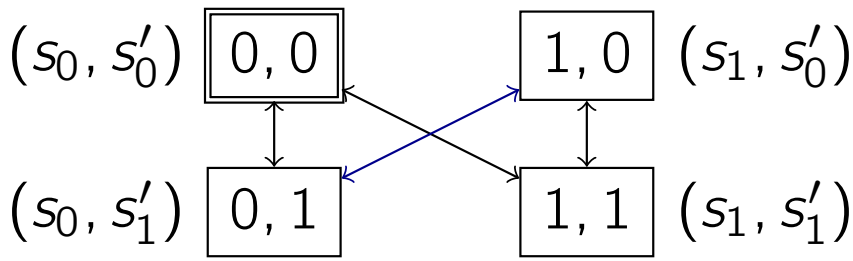


Модели Крипке, описывающие поведение левой и правой задержек:





Синхронная композиция этих моделей:



Моделирование параллелизма

Асинхронное исполнение характерно для систем
без встроенных средств синхронизации компонентов,
в том числе (*с «примесью» синхронности*) для программных систем

Переходы асинхронной композиции моделей
(*без взаимодействия компонентов*) определяются так:

$$(s_1, s_1') \rightsquigarrow (s_2, s_2') \quad \Leftrightarrow \quad (s_1 \to s_2 \text{ и } s_1' = s_2') \text{ или } (s_1 = s_2 \text{ и } s_1' \mapsto s_2')$$

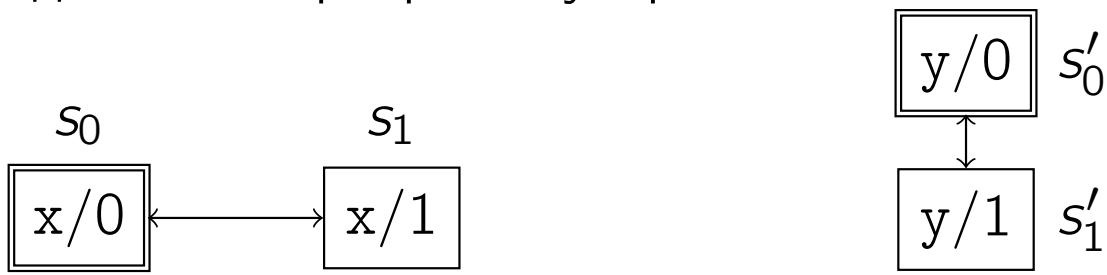
Моделирование параллелизма

Для примера рассмотрим две параллельно работающие программы, в цикле выполняющие одно присваивание:

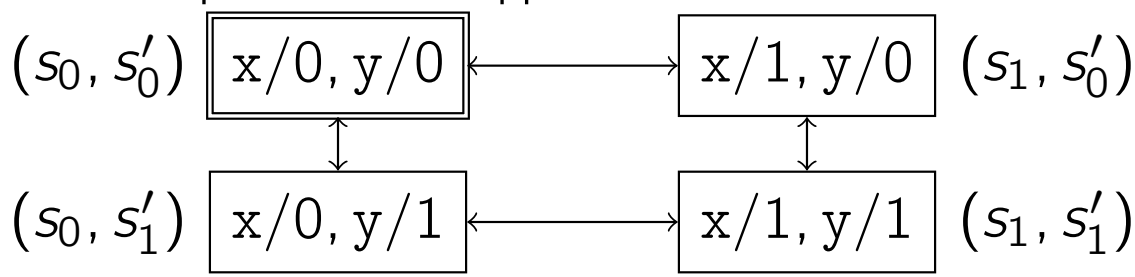
$$\pi_1 : \mathrm{x} := \mathrm{x} + 1; \qquad \pi_2 : \mathrm{y} := \mathrm{y} + 1;$$

Для простоты будем считать, что эти программы выполняются в условиях арифметики одноразрядных чисел с переполнением

Модели Крипке для этих программ устроены так:

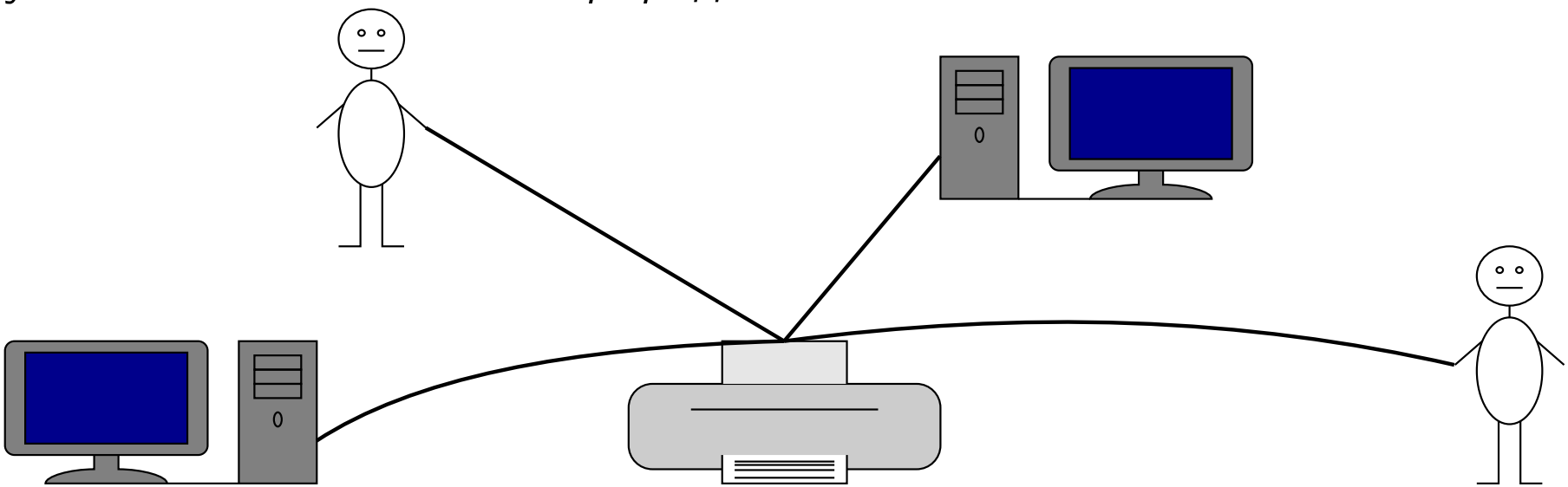


Асинхронная композиция этих моделей:



Моделирование взаимодействия

Пример: с сетевым принтером пытаются взаимодействовать участники *неизвестной природы*



Принтер работает последовательно: принимает информацию и производит печать согласно содержащейся в нём *программе*

Программы остальных участников, *если они есть*, неизвестны

Моделирование взаимодействия

Предположим, что в контроллере принтера
есть однобитовый регистр $\mathtt{R}$, доступный на чтение и запись
всем желающим послать запрос на печать:

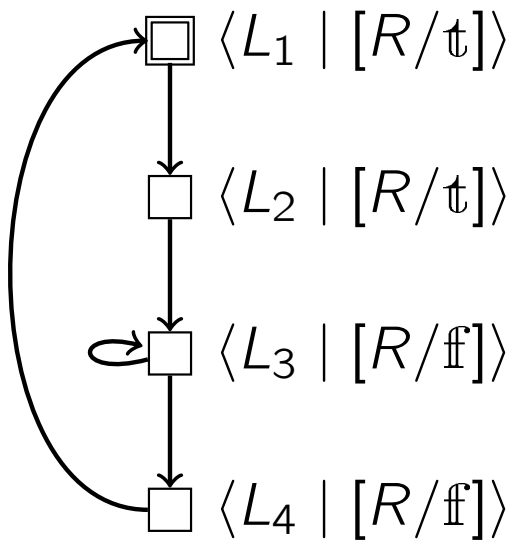
$$\mathtt{R} = \mathtt{t} \Leftrightarrow \text{принтер свободен для печати}$$

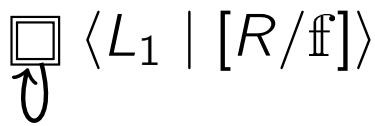
Тогда программу π, посредством которой можно организовать
взаимодействие участника с принтером, можно устроить так:

while $\mathtt{t}$ **do**
$L1:$ **while** $\neg\mathtt{R}$ **do** $\emptyset$ **od**
$L2:$ $\mathtt{R} := \mathtt{f}$;
$L3:$ послать данные для печати
$L4:$ $\mathtt{R} := \mathtt{t}$;
od

Моделирование взаимодействия

Модель Крипке для π: (*функция разметки опущена*)





Моделирование взаимодействия

Программа в вычислительной системе
может взаимодействовать с другими программами:
общие переменные, обмен сообщениями, сигналы, ...

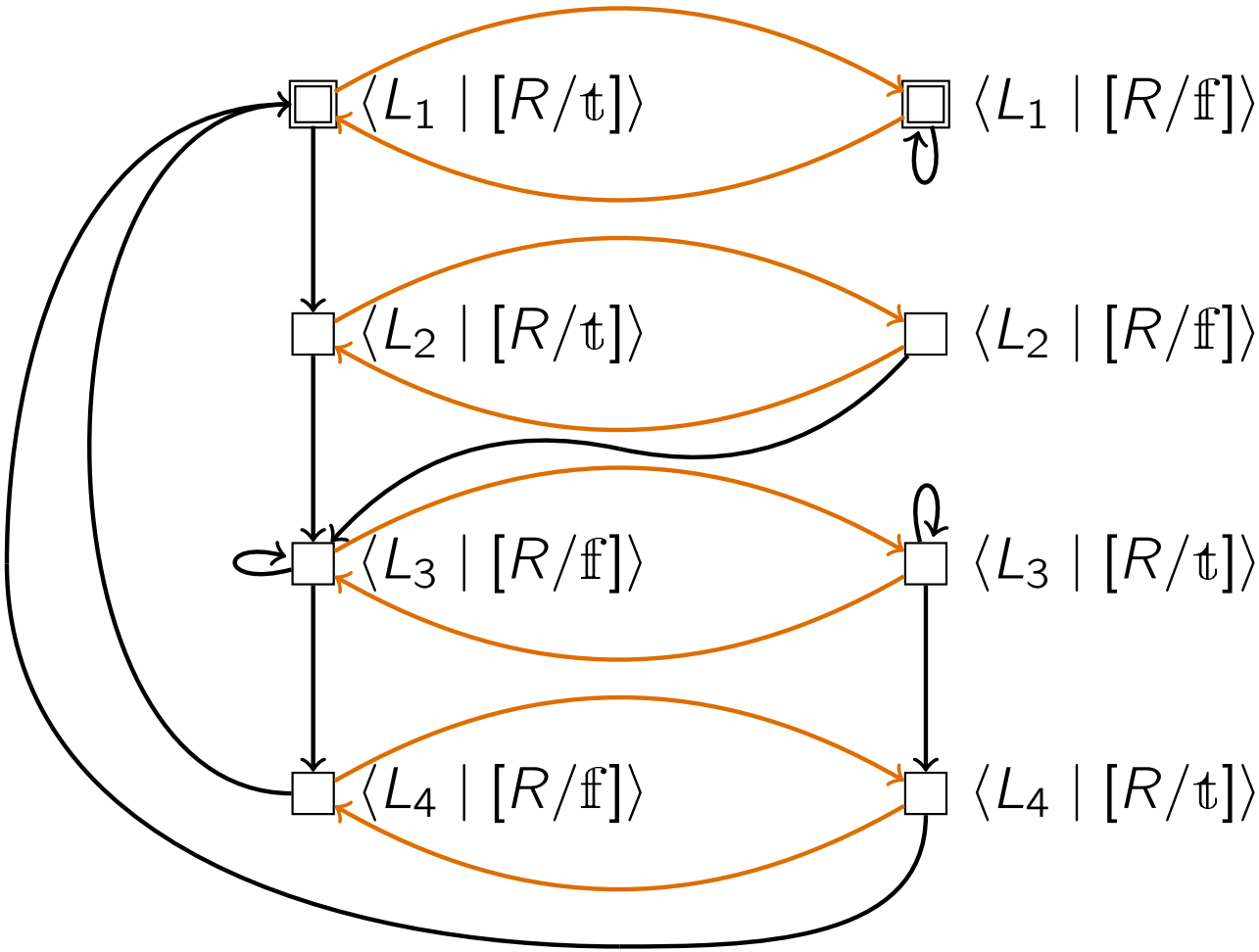
Такое взаимодействие выражается в том, что состояние вычисления программы может измениться под воздействием её **окружения**

Например, регистр `R` в примере может быть изменён любым участником

Чтобы учесть такое изменение, следует добавить в модель Крипке
переходы, отвечающие всем возможностям окружения
повлиять на состояние вычисления программы

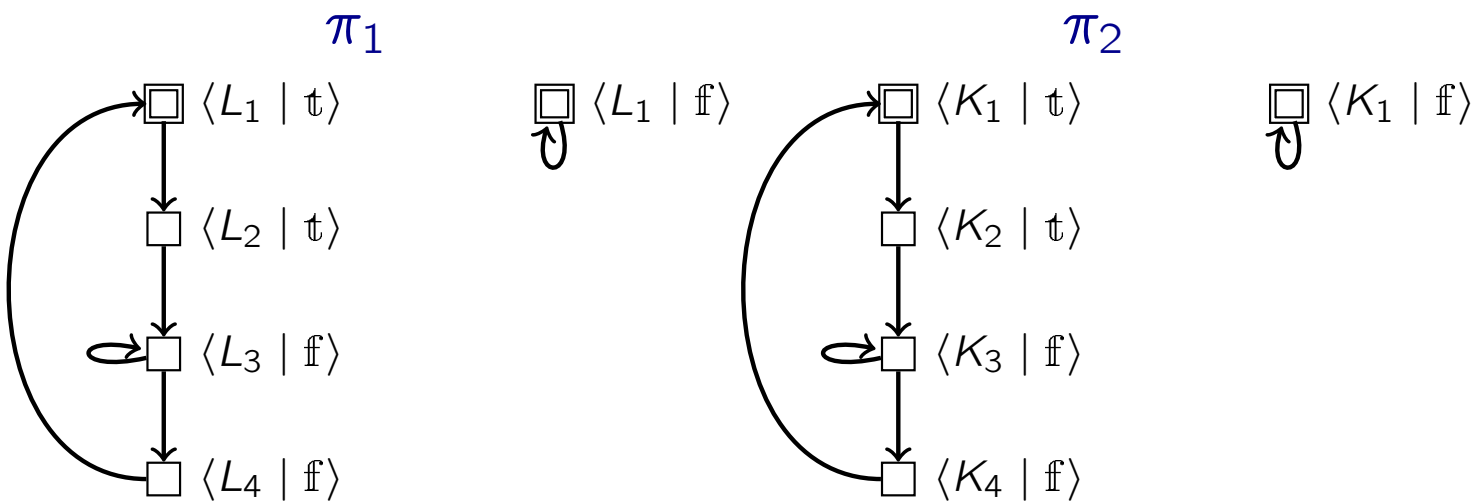
Моделирование взаимодействия

Модель Крипке для программы π с окружением,
способным произвольно переключать значение регистра $\mathtt{R}$:



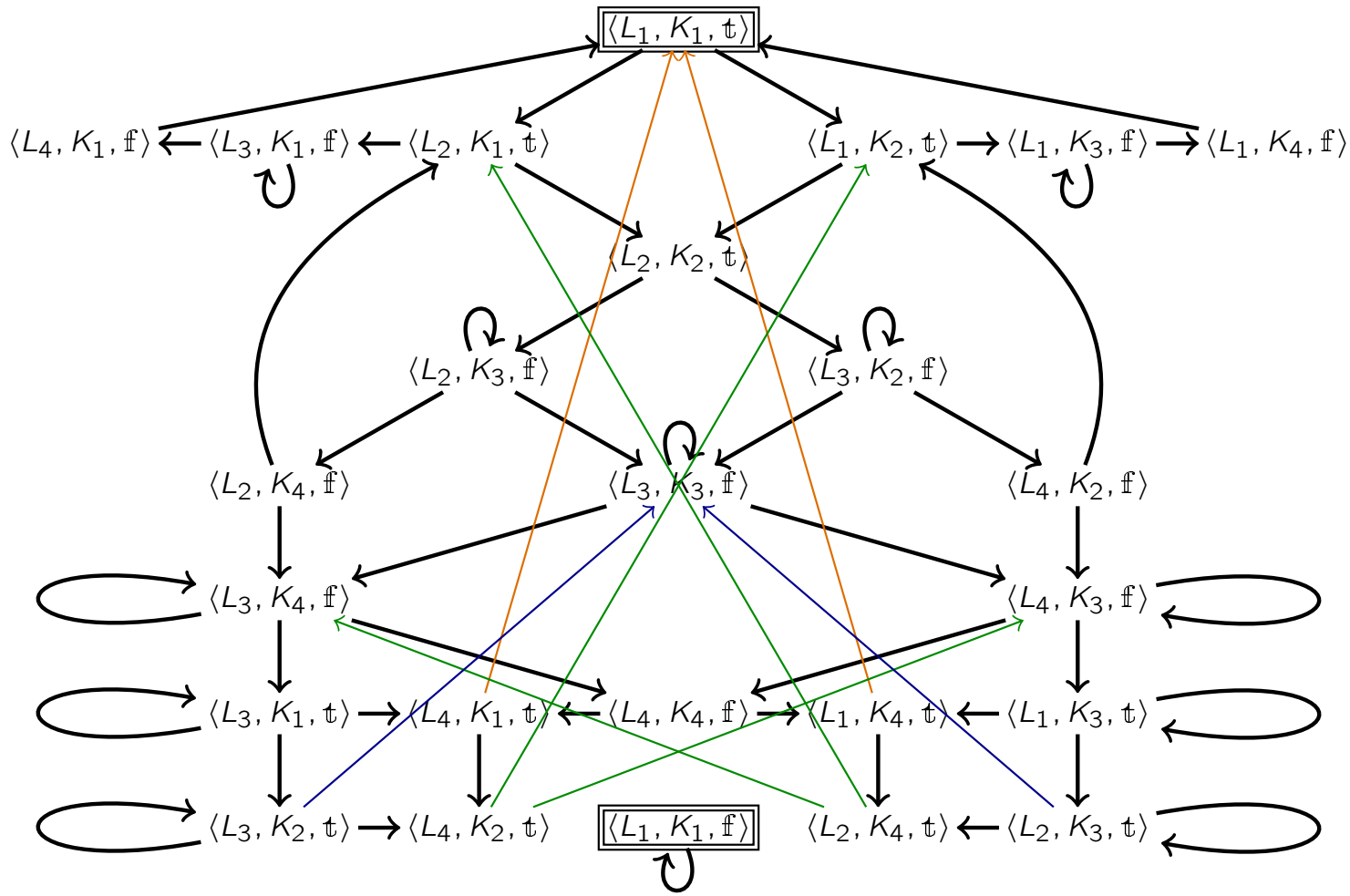
Моделирование взаимодействия

Рассмотрим две (одинаковые) программы взаимодействия с сетевым принтером, выполняющиеся согласно следующим моделям Крипке:



Моделирование взаимодействия

Модель Крипке, описывающая асинхронное исполнение π_1 и π_2 с общим регистром $\mathtt{R}$:



Гранулярность и атомарность в моделях

Переход t в модели отвечает выполнению
атомарного действия системы:

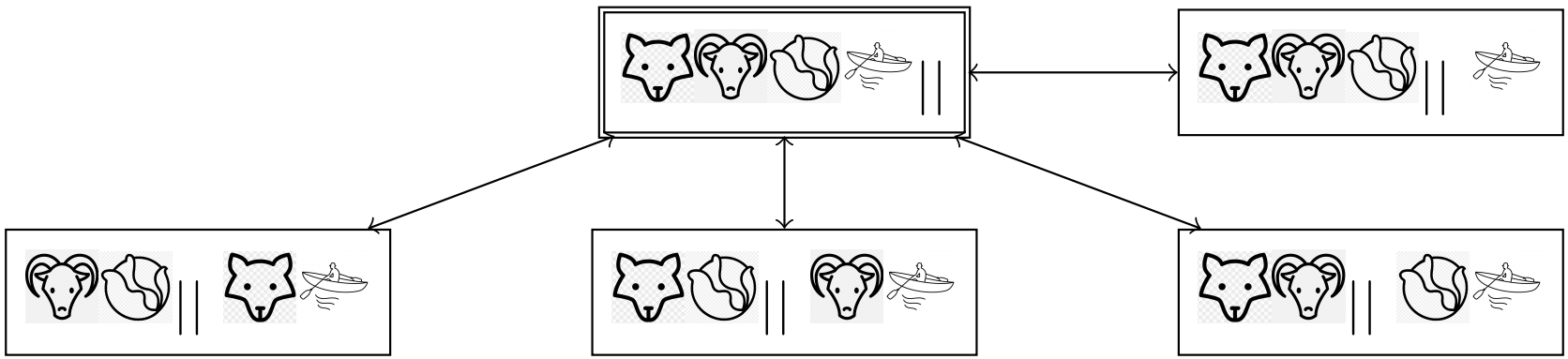
- в выполнение t не могут вмешаться другие действия
- t невозможно или неразумно разделять на более простые действия
- при выполнении t не наблюдаются промежуточные состояния

Выбор гранулярности действий в модели: того, какие именно (насколько детальные или абстрактные) действия будут считаться атомарными — играет важную роль при разработке модели:

- Если действия модели слишком абстрактны, то в модели могут отсутствовать некоторые ошибки, наблюдающиеся при частичном выполнении и «перекрытии» реальных действий
- Если действия модели слишком детальны, то это может
  - существенно увеличить размер модели за счёт несуществующих или «неважных» состояний и из-за этого
  - понизить эффективность верификации

Гранулярность и атомарность в моделях

Например,



Нужно ли рассматривать отдельное действие «плавание по реке»?

А «посадка в лодку» и «высадка из лодки»?

Можно ли посчитать атомарным плавание туда и обратно?

Гранулярность и атомарность в моделях

Другой пример

Рассмотрим две параллельно выполняющиеся программы, каждая из которых выполняет одну команду

$$\pi_1 : \mathtt{x} := \mathtt{x} + \mathtt{y}; \qquad \pi_2 : \mathtt{y} := \mathtt{x} + \mathtt{y};$$

Устроит ли нас, если эти две команды будут считаться атомарными?

Тогда в вычислении системы на $[\mathtt{x}/2, \mathtt{y}/3]$ будут достигаться только состояния данных $[\mathtt{x}/5, \mathtt{y}/3]$, $[\mathtt{x}/5, \mathtt{y}/8]$, $[\mathtt{x}/2, \mathtt{y}/5]$ и $[\mathtt{x}/7, \mathtt{y}/5]$

Но реализация таких присваиваний на языке ассемблера может содержать и более одной команды:

```
load $1, x        load $3, x
load $2, y        load $4, y
add $1, $2        add $3, $4
store $1, x       store $3, y
```

Если атомарными считать ассемблерные команды,
то достижимы и другие состояния данных — например, $[\mathtt{x}/5, \mathtt{y}/5]$ — что может оказаться нежелательным (ошибкой)

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 10

Пара слов о последовательностях

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Для последовательностей $\mathfrak{S} = (x_1, x_2, x_3, \dots)$, $\overline{\mathfrak{S}} = (y_1, y_2, y_3, \dots)$,
в том числе для путей и для трасс в моделях,
будем использовать следующие понятия и обозначения

$\mathfrak{S}[i] = x_i$ — i-й элемент последовательности $\mathfrak{S}$ (нумерация с единицы)

$x \in \mathfrak{S}$ означает, что существует номер i, такой что $x = \mathfrak{S}[i]$

$|\mathfrak{S}|$ — длина последовательности $\mathfrak{S}$: если $\mathfrak{S} = (x_1, \dots, x_n)$, то $|\mathfrak{S}| = n$,
а если последовательность $\mathfrak{S}$ бесконечна, то $|\mathfrak{S}| = \infty$

Конкатенация (сцепление) $\mathfrak{S}\overline{\mathfrak{S}}$, а также $\mathfrak{S} \cdot \overline{\mathfrak{S}}$,
последовательностей $\mathfrak{S}$ и $\overline{\mathfrak{S}}$ задаётся так:

- Если $\mathfrak{S} = (x_1, x_2, \dots, x_n)$, то $\mathfrak{S}\overline{\mathfrak{S}} = (x_1, x_2, \dots, x_n, y_1, y_2, y_3, \dots)$
- Если $\overline{\mathfrak{S}}$ — пустая последовательность, то $\mathfrak{S}\overline{\mathfrak{S}} = \mathfrak{S}$
- В остальных случаях сцепление не задано (не существует)

Для последовательностей $\mathfrak{S} = (x_1, x_2, x_3, \dots)$, $\overline{\mathfrak{S}} = (y_1, y_2, y_3, \dots)$,
в том числе для путей и для трасс в моделях,
будем использовать следующие понятия и обозначения

Если существует последовательность $\mathfrak{S}'$, такая что $\mathfrak{S} = \overline{\mathfrak{S}}\mathfrak{S}'$, то
$\overline{\mathfrak{S}}$ — префикс последовательности $\mathfrak{S}$ и
$\mathfrak{S}$ — продолжение последовательности $\overline{\mathfrak{S}}$

$\mathfrak{S}^{\leq n}$ и $\mathfrak{S}^{<n}$ — префиксы последовательности $\mathfrak{S}$,
имеющие длину n и $n-1$ соответственно

Если существует последовательность $\mathfrak{S}'$, такая что $\mathfrak{S} = \mathfrak{S}'\overline{\mathfrak{S}}$, то
$\overline{\mathfrak{S}}$ — суффикс последовательности $\mathfrak{S}$

$\mathfrak{S}^{\geq n}$ и $\mathfrak{S}^{<n}$ — суффиксы последовательности $\mathfrak{S}$,
такие что $\mathfrak{S} = \mathfrak{S}^{\leq n}\mathfrak{S}^{>n} = \mathfrak{S}^{<n}\mathfrak{S}^{\geq n}$

X^* и X^ω — соответственно множество
всех конечных и всех бесконечных последовательностей,
составленных из элементов множества X

Если сказано, что X — алфавит, то

- элементы множества X называются буквами и символами,
- элементы множества X^* — словами, а
- элементы множества X^ω — ω-словами

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

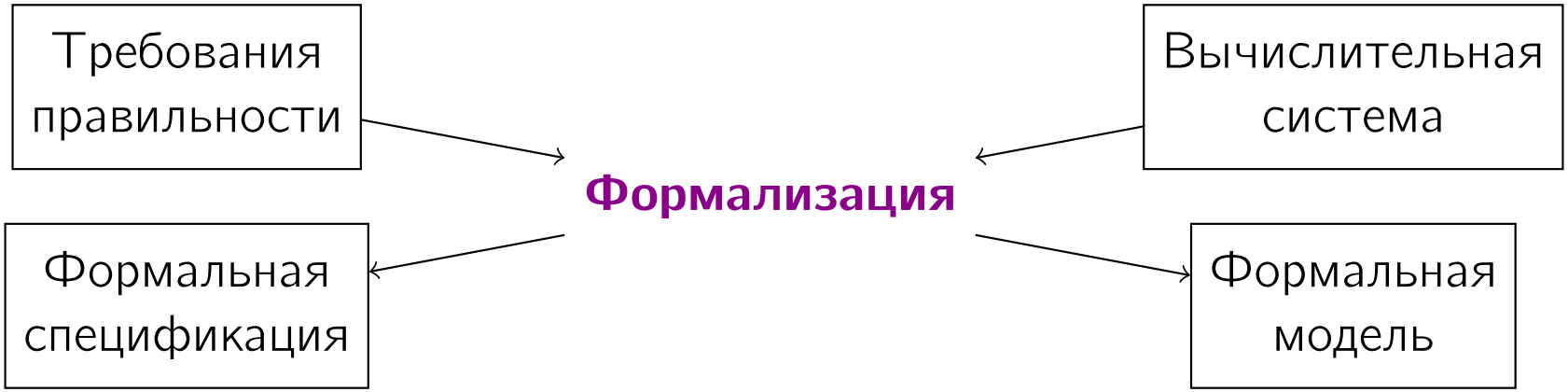
Блок 11

Свойства трасс
Безопасность и живость

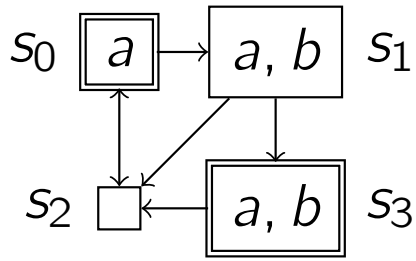
Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Напоминание



Модель Крипке M над множеством атомарных высказываний $\{a, b\}$:



Вычисление τ модели M (бесконечный начальный путь):

$$s_3 \to s_2 \to s_0 \to s_2 \to s_0 \to s_1 \to s_3 \to s_2 \to \ldots$$

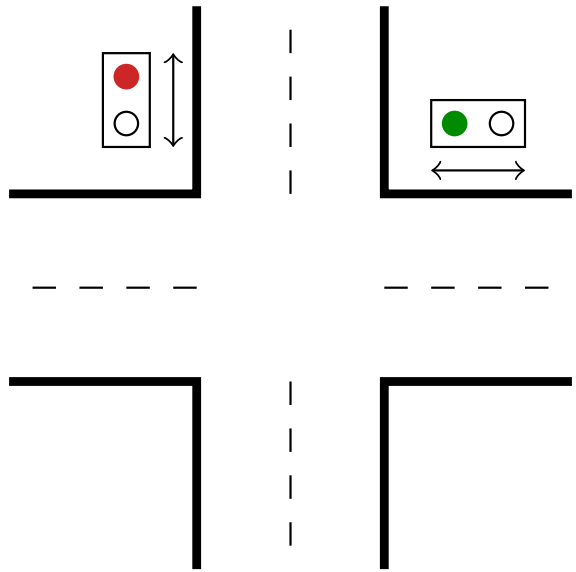
Трасса вычисления τ:

$$\{a, b\}, \emptyset, \{a\}, \emptyset, \{a\}, \{a, b\}, \{a, b\}, \emptyset, \ldots$$

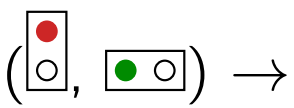
Перейдём к тому, как могут быть устроены формальные спецификации моделей Крипке и соответствующие требования, предъявляемые к вычислительным системам

Свойства трасс

Пример: перекрёсток со светофорами

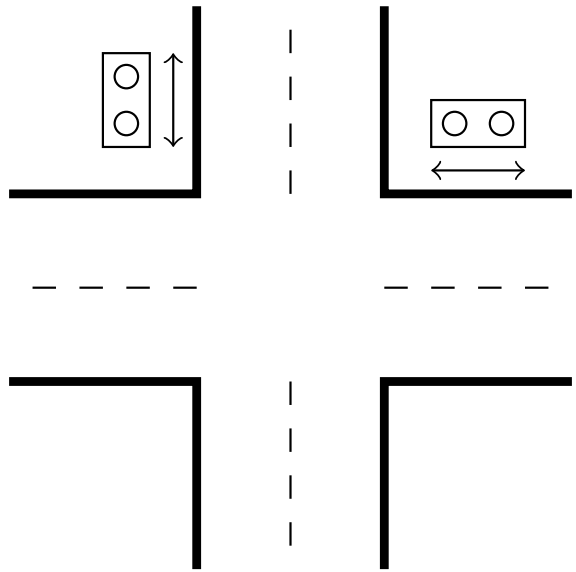


Пример вычисления этой системы:

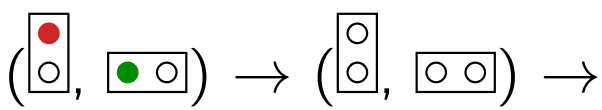


Свойства трасс

Пример: перекрёсток со светофорами

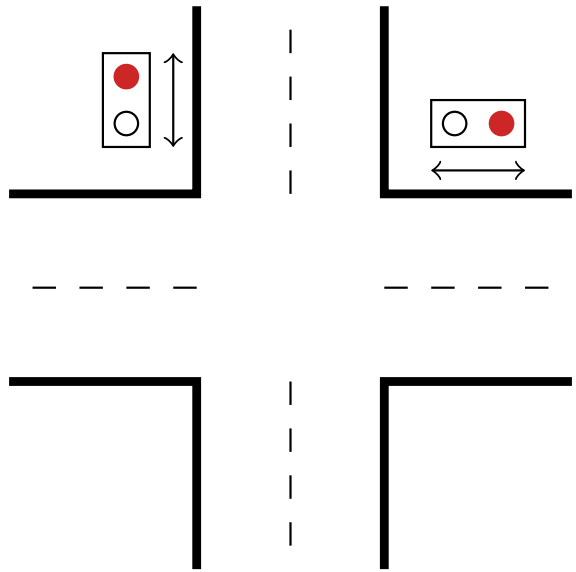


Пример вычисления этой системы:

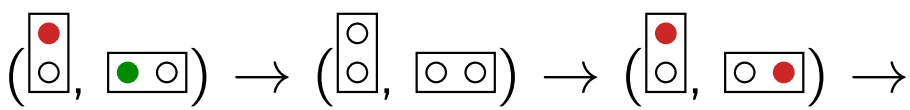


Свойства трасс

Пример: перекрёсток со светофорами

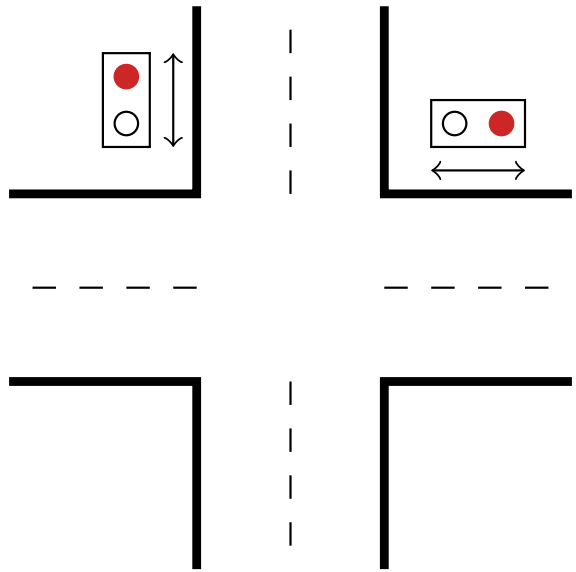


Пример вычисления этой системы:

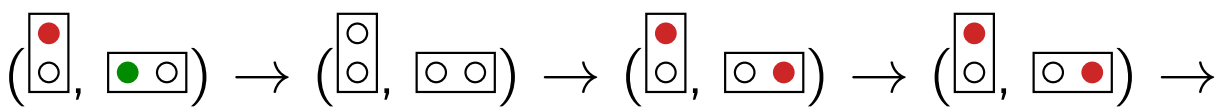


Свойства трасс

Пример: перекрёсток со светофорами

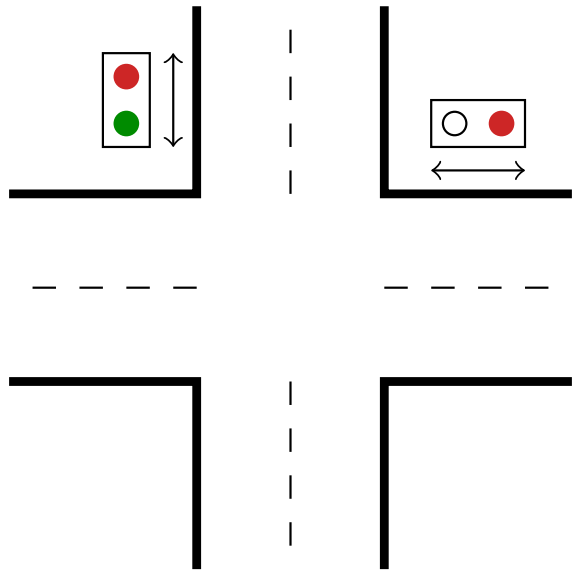


Пример вычисления этой системы:

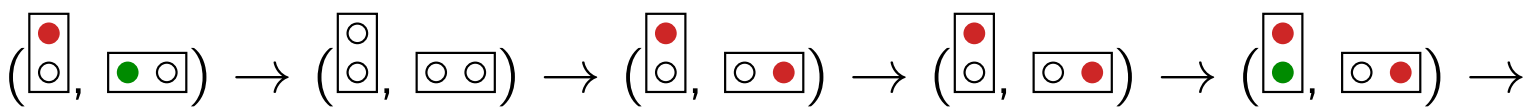


Свойства трасс

Пример: перекрёсток со светофорами

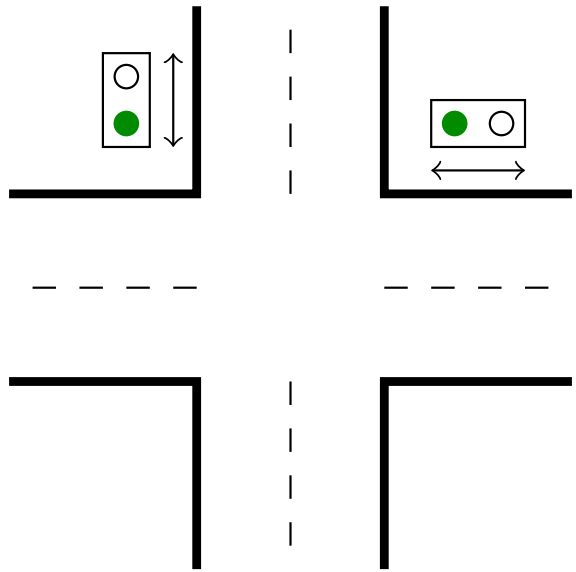


Пример вычисления этой системы:



Свойства трасс

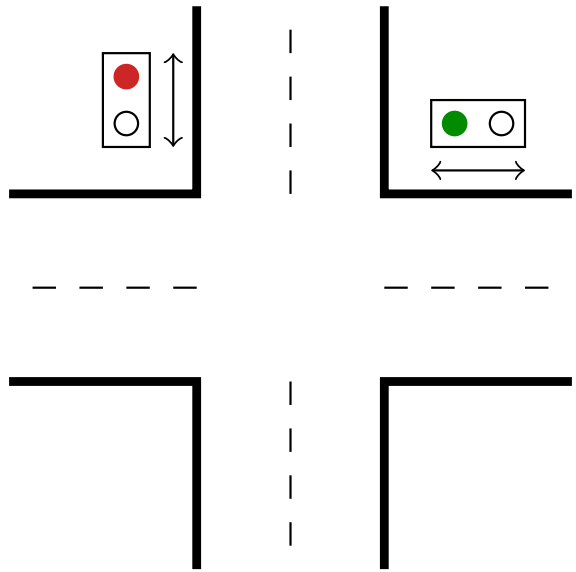
Пример: перекрёсток со светофорами



Пример вычисления этой системы:

Свойства трасс

Пример: перекрёсток со светофорами

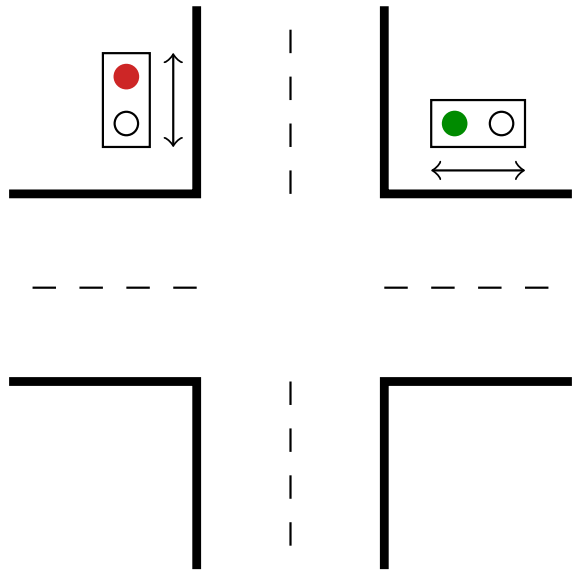


Какие требования было бы разумно предъявить к этой системе:

- Никакой светофор никогда не может быть • и • одновременно
- Сколько бы ни выполнялась система,
  каждый светофор рано или поздно ещё хотя бы раз станет •
- Никогда светофоры не будут • одновременно
- Каждый светофор бесконечно часто бывает и •, и •

Свойства трасс

Пример: перекрёсток со светофорами



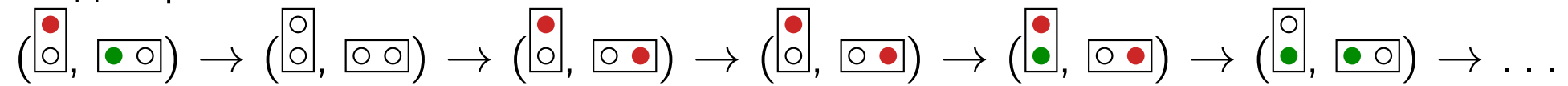
Модель Крипке, составленную с учётом этих требований, разумно строить над такими атомарными высказываниями:

- $r_{\updownarrow}$: светофор $\updownarrow$ красный
- $g_{\updownarrow}$: светофор $\updownarrow$ зелёный
- $r_{\leftrightarrow}$: светофор $\leftrightarrow$ красный
- $g_{\leftrightarrow}$: светофор $\leftrightarrow$ зелёный

Свойства трасс

Пример: перекрёсток со светофорами

Тогда трасса вычисления



устроена так:

$$\{r_{\updownarrow}, g_{\leftrightarrow}\}, \quad \emptyset, \quad \{r_{\updownarrow}, r_{\leftrightarrow}\}, \quad \{r_{\updownarrow}, r_{\leftrightarrow}\}, \quad \{r_{\updownarrow}, g_{\updownarrow}, r_{\leftrightarrow}\}, \quad \{g_{\updownarrow}, g_{\leftrightarrow}\}, \quad \ldots$$

Свойством трасс будем называть любое множество трасс

Будем говорить, что трасса τ обладает свойством P, если $\tau \in P$

Например, требование «Никогда светофоры не будут $\bullet$ одновременно» отвечает свойству

$$\{\tau \mid \tau \in (2^{\mathsf{AP}})^{\omega}, \quad \forall \sigma \in \tau : \{g_{\leftrightarrow}, g_{\updownarrow}\} \not\subseteq \sigma\},$$

и трасса, изображённая выше, не обладает этим свойством

Свойства трасс

Для модели Крипке $M = (S, S_0, \to, L)$ над AP и её состояния s будем использовать такие понятия и обозначения:

- $\Pi(M, s)$ — множество всех бесконечных путей в M, исходящих из s
- $\Pi(M)$ — множество всех вычислений M
  - То есть $\Pi(M) = \bigcup\limits_{s_0 \in S_0} \Pi(M, s_0)$
- $\mathrm{Tr}(M, s)$ — множество всех трасс путей из $\Pi(M, s)$
- $\mathrm{Tr}(M)$ — множество всех трасс вычислений из $\Pi(M)$
  - То есть $\mathrm{Tr}(M) = \bigcup\limits_{s_0 \in S_0} \mathrm{Tr}(M, s_0)$
- Tr — множество всех трасс (для заданного множества AP)
- Tr_f — множество всех конечных трасс
- M удовлетворяет свойству трасс P ($M \models P$), если $\mathrm{Tr}(M) \subseteq P$

Свойства трасс

Пояснение соотношения $M \models P$
для модели Крипке M и свойства трасс P:

- Всевозможные трассы делятся свойством P на хорошие (обладающие свойством P) и плохие (не обладающие свойством P)
- Соотношение $M \models P$ означает, что все трассы модели M хорошие (т.е. что в модели M нет ни одной плохой трассы)

Утверждение
Для любых моделей Крипке M, M' и свойства трасс P верно: если $\mathrm{Tr}(M) \subseteq \mathrm{Tr}(M')$ и $M' \models P$, то $M \models P$

Доказательство. Очевидным образом следует из определений и из свойства транзитивности включения множеств

Свойства безопасности и живости

При анализе поведения систем
зачастую рассматриваются свойства трасс двух классов:

- Свойства безопасности
  - Safety properties

- Свойства живости
  - Или, по-другому, — свойства живучести
  - Liveness properties

Свойства безопасности и живости

Свойство трасс P называется свойством безопасности,
если у любой трассы τ, не обладающей этим свойством,
существует конечный префикс,
любое бесконечное продолжение которого не обладает этим свойством

$$(\forall \tau \in \mathrm{Tr} \setminus P : \exists \tau_1 \in \mathrm{Tr}_f : \exists \tau_2 \in \mathrm{Tr} : \tau = \tau_1\tau_2 \text{ и } \forall \tau_3 \in \mathrm{Tr} : \tau_1\tau_3 \notin P)$$

Пояснение

Трассы, обладающие свойством P, считаются безопасными,
а не обладающие — опасными

При этом понятие (без)опасности подобрано так, что если трасса

$$\sigma_1 \to \sigma_2 \to \sigma_3 \to \ldots$$

опасна, то существует обозримая (конечная) совокупность событий

$$\sigma_1 \to \sigma_2 \to \sigma_3 \to \cdots \to \sigma_n,$$

от начала работы системы до некоторого (конечного) момента времени,
по которой и все другие трассы можно признать опасными

То есть опасность, однажды наступив,
не может быть устранена дальнейшими событиями

Свойства безопасности и живости

Примеры требований, отвечающих свойствам безопасности:

- Два процесса не обратятся одновременно к одной ячейке памяти
  - *Опасность*: сейчас два процесса обращаются к одной ячейке
- Пока принтер не завершит печать,
  он не доступен для других устройств
  - *Опасность*: сейчас принтер занят печатью для одного устройства
    и при этом доступен для другого
- Команда выполняется процессором не более трёх тактов подряд
  - *Опасность*: команда выполняется четыре последних такта
- Красный свет загорается только после жёлтого
  - *Опасность*:
    раньше жёлтый свет не загорался, а сейчас горит красный

Свойства безопасности и живости

Свойство трасс P называется свойством живости,
если для любой конечной трассы
существует бесконечное продложение, обладающее этим свойством

$$(\forall \tau_1 \in \mathrm{Tr}_f : \exists \tau_2 \in \mathrm{Tr} : \tau_1 \tau_2 \in P)$$

Пояснение

Определение живости можно прочитать так:
как бы ни работала система, обязательно есть возможность ей выполняться дальше так, чтобы она была признана живой (не сломавшейся, не зависшей, не отключившейся, ...)

Способ продолжения произвольной трассы до входящей в P — это способ подтверждения живости, не зависящий от истории событий до текущего момента и задающийся в терминах
как конечного, так и бесконечного продолжения работы системы

Мёртвая в этом смысле система — это такая, которая
после выполнения некоторых действий оказалась неспособной
ни при каких обстоятельствах подтвердить свою живость

Свойства безопасности и живости

Примеры требований, отвечающих свойствам живости:

- Рано или поздно загорится зелёный свет
  - *Мёртвая система* больше не может зажечь зелёный свет
- После завершения печати принтер стирает содержимое буфера
  - *Мёртвая система* ни при каких условиях не опустошит буфер
- Рано или поздно наступит следующий такт работы процессора
  - *Мёртвая система* потеряла возможность осциллировать тактовым сигналом
- Процесс бесконечно часто обращается к заданной ячейке памяти
  - *Мёртвая система* может обратиться к ячейке памяти лишь конечное число раз

Свойства безопасности и живости

Утверждение. **Если свойство трасс P является и свойством безопасности, и свойством живости, то $P = \mathrm{Tr}$**

Доказательство.

Можете попробовать самостоятельно, это не очень трудно

Утверждение. **Для любого свойства трасс P существуют такие свойство безопасности P_s и свойство живости P_ℓ, для которых верно $P = P_s \cap P_\ell$**

Доказательство.

Можете попробовать самостоятельно (и это может быть трудно)

Свойства безопасности и живости

Пример

Процесс при запуске открывает файл
и затем бесконечно часто обращается к этом файлу

Это не свойство безопасности: если файл открыт при запуске, то для этого случая невозможно подобрать подходящее понятие опасности

Это не свойство живости: если файл не открыт при запуске, то позже невозможно сделать его «открытым при запуске»

Но это пересечение свойства безопасности

Процесс при запуске открывает файл

и свойства живости

Процесс бесконечно часто обращается к файлу

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 12

Логика линейного времени (LTL)

Постановка задачи верификации моделей Крипке относительно LTL

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Напоминание

Для моделей Крипке обсудили «лобовой» теоретико-множественный способ задания спецификаций в виде свойств трасс

Но свойство трасс — это множество, состоящее из бесконечных последовательностей и являющееся в общем случае бесконечным

Для эффективного использования таких свойств необходимо иметь удобный язык их представления

Пара слов о логике высказываний

Самый простой логический язык записи спецификаций,
в сравнении с которым можно объяснять устройство других языков —
это язык логики высказываний (ЛВ)

Иногда для языков предлагаются два варианта синтаксиса:

- Полный с широким спектром синтаксических конструкций
  «на все случаи жизни»
- Краткий с небольшим набором синтаксических конструкций
  и способом выражения остальных через него

Будем придерживаться «ленивого» способа изложения языков:

- Строго вводится краткий синтаксис с семантикой
- Остальные конструкции полного синтаксиса
  вводятся как сокращения
- Смысл всех конструкций содержательно поясняется
- В примерах, иллюстрациях, выкладках и т.п. используется
  полный синтаксис согласно содержательным пояснениям
- В обоснованиях по желанию используется краткий синтаксис

Пара слов о логике высказываний

БНФ, задающая краткий синтаксис формул логики высказываний над множеством атомарных высказываний AP:

$$\varphi \ ::= \ \mathbb{t} \ \mid \ p \ \mid \ (\varphi \,\&\, \varphi) \ \mid \ (\neg\varphi),$$

где $p \in \mathrm{AP}$ и φ — формула

Операции $\&$ и $\neg$ имеют естественный содержательный смысл: связки «и» и «не» в предложениях

Другие операции полного синтаксиса:

- $\mathbb{f} = \neg\mathbb{t}$
- $\psi_1 \vee \psi_2 = \neg((\neg\psi_1)\,\&\,(\neg\psi_2))$
  - Содержательный смысл: союз «или» в неисключающем смысле
- $\psi_1 \to \psi_2 = (\neg\psi_1) \vee \psi_2$
  - Содержательный смысл: «если-то»

Приоритеты операций по убыванию: $\neg$; $\&$; $\vee$; $\to$

Скобки в формулах нередко опускаются согласно приоритетам и согласно ассоциативности операций $\&$ и $\vee$

Пара слов о логике высказываний

Правила, задающие выполнимость формулы φ
в интерпретации $\mathcal{I} : \mathrm{AP} \to \{\mathbb{t}, \mathbb{f}\}$ ($\mathcal{I} \models \varphi$):

- Соотношение $\mathcal{I} \models \mathbb{t}$ выполняется всегда
- $\mathcal{I} \models p \quad \Leftrightarrow \quad \mathcal{I}(p) = \mathbb{t}$
- $\mathcal{I} \models (\psi_1 \,\&\, \psi_2) \quad \Leftrightarrow \quad \mathcal{I} \models \psi_1$ и $\mathcal{I} \models \psi_2$
- $\mathcal{I} \models (\neg\varphi) \quad \Leftrightarrow \quad \mathcal{I} \not\models \varphi$

Например,

для интерпретации $\mathcal{I}$, такой что $\mathcal{I}(\mathrm{x}) = \mathbb{t}$ и $\mathcal{I}(\mathrm{y}) = \mathbb{f}$, верно следующее:

- $\mathcal{I} \models \mathrm{x}$
- $\mathcal{I} \not\models (\neg x)$
- $\mathcal{I} \not\models \mathrm{y}$
- $\mathcal{I} \models (\neg\mathrm{y})$
- $\mathcal{I} \models (\mathrm{x} \,\&\, \neg\mathrm{y})$
- $\mathcal{I} \not\models (\mathrm{x} \,\&\, \mathrm{y})$

Темпоральные логики

В логике высказываний выполнимость формулы зависит от и определяется для истинностных значений атомарных высказываний

В темпоральных логиках учитывается изменение значений атомарных высказываний с течением **времени**, и выполнимость формулы зависит от выбора рассматриваемого момента времени и от взаимосвязи значений атомарных высказываний в различные моменты времени

Операции темпоральной логики, как правило, делятся на

- операции ЛВ с тем же содержательным смыслом, что и в ЛВ, и
- темпоральные операции, позволяющие рассуждать о взаимосвязи значений высказываний в различные моменты времени

Прежде всего обсудим наиболее известную и популярную логику, предназначенную для записи свойств трасс:
логику линейного времени (LTL)

Логика линейного времени (LTL)

БНФ, задающая краткий синтаксис ltl-формул
над множеством атомарных высказываний AP:

$$\varphi ::= \mathbb{t} \mid p \mid (\varphi \,\&\, \varphi) \mid (\neg\varphi) \mid (\mathbf{X}\varphi) \mid (\varphi \mathbf{U} \varphi),$$

где $p \in \mathrm{AP}$ и φ — ltl-формула

Моментами времени дальше будем называть
натуральные чи́сла ($\mathbb{N}$): $1, 2, 3, \ldots$

Понятие выполнимости формул в LTL уточняется так:
формула выполняется или не выполняется (*истинна или ложна*)
не «абсолютно», а в те или иные моменты времени

X и **U** — это темпоральные операции
со следующим содержательным смыслом:

- $\mathbf{X}\varphi$: φ будет выполнено в следующий момент времени
  (*относительно рассматриваемого момента*)
- $\psi_1 \mathbf{U} \psi_2$: в будущем когда-нибудь выполнится ψ_2,
  а до тех пор всегда будет выполняться ψ_1

Логика линейного времени (LTL)

Другие операции полного синтаксиса:

- $\mathbf{F}\varphi = \mathbb{t}\mathbf{U}\varphi$
  - Дословное прочтение: в будущем когда-нибудь выполнится φ, а до тех пор всегда будет выполняться $\mathbb{t}$
  - Иными словами: в будущем когда-нибудь выполнится φ
- $\mathbf{G}\varphi = \neg(\mathbf{F}(\neg\varphi))$
  - Дословное прочтение: неверно то, что когда-нибудь в будущем выполнится формула не-φ
  - Иными словами: в будущем всегда будет выполняться формула φ

Приоритеты операций по убыванию:

$\neg$, **F**, **G** и **X**;

затем **U**;

затем остальные операции с обычными приоритетами

Логика линейного времени (LTL)

Примеры формул,
выражающих требования правильности вычислительных систем:

- Никогда светофоры $\updownarrow$ и $\leftrightarrow$ не будут $\bullet$ одновременно
$$\neg \mathbf{F}(g_{\updownarrow} \,\&\, g_{\leftrightarrow})$$
- Когда-нибудь наступит лето, а до тех пор будет холодно
$$\mathit{cold}\,\mathbf{U}\,\mathit{summer}$$
- Двух подряд плохих дней не бывает:
$$\mathbf{G}(\mathit{bad\_day} \rightarrow \mathbf{X}\neg \mathit{bad\_day})$$
- После $\bullet$ светофор $\updownarrow$ рано или поздно станет $\bullet$
$$\mathbf{G}(r_{\updownarrow} \rightarrow \mathbf{F}\, g_{\updownarrow})$$
- Светофор $\updownarrow$ бесконечно часто бывает $\bullet$
$$\mathbf{GF}\, g_{\updownarrow}$$
- Мне уготована вечность в раю или в аду
$$\mathbf{F}(\mathbf{G}\mathit{heaven} \vee \mathbf{G}\mathit{hell})$$

Логика линейного времени (LTL)

Роль интерпретаций для ltl-формул играют трассы:
событием $\tau[i]$ трассы τ задаются истинностные значения всех атомарных высказываний в момент времени i

Семантика ltl-формул задаётся отношением выполнимости ltl-формулы φ на трассе τ ($\tau \models \varphi$):

- Соотношение $\tau \models \mathbb{t}$ верно всегда
- $\tau \models p$, где $p \in \mathrm{AP} \quad \Leftrightarrow \quad p \in \tau[1]$
- $\tau \models (\psi_1 \,\&\, \psi_2) \quad \Leftrightarrow \quad \tau \models \psi_1$ и $\tau \models \psi_2$
- $\tau \models (\neg\varphi) \quad \Leftrightarrow \quad \tau \not\models \varphi$
- $\tau \models (\mathbf{X}\varphi) \quad \Leftrightarrow \quad \tau^{\geq 2} \models \varphi$
- $\tau \models (\psi_1 \mathbf{U} \psi_2) \quad \Leftrightarrow$ существует момент времени i, такой что
  - $\tau^{\geq i} \models \psi_2$ и
  - для любого момента времени j, такого что $j < i$, верно $\tau^{\geq j} \models \psi_1$

Запись $\tau^{\geq m} \models \varphi$ можно содержательно трактовать как выполнимость формулы φ на трассе τ в момент времени m

Логика линейного времени (LTL)

Утверждение (семантика F)

Для любых ltl-формулы φ и трассы τ верно:

$\tau \models \mathbf{F}\varphi \quad \Leftrightarrow \quad$ **существует момент времени i, такой что** $\tau^{\geq i} \models \varphi$

Утверждение (семантика G)

Для любых ltl-формулы φ и трассы τ верно:

$\tau \models \mathbf{G}\varphi \quad \Leftrightarrow \quad$ **для любого момента времени i верно** $\tau^{\geq i} \models \varphi$

Доказательство. Очевидным образом следует из определений **F** и **G** и семантики ltl-формул

Логика линейного времени (LTL)

Утверждение. **Для любых ltl-формулы φ и трассы τ верно:**

$\tau \models \mathbf{GF}\varphi \quad \Leftrightarrow$ **для бесконечного числа попарно различных моментов времени i верно $\tau^{\geq i} \models \varphi$**

Доказательство. Перепишем это утверждение «негативно»:

$\tau \not\models \mathbf{GF}\varphi \quad \Leftrightarrow$ для не более чем конечного числа моментов времени i верно $\tau^{\geq i} \models \varphi$

($\Rightarrow$) Пусть $\tau \not\models \mathbf{GF}\varphi$

По семантике **F** и **G**, верно следующее: существует момент времени k, такой что для любого момента k', такого что $k' \geq k$, верно $\tau^{\geq k'} \not\models \varphi$

Значит, соотношение $\tau^{\geq i} \models \varphi$ выполняется только для $i < k$,
то есть для не более чем k моментов времени

($\Leftarrow$) Пусть соотношение $\tau^{\geq i} \models \varphi$ выполняется для
не более чем конечного числа моментов времени i

Рассмотрим наибольший момент времени k, такой что $\tau^{\geq k} \models \varphi$

Тогда для любого момента k', такого что $k' \geq k + 1$, верно $\tau^{\geq k'} \not\models \varphi$

По семантике **F** и **G**, это означает, что $\tau \not\models \mathbf{GF}\varphi$ ▼

Логика линейного времени (LTL)

Будем говорить, что формула выполняется почти всегда
(*более широко — нечто справедливо почти всегда*),
если она выполняется во все моменты времени,
кроме, быть может, некоторого конечного числа моментов

Утверждение. **Для любых ltl-формулы φ и трассы τ верно:**
формула φ почти всегда выполняется на τ $\Leftrightarrow$
существует момент времени k, такой что
φ выполняется на τ во все моменты, не меньшие k

Доказательство.

($\Rightarrow$) Пусть φ почти всегда выполняется на τ
Тогда множество $X = \{i \mid i \in \mathbb{N}, \tau^{\geq i} \not\models \varphi\}$ конечно
Пусть k — наибольший момент из X
Тогда φ выполняется на τ во все моменты, не меньшие $k+1$

($\Leftarrow$) Пусть существует момент k, такой что
для всех моментов k', не меньших k, верно $\tau^{\geq k'} \models \varphi$
Тогда множество $X = \{i \mid i \in \mathbb{N}, \tau^{\geq i} \not\models \varphi\}$ включено в $\{1, 2, \ldots, k-1\}$
Следовательно, множество X конечно ▼

Логика линейного времени (LTL)

Утверждение. **Для любых ltl-формулы φ и трассы τ верно:**
$\tau \models \mathbf{FG}\varphi \quad \Leftrightarrow \quad$ **формула φ выполняется на τ почти всегда**

Доказательство.

По предыдущему утверждению,
φ выполняется на τ почти всегда $\Leftrightarrow$
существует момент k, такой что
для любого момента k', не меньшего k, верно $\tau^{\geq k} \models \varphi$

По семантике **F** и **G**, последнее соотношение равносильно $\tau \models \mathbf{FG}\varphi$ ▼

Задача model checking относительно LTL

В блоке 11 для модели Крипке M и свойства трасс P было введено обозначение $M \models P$ того, что модель M удовлетворяет свойству P

Ltl-формула φ может восприниматься как
способ представления свойства трасс $\mathrm{Tr}(\varphi) = \{\tau \mid \tau \in \mathrm{Tr}, \tau \models \varphi\}$

Ltl-формула φ выполняется на модели M ($M \models \varphi$), если
справедливо включение $\mathrm{Tr}(M) \subseteq \mathrm{Tr}(\varphi)$

Небольшое пояснение:

- Ltl-формула делит все трассы на хорошие (на которых формула выполняется) и плохие (на которых формула не выполняется)
- Соотношение $M \models \varphi$ означает, что все трассы модели M хорошие (т.е. что в модели M нет ни одной плохой трассы)

Задача model checking для LTL (MC-LTL) формулируется так:

Для заданной модели Крипке M и заданной ltl-формулы φ проверить справедливость соотношения

$$M \models \varphi$$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 13

Размеченные системы переходов
Справедливость для систем переходов
Справедливость в LTL

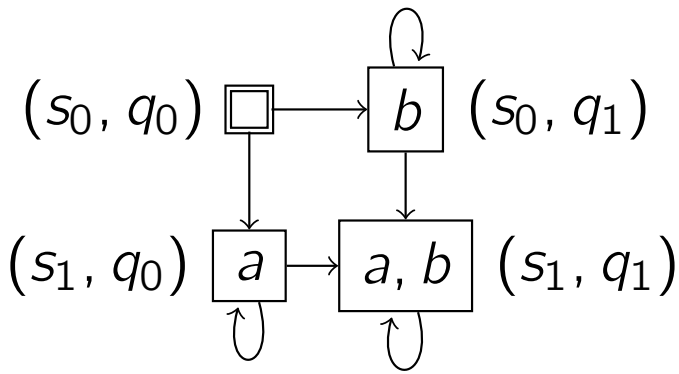
Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступительный пример

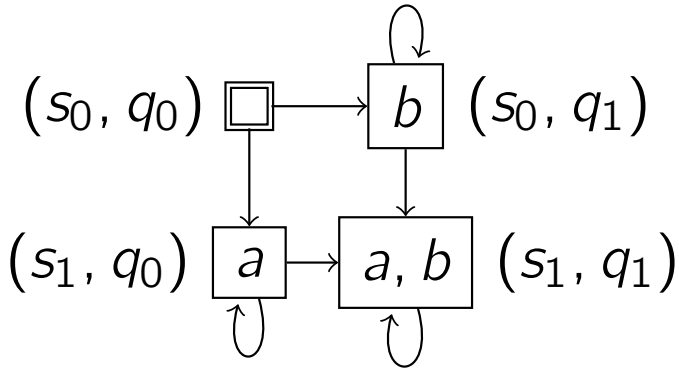
Рассмотрим такие две модели Крипке:

Асинхронная композиция этих моделей Крипке устроена так:



Насколько «реалистична» такая композиция?

Вступительный пример



Представим себе, что исходные модели отвечают программам π_1 и π_2,
асинхронная композиция — их параллельному выполнению,
a означает, что π_1 выполнила своё единственное действие и
b означает что π_2 выполнила своё единственное действие

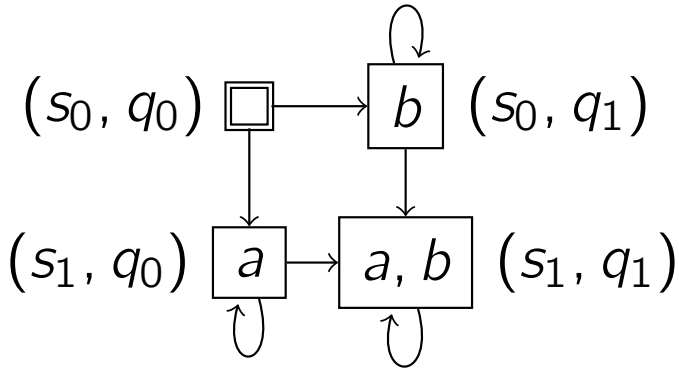
«Реальность» говорит, что если наблюдать
за независимым параллельным выполнением π_1 и π_2
достаточно долго, то каждая из программ выполнит своё действие

При этом в асинхронной композиции
есть «нереалистичное» вычисление

$$(s_0, q_0) \to (s_1, q_0) \to (s_1, q_0) \to (s_1, q_0) \to \ldots,$$

в котором π_2 не выполняет ни одного действия

Вступительный пример



$$\rho = ((s_0, q_0) \to (s_1, q_0) \to (s_1, q_0) \to (s_1, q_0) \to \dots),$$

При этом каждый конечный префикс ρ реалистичен:
в зависимости от темпа выполнения π_2, программа π_1 может выполнить любое число своих действий перед первым действием π_2

Можно сказать, что ρ **несправедливо** по отношению к π_2,
так как не даёт этой программе права выполнить даже одно действие

Точно так же вычисление

$$(s_0, q_0) \to (s_0, q_1) \to (s_0, q_1) \to (s_0, q_1) \to \dots$$

несправедливо по отношению к π_1

Так в моделях появляется понятие **справедливости**

Системы переходов

Для наиболее полного строгого задания справедливости
обобщим понятие модели Крипке,
добавив обозначение выполняемых **действий** на переходы

Размеченная система переходов (с.п.)
над множествами атомарных высказываний AP и действий Act —
это система $TS = (S, S_0, \to, L)$, отличающаяся
от модели Крипке только устройством множества переходов $\to$:

- $\to \subseteq S \times \mathrm{Act} \times S$

С.п. будем называть конечной, если конечны множества S, AP и Act

Переход $(s, \alpha, s') \in \to$ будем также понимать как помеченную дугу $s \xrightarrow{\alpha} s'$

Будем говорить, что при выполнении перехода $s \xrightarrow{\alpha} s'$
выполняется действие α

Записью $\mathrm{Act}(TS, s)$ для с.п. TS и состояния s обозначим
множество действий, которые могут выполниться в с.п. из состояния s:

$$\mathrm{Act}((S, S_0, \to, L), s) = \{\alpha \mid \exists s' : s \xrightarrow{\alpha} s'\}$$

Виды справедливости

Принято рассматривать три вида справедливости:

1. Безусловная справедливость:
   система бесконечно часто выполняет действия множества A

   - Соответствующая несправедливость:
     с некоторого момента действия из A перестают выполняться

   - Пример несправедливости:
     переходы, отвечающие программе π в асинхронной композиции,
     с некоторого момента никогда более не выполняются

Виды справедливости

Принято рассматривать три вида справедливости:

2. Сильная справедливость:
   если система бесконечно часто
   получает возможность выполнить действия множества A,
   то она бесконечно часто выполняет эти действия

   - Соответствующая несправедливость:
     с некоторого момента система
     бесконечно часто может выполнить действия из A,
     но ни разу не выполняет

   - Пример несправедливости:
     с некоторого момента принтер бесконечно часто
     (регулярно время от времени) оказывается свободным,
     но программе ни разу не удаётся его занять

Виды справедливости

Принято рассматривать три вида справедливости:

3. Слабая справедливость:
   если система почти всегда
   имеет возможность выполнить действия множества A,
   то она бесконечно часто выполняет эти действия

   - Соответствующая несправедливость:
     с некоторого момента система
     всегда может выполнить действия из A,
     но ни разу не выполняет

   - Пример несправедливости:
     с некоторого момента принтер постоянно
     (без перерывов, на которые можно было бы всё списать)
     ожидает данные на печать, но никогда их не получает

Справедливость в системах переходов

Рассмотрим бесконечный путь ρ вида

$$s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} s_3 \xrightarrow{\alpha_3} \dots$$

в системе переходов TS

Этот путь для заданного множеств действий A будем называть

- безусловно A-справедливым,
  если действия из A выполняются в ρ бесконечно часто
- сильно A-справедливым, если верно хотя бы одно из двух:
  - число моментов времени i, таких что $\mathrm{Act}(TS, s_i) \cap A \neq \emptyset$, конечно
  - ρ безусловно A-справедлив
- слабо A-справедливым, если верно хотя бы одно из двух:
  - число моментов времени i, таких что $\mathrm{Act}(TS, s_i) \cap A = \emptyset$, бесконечно
  - ρ безусловно A-справедлив

Справедливость в системах переходов

Рассмотрим бесконечный путь ρ вида

$$s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} s_3 \xrightarrow{\alpha_3} \dots$$

в системе переходов TS

Ограничениями справедливости назовём тройку $\mathcal{F} = (\mathcal{F}_u, \mathcal{F}_s, \mathcal{F}_w)$, где $\mathcal{F}_u, \mathcal{F}_s, \mathcal{F}_w \subseteq 2^{\mathsf{Act}}$

Путь ρ будем называть $\mathcal{F}$-справедливым для ограничений справедливости $\mathcal{F} = (\mathcal{F}_u, \mathcal{F}_s, \mathcal{F}_w)$, если он

- безусловно A-справедлив для каждого $A \in \mathcal{F}_u$,
- сильно A-справедлив для каждого $A \in \mathcal{F}_s$ и
- слабо A-справедлив для каждого $A \in \mathcal{F}_w$

Обозначим записями $\Pi_{\mathcal{F}}(TS)$ и $\mathrm{Tr}_{\mathcal{F}}(TS)$ соответственно множество всех $\mathcal{F}$-справедливых вычислений с.п. TS и множество всех трасс таких путей

Справедливость в LTL

Будем говорить, что ltl-формула выполняется на с.п. TS в ограничениях справедливости $\mathcal{F} = (\mathcal{F}_u, \mathcal{F}_s, \mathcal{F}_w)$ ($TS, \mathcal{F} \models \varphi$), если $\mathrm{Tr}_{\mathcal{F}}(TS) \subseteq \mathrm{Tr}(\varphi)$

Пусть возможность выполнить действие из A на следующем переходе отвечает ltl-формуле Φ_A, а выполнение действия A на следующем переходе отвечает формуле Ψ_A

Сопоставим ограничениям $\mathcal{F}$ формулу $\Phi_{\mathcal{F}}$ следующего вида:

$$\Big(\mathop{\&}_{A\in\mathcal{F}_u} \mathbf{GF}\Psi_A\Big) \& \Big(\mathop{\&}_{A\in\mathcal{F}_s} (\mathbf{GF}\Phi_A \to \mathbf{GF}\Psi_A)\Big) \& \Big(\mathop{\&}_{A\in\mathcal{F}_w} (\mathbf{FG}\Phi_A \to \mathbf{GF}\Psi_A)\Big)$$

Утверждение (о справедливости в LTL). Для любых конечной с.п. TS, ограничений справедливости $\mathcal{F}$ и формулы φ верно:

$$TS, \mathcal{F} \models \varphi \Leftrightarrow TS \models \Phi_{\mathcal{F}} \to \varphi$$

Доказательство.

Можете попробовать самостоятельно, вспомнив семантику ltl-формул и утверждения о формулах вида $\mathbf{FG}\psi$ и $\mathbf{GF}\psi$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 14

Автоматный алгоритм model checking для LTL: общая схема

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

С.п. (и, в частности, модель Крипке) очень похожа по устройству и поведению на автомат, и отличий от автомата имеет не очень много:

1. В с.п. нет чтения входных символов,
   но вместо этого есть выполнение действий
   - Модель Крипке — это с.п., в которой действия несущественны
2. В с.п. нет выходных символов,
   но вместо этого есть происхождение событий
3. В с.п. интерес представляют бесконечные пути,
   тогда как в «привычных» автоматах — конечные

Если системы переходов воспринимать как автоматы,
то трассы можно расценивать как бесконечные слова,
а свойства трасс — как языки, распознающиеся такими автоматами

На этих соображениях основывается автоматный алгоритм верификации моделей Крипке относительно LTL
(решения задачи MC-LTL)

Общая схема автоматного алгоритма

Дано: модель Крипке M и ltl-формула φ

Желаемый результат:
«да», если верно соотношение $M \models \varphi$, и «нет» иначе

Общая схема алгоритма:

1. По модели M строится автомат A_M, распознающий $\mathrm{Tr}(M)$
2. По ltl-формуле φ строится автомат $A_{\neg\varphi}$, распознающий $\mathrm{Tr}(\neg\varphi)$
3. Строится пересечение $A_\cap$ автоматов A_M и $A_{\neg\varphi}$:
   автомат, распознающий $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi)$
4. Проверяется пустота автомата $A_\cap$: $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi) \stackrel{?}{=} \emptyset$
5. Выдаётся ответ: «да» $\Leftrightarrow$ автомат $A_\cap$ пуст

Согласно схеме автоматного алгоритма,
для проверки выполнимости ltl-формулы на модели Крипке потребуется:

- Ввести разновидность автоматов,
  способных распознавать языки, состоящие из бесконечных слов

- Научиться строить автоматы, распознающие
  - множество трасс произвольной модели Крипке
  - свойство, задаваемое произвольной ltl-формулой
  - пересечение языков двух произвольных автоматов

- Научиться проверять пустоту произвольного автомата

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 15

Автоматы Бюхи
Обобщённые автомты Бюхи

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

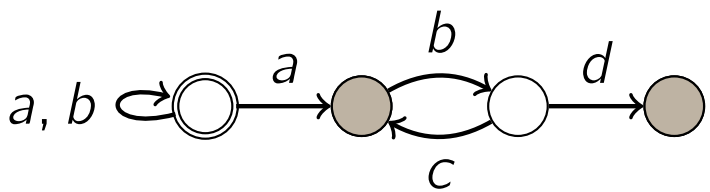
Автоматы Бюхи

Автомат Бю́хи над алфавитом Σ — это система $A = (S, S_0, \to, F)$, где:

- S — конечное множество состояний
- S_0 — множество начальных состояний
- $\to \subseteq S \times \Sigma \times S$ — отношение переходов
  - Изображение семейства переходов $(s, x, s'), (s, y, s'), \ldots$: $s \xrightarrow{x,y,\ldots} s'$
- $F \subseteq S$ — множество допускающих состояний

Как и для моделей Крипке, для автоматов будут применяться терминология и обозначения из теории графов

Пример:



◯ — состояние

◎ — начальное состояние

● — допускающее состояние

Автоматы Бюхи

В записи слов и ω-слов будем опускать разделители:

$$\langle\!\langle x_1, x_2, x_3, \ldots \rangle\!\rangle = \langle\!\langle x_1 x_2 x_3 \ldots \rangle\!\rangle$$

Будем говорить, что бесконечный путь

$$s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} s_3 \xrightarrow{x_3} \ldots$$

в автомате Бюхи порождается ω-словом $x_1 x_2 x_3 \ldots$

Вычисление автомата Бюхи — это бесконечный путь, начинающийся в начальном состоянии

$\mathrm{Tr}(A, w)$ — множество всех вычислений A, порождающихся ω-словом w

$inf(\rho)$ — множество всех состояний, встречающихся бесконечно часто в бесконечном пути ρ

Автоматы Бюхи

Вычисление ρ автомата Бюхи $A = (S, S_0, \to, F)$ успешно (является принимающим), если хотя бы одно допускающее состояние повторяется в нём бесконечно часто

$$(inf(\rho) \cap F \neq \emptyset)$$

ω-слово w принимается автоматом Бюхи A, если A содержит хотя бы одно успешное вычисление, порождаемое этим словом

$$(\exists \rho \in \mathrm{Tr}(A, w) : inf(\rho) \cap F \neq \emptyset)$$

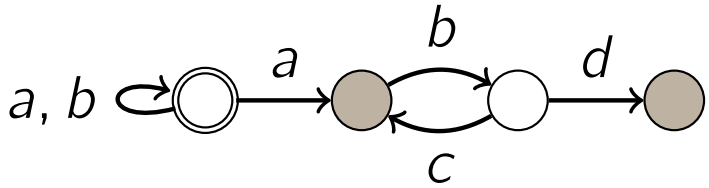
Языком и ω-языком над заданным алфавитом будем называть соответственно множество слов и множество ω-слов

$L(A)$ — так будет обозначаться ω-язык, распознающийся автоматом A (или, более коротко, — ω-язык автомата A):
множество всех ω-слов, принимаемых автоматом A

Автоматы Бюхи

Пример

Автоматом Бюхи A



распознаётся ω-язык

$$L(A) = \{Wabcbcbc \dots bc \cdots \mid W \in \{a, b\}^*\}$$

На некоторых этапах автоматного алгоритма *удобно* и *естественно* будет строить автоматы в *более общего* вида, в котором у автомата может быть более одного допускающего множества

Обобщённые автоматы Бюхи

Обобщённый автомат Бюхи над алфавитом Σ — это система $GA = (S, S_0, \to, \mathcal{F})$, где:

- S, S_0 и $\to$ — такие же множества состояний, начальных состояний и переходов, что и в автомате Бюхи
- $\mathcal{F} \subseteq 2^S$ — семейство допускающих множеств состояний

Все понятия, введённые для автоматов Бюхи, кроме успешного вычисления, дословно переносятся на обобщённые автоматы

Вычисление ρ обобщённого автомата Бюхи GA успешно, если в **каждом** допускающем множестве есть хотя бы одно состояние, повторяющееся в ρ бесконечно часто

$$(\forall F \in \mathcal{F} : inf(\rho) \cap F \neq \emptyset)$$

Формульная запись того, что слово w принимается автоматом $A = (S, S_0, \to, \mathcal{F})$, выглядит так:

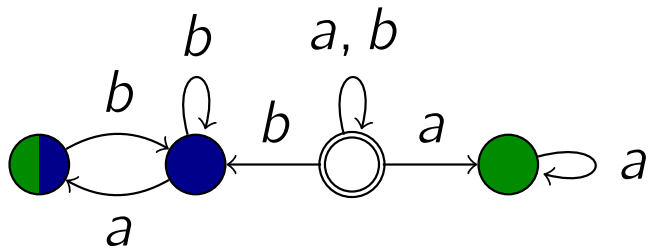
$$\exists \rho \in \mathrm{Tr}(A, w) : \forall F \in \mathcal{F} : inf(\rho) \cap F \neq \emptyset$$

Обобщённые автоматы Бюхи

Далее состояния, принадлежащие одному допускающему множеству, изображаются как окрашенные в один (не белый) цвет

Если вершина принадлежит нескольким принимающим множествам, то она окрашивается во все соответствующие цвета

Пример



Успешное вычисление этого автомата — это вычисление, в котором бесконечно часто повторяются
хотя бы одна синяя вершина и хотя бы одна зелёная вершина

Этим автоматом распознаётся множество всех ω-слов вида WbU, где

- $W \in \{a, b\}^*$ и
- U — любое ω-слово над $\{a, b\}$, содержащее бесконечно много «a», но ни одного «aa»

Обобщённые автоматы Бюхи

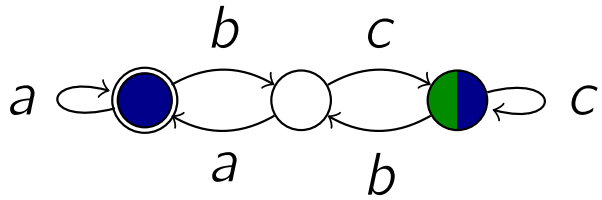
Автомат Бюхи $A = (S', S'_0, \mapsto, F)$ назовём разобобщением обобщённого автомата Бюхи $GA = (S, S_0, \to, \mathcal{F})$, если он устроен так:

- Произвольно упорядочим допускающие множества GA:
$$\mathcal{F} = \{F_0, \ldots, F_{k-1}\}$$
- $S' = S \times \{0, 1, \ldots, k-1\}$
- $S'_0 = S_0 \times \{0\}$
- $F = \{(s, i) \mid s \in F_i,\ i \in \{0, 1, \ldots, k-1\}\}$
- Для каждого перехода $(s \xrightarrow{x} s') \in \to$ и каждого $i \in \{0, 1, \ldots, k-1\}$ в $\mapsto$ включаются следующие переходы (и только такие переходы):
  - если $(s, i) \notin F$, то $(s, i) \overset{x}{\mapsto} (s', i)$
  - иначе $(s, i) \overset{x}{\mapsto} (s', (i+1) \mod k)$

Обобщённые автоматы Бюхи

Пример

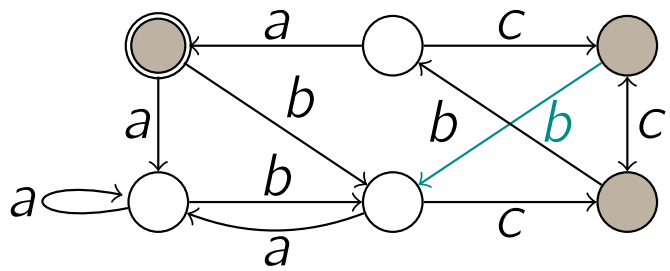
Для обобщённого автомата Бюхи GA



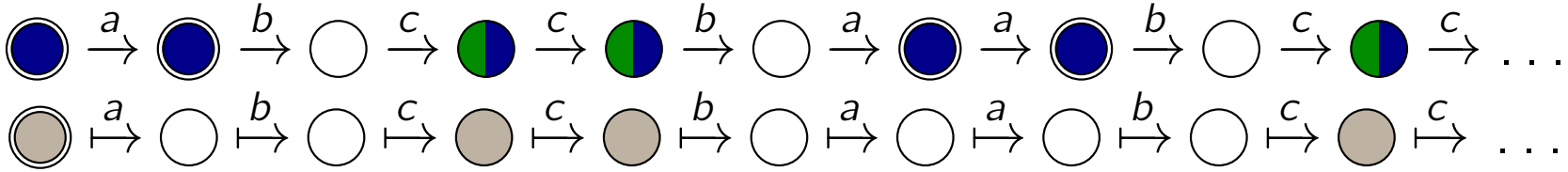
с порядком допускающих множеств (синее, зелёное)
соответствует такое разобобщение A:

$(s,$ 0 $):$

$(s,$ 1 $):$



Пример взаимно соответствующих вычислений этих автоматов:



Теорема о разобобщении автомата Бюхи

Для любого обобщённого автомата Бюхи GA
и любого его разобобщения A **верно** $L(A) = L(GA)$

Доказательство ($L(A) \subseteq L(GA)$)

Для определённости положим, что
$GA = (S, S_0, \to, \{F_0, \ldots, F_{k-1}\})$ и $A = (S', S'_0, \mapsto, F)$

Рассмотрим произвольное ω-слово $w \in L(A)$

В A существует успешное вычисление ρ вида $(s_1, i_1) \overset{w[1]}{\longmapsto} (s_2, i_2) \overset{w[2]}{\longmapsto} \ldots$

По устройству A и успешности ρ, в ρ содержится бесконечная подпоследовательность $((q_0, 0), (q_1, 1), \ldots, (q_n, n \mod k), \ldots) \in F^\omega$

По построению A:

- $\rho_g = (s_1 \overset{w[1]}{\longrightarrow} s_2 \overset{w[2]}{\longrightarrow} \ldots)$ — вычисление GA
- Для всех $i \in \{0, 1, 2, \ldots\}$ верно $q_i \in F_{i \mod k}$

Значит, $\{q_i, q_{i+k}, q_{i+2k}, \ldots\} \subseteq inf(\rho_g) \cap F_i$ для всех $i \in \{0, 1, \ldots, k-1\}$

Следовательно, ρ_g — успешное вычисление GA, и $w \in L(GA)$

Теорема о разобобщении автомата Бюхи

Для любого обобщённого автомата Бюхи GA
и любого его разобобщения A **верно** $L(A) = L(GA)$

Доказательство ($L(A) \supseteq L(GA)$)

$$(GA = (S, S_0, \to, \{F_0, \ldots, F_{k-1}\}),\ A = (S', S'_0, \mapsto, F))$$

Рассмотрим произвольное ω-слово $w \in L(GA)$

В GA существует успешное вычисление ρ_g вида $s_1 \xrightarrow{w[1]} s_2 \xrightarrow{w[2]} \ldots$

По успешности ρ_g, существуют такие индексы $i_0, i_1, i_2, \ldots$:

- i_0 — наименьший, для которого $s_{i_0} \in F_0$
- i_m, $m \geq 1$, — наименьший, для которого $i_m > i_{m-1}$ и $s_{i_m} \in F_{m \bmod k}$

Тогда, по построению, в A содержится успешное вычисление, принимающее w:

$(s_1, 1) \mapsto \cdots \mapsto (s_{i_0}, 0) \mapsto (s_{i_0+1}, 1) \mapsto \cdots \mapsto (s_{i_1}, 1) \mapsto$
$(s_{i_1+1}, 1) \mapsto \cdots \mapsto \cdots \mapsto (s_{i_{k-1}}, k-1) \mapsto (s_{i_{k-1}+1}, 0) \mapsto \ldots\ (s_{i_k}, k) \mapsto \ldots$

Значит, $w \in L(A)$ ▼

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 16

Пересечение автоматов Бюхи

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Общая схема автоматного алгоритма model checking для LTL:

1. По модели Крипке M строится автомат A_M, распознающий $\mathrm{Tr}(M)$
2. По ltl-формуле φ строится автомат $A_{\neg\varphi}$, распознающий $\mathrm{Tr}(\neg\varphi)$
3. Строится пересечение $A_\cap$ автоматов A_M и $A_{\neg\varphi}$:
   автомат, распознающий $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi)$
4. Проверяется пустота автомата $A_\cap$: $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi) \stackrel{?}{=} \emptyset$
5. Выдаётся ответ: «да» $\Leftrightarrow$ автомат $A_\cap$ пуст

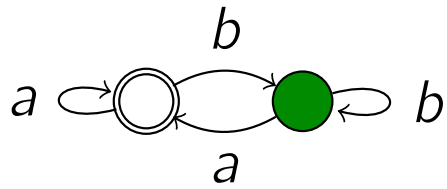
Автомат Бюхи A будем называть
пересечением автоматов Бюхи A_1 и A_2, если $L(A) = L(A_1) \cap L(A_2)$

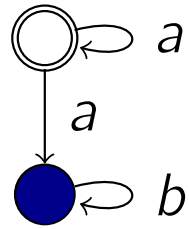
Записью $A' \otimes A''$
для автоматов Бюхи $A' = (S', S'_0, \to, F')$ и $A'' = (S'', S''_0, \mapsto, F'')$
обозначим синхронную композицию этих автоматов,
то есть **обобщённый** автомат Бюхи $(S, S_0, \Rightarrow, \mathcal{F})$ следующего вида:

- $S = S' \times S''$
- $S_0 = S'_0 \times S''_0$
- $\mathcal{F} = \{F' \times S'', S' \times F''\}$
- $(s'_1, s''_1) \overset{x}{\Rightarrow} (s'_2, s''_2) \Leftrightarrow s'_1 \overset{x}{\to} s'_2$ и $s''_1 \overset{x}{\mapsto} s''_2$

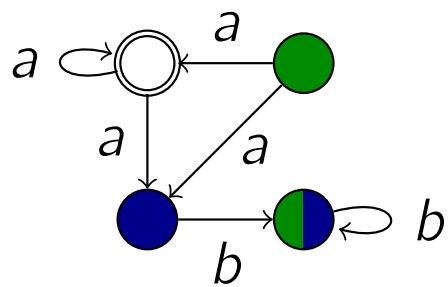
Пример

Синхронной композицией автоматов Бюхи

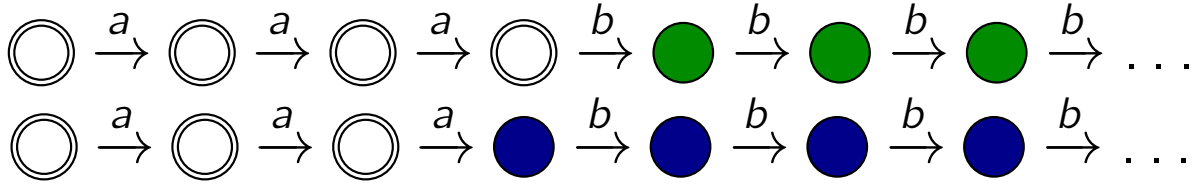




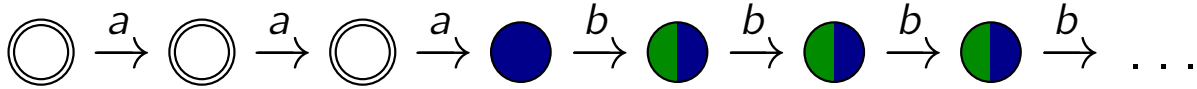
является обобщённый автомат Бюхи



Паре вычислений исходных автоматов



взаимно соответствует вычисление композиции



Теорема о пересечении автоматов Бюхи

Для любой пары автоматов Бюхи A', A''
разобобщение автомата $A' \otimes A''$ является их пересечением
Доказательство

По теореме о разобобщении автомата Бюхи,
достаточно показать, что верно $L(A' \otimes A'') = L(A') \cap L(A'')$

Пусть, для определённости,

- $A' = (S', S'_0, \to, F')$
- $A'' = (S'', S''_0, \mapsto, F'')$
- $A' \otimes A'' = (S, S_0, \Rightarrow, \{F_1, F_2\})$

Теорема о пересечении автоматов Бюхи

Для любой пары автоматов Бюхи A', A''
разобобщение автомата $A' \otimes A''$ является их пересечением

Доказательство ($L(A' \otimes A'') \subseteq L(A') \cap L(A'')$)

$(A' = (S', S'_0, \to, F'),\ A'' = (S'', S''_0, \mapsto, F''),\ A' \otimes A'' = (S, S_0, \Rightarrow, \{F_1, F_2\}))$

Рассмотрим произвольное ω-слово $w \in L(A' \otimes A'')$

В $A' \otimes A''$ существует успешное вычисление ρ вида

$$(s'_1, s''_1) \overset{w[1]}{\Longrightarrow} (s'_2, s''_2) \overset{w[2]}{\Longrightarrow} \dots$$

По заданию автомата $A' \otimes A''$, верно следующее:

- $\rho' = (s'_1 \xrightarrow{w[1]} s'_2 \xrightarrow{w[2]} \dots)$ — вычисление автомата A'
  - Так как $inf(\rho) \cap (F' \times S'') \neq \emptyset$, то и $inf(\rho') \cap F' \neq \emptyset$
  - Значит, вычисление ρ' успешно
- $\rho'' = (s''_1 \overset{w[1]}{\longmapsto} s''_2 \overset{w[2]}{\longmapsto} \dots)$ — вычисление автомата A''
  - Аналогично, вычисление ρ'' успешно

Значит, $w \in L(A')$ и $w \in L(A'')$, то есть $w \in L(A') \cap L(A'')$

Теорема о пересечении автоматов Бюхи

Для любой пары автоматов Бюхи A', A''
разобобщение автомата $A' \otimes A''$ является их пересечением

Доказательство ($L(A' \otimes A'') \supseteq L(A') \cap L(A'')$)

($A' = (S', S'_0, \to, F')$, $A'' = (S'', S''_0, \mapsto, F'')$, $A' \otimes A'' = (S, S_0, \Rightarrow, \{F_1, F_2\})$)

Рассмотрим ω-слово $w \in L(A') \cap L(A'')$

Тогда существуют успешные вычисления ρ', ρ'' соответственно вида

$$(s'_1 \xrightarrow{w[1]} s'_2 \xrightarrow{w[2]} \dots) \quad \text{и} \quad (s''_1 \xmapsto{w[1]} s''_2 \xmapsto{w[2]} \dots)$$

По заданию автомата $A' \otimes A''$, верно следующее:

- $\rho = ((s'_1, s''_1) \xRightarrow{w[1]} (s'_2, s''_2) \xRightarrow{w[2]} \dots)$ — вычисление $A' \otimes A''$
- Так как $inf(\rho') \cap F' \neq \emptyset$, то и $inf(\rho) \cap (F' \times S'') \neq \emptyset$
- Аналогично, $inf(\rho) \cap (S' \times F'') \neq \emptyset$

Значит, вычисление ρ успешно, и $w \in L(A' \otimes A'')$ ▼

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 17

Проверка пустоты автомата Бюхи

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Общая схема автоматного алгоритма model checking для LTL:

1. По модели Крипке M строится автомат A_M, распознающий $\mathrm{Tr}(M)$
2. По ltl-формуле φ строится автомат $A_{\neg\varphi}$, распознающий $\mathrm{Tr}(\neg\varphi)$
3. Строится пересечение $A_\cap$ автоматов A_M и $A_{\neg\varphi}$: автомат, распознающий $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi)$
4. Проверяется пустота автомата $A_\cap$: $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi) \stackrel{?}{=} \emptyset$
5. Выдаётся ответ: «да» $\Leftrightarrow$ автомат $A_\cap$ пуст

Автомат Бюхи A пуст, если им распознаётся пустой язык,
и непуст иначе

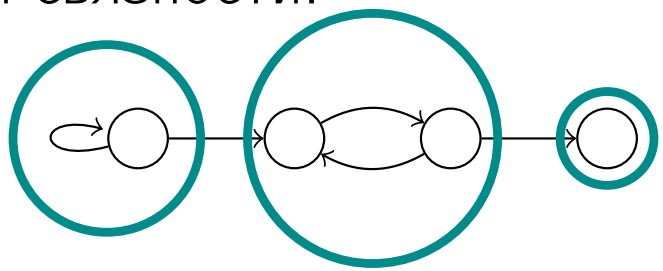
Для лучшего понимания теоремы,
сводящей проверку пустоты автомата Бюхи к графовым задачам,
полезно напомнить/ввести соответствующие понятия из теории графов

Вершина u ориентированного графа достижима из вершины v,
если в этом графе существует путь из v в u
(быть может, тривиальный, если $u = v$)

Ориентированный граф называется сильно связным,
если любые две его вершины достижимы друг из друга

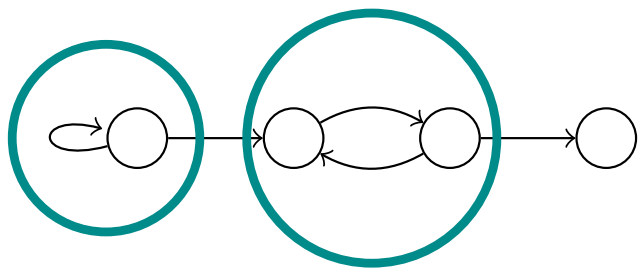
Компонента сильной связности (к.с.с.) ориентированного графа — это максимальный по включению вершин и дуг сильно связный подграф этого графа

Например, ниже в графе обведены окружностями все компоненты сильной связности:



Компонента сильной связности нетривиальна (н.к.с.с.), если в ней содержится хотя бы одна дуга

Например, ниже в графе обведены окружностями все нетривиальные компоненты сильной связности:



Теорема (о проверке пустоты автомата Бюхи)
Для любого автомата Бюхи A верно следующее: $L(A) \neq \emptyset \Leftrightarrow$ в A хотя бы из одного начального состояния достижима хотя бы одна н.к.с.с., содержащая хотя бы одно допускающее состояние

Доказательство

($\Leftarrow$) Если в A из начального состояния по пути $s_1 \to \cdots \to s_k$ достижима н.к.с.с. с путём $s_k \to \pi \to s_k$ через допускающее состояние, то $s_1 \to \cdots \to s_k(\to \pi \to s_k)^\infty$ — успешное вычисление A

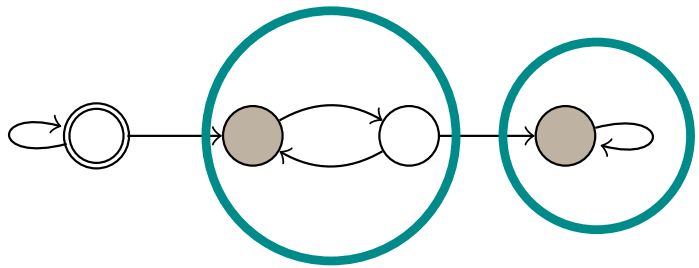
($\Rightarrow$) Если $L(A) \neq \emptyset$, то в A существует успешное вычисление ρ

По бесконечности и успешности, ρ содержит префикс вида $s_1 \to \cdots \to s_k \to \pi \to s_k$ с допускающим состоянием в подпути $\pi \to s_k$

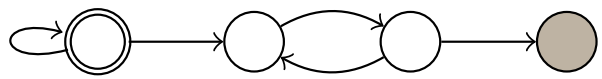
Тогда состояния подпути $\pi \to s_k$ входят в искомую н.к.с.с. ▼

Примеры

Следующий автомат Бюхи непуст
(н.к.с.с. с допускающим состоянием обведены кругами):



Следующий автомат Бюхи пуст
(не содержит н.к.с.с. с допускающим состоянием):



Поиск н.к.с.с. в ориентированном графе — это известная задача,
для которой известны эффективные решающие алгоритмы,
выходящие за рамки курса: «лобовой» с транзитивным замыканием,
Косарайю, Тарьяна, стековый, ...

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 18

Автоматы Бюхи
для моделей Крипке

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

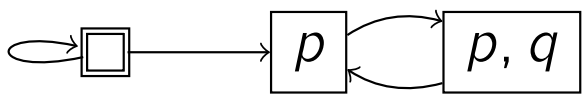
Общая схема автоматного алгоритма model checking для LTL:

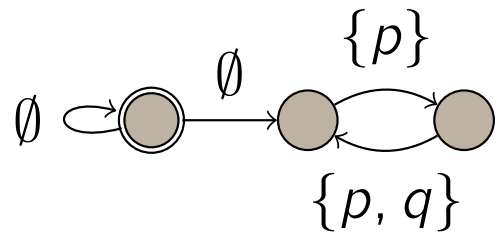
1. По модели Крипке M строится автомат A_M, распознающий $\mathrm{Tr}(M)$
2. По ltl-формуле φ строится автомат $A_{\neg\varphi}$, распознающий $\mathrm{Tr}(\neg\varphi)$
3. Строится пересечение $A_\cap$ автоматов A_M и $A_{\neg\varphi}$: автомат, распознающий $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi)$
4. Проверяется пустота автомата $A_\cap$: $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi) \stackrel{?}{=} \emptyset$
5. Выдаётся ответ: «да» $\Leftrightarrow$ автомат $A_\cap$ пуст

Для конечной модели Крипке $M = (S, S_0, \to, L)$ над AP зададим моделирующий её автомат Бюхи $A_M = (S', S'_0, \mapsto, F)$ над 2^{AP} так:

- $S' = F = S$
- $S'_0 = S_0$
- $s_1 \overset{x}{\mapsto} s_2 \Leftrightarrow s_1 \to s_2$ и $L(s_1) = x$

Пример (слева — модель Крипке M, справа — автомат A_M):





Теорема (о трансляции модели Крипке в автомат Бюхи)
Для любой конечной модели Крипке M **верно** $\mathrm{Tr}(M) = L(A_M)$

Доказательство. Очевидно?
(*Трасса вычисления M является словом, порождающим то же вычисление A_M, и наоборот; и все вычисления A_M успешны*)

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 19

Автоматы Бюхи
для ltl-формул

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

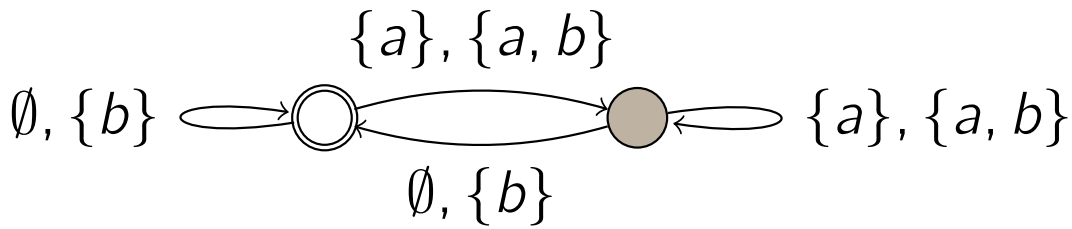
Общая схема автоматного алгоритма model checking для LTL:

1. По модели Крипке M строится автомат A_M, распознающий $\mathrm{Tr}(M)$
2. По ltl-формуле φ строится автомат $A_{\neg\varphi}$, распознающий $\mathrm{Tr}(\neg\varphi)$
3. Строится пересечение $A_\cap$ автоматов A_M и $A_{\neg\varphi}$: автомат, распознающий $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi)$
4. Проверяется пустота автомата $A_\cap$: $\mathrm{Tr}(M) \cap \mathrm{Tr}(\neg\varphi) \stackrel{?}{=} \emptyset$
5. Выдаётся ответ: «да» $\Leftrightarrow$ автомат $A_\cap$ пуст

Начнём с примеров ($\mathrm{AP} = \{a, b\}$)

$\varphi = \mathbf{GF}a$

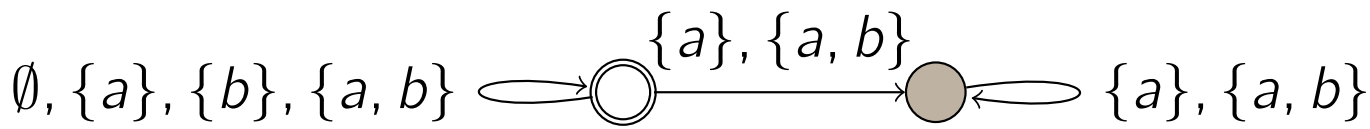
$A_\varphi = ?$



Легко видеть, что $L(A_\varphi) = \mathrm{Tr}(\varphi)$

$\psi = \mathbf{FG}a$

$A_\psi = ?$

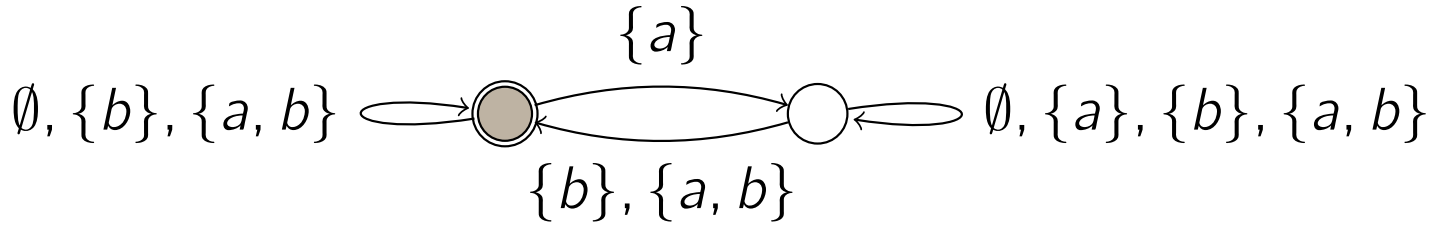


Легко видеть, что $L(A_\psi) = \mathrm{Tr}(\psi)$

Начнём с примеров ($\mathrm{AP} = \{a, b\}$)

$\varphi = \mathbf{G}(a \to \mathbf{F}b)$

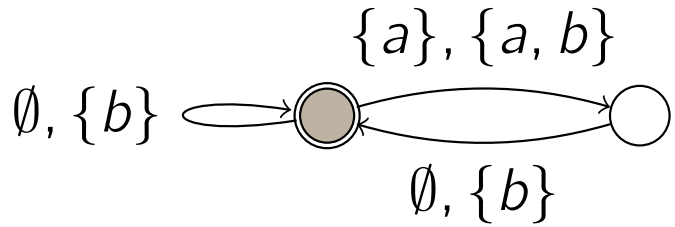
$A_\varphi = ?$



Легко видеть, что $L(A_\varphi) = \mathrm{Tr}(\varphi)$

$\psi = \mathbf{G}(a \to \mathbf{X}\neg a)$

$A_\psi = ?$



Легко видеть, что $L(A_\psi) = \mathrm{Tr}(\psi)$

А как быть с произвольной ltl-формулой?

Опишем способ построения таких автоматов без доказательства

Дано: произвольная ltl-формула φ

Требуется: построить (конструктивно задать) автомат Бюхи A_φ, для которого верно $L(A_\varphi) = \mathrm{Tr}(\varphi)$

Согласно теореме о разобобщении автомата Бюхи, достаточно показать, как построить **обобщённый** автомат Бюхи GA_φ, такой что $L(GA_\varphi) = \mathrm{Tr}(\varphi)$

Без ограничения общности можно полагать, что φ — формула без двойных отрицаний: не содержит подформул $\neg\neg\psi$ (т.к. $\neg\neg\psi \equiv \psi$)

Формулы вида $\neg\psi$ далее будем называть негативными, а остальные — позитивными

Замыкание Фишера-Ладнера $[\varphi]_{fl}$ формулы φ — это множество формул, содержащее следующие формулы и только их:

1. Все позитивные подформулы формулы φ
2. Для каждой подформулы вида $\psi\mathbf{U}\chi$ формулы φ — формулу $\mathbf{X}(\psi\mathbf{U}\chi)$

Например, $[\neg(p\mathbf{U}\neg q)]_{fl} = \{p, q, p\mathbf{U}\neg q, \mathbf{X}(p\mathbf{U}\neg q)\}$

Гипотезой (для формулы φ) назовём множество формул вида

$$F \cup \{\neg\psi \mid \psi \in [\varphi]_{fl} \setminus F\},$$

где $F \subseteq [\varphi]_{fl}$

Например, $\{\neg p, \neg q, p\mathbf{U}\neg q, \neg\mathbf{X}(p\mathbf{U}\neg q)\}$ — гипотеза для $\neg(p\mathbf{U}\neg q)$

Гипотезу H объявим совместной, если
для любых формул вида $\psi_1 \,\&\, \psi_2$ и $\chi_1\mathbf{U}\chi_2$ из $[\varphi]_{fl}$ верно:

- $\psi_1 \,\&\, \psi_2 \in H \Leftrightarrow \{\psi_1, \psi_2\} \subseteq H$
- $\chi_1\mathbf{U}\chi_2 \in H \Leftrightarrow \chi_2 \in H$ или $\{\chi_1, \mathbf{X}(\chi_1\mathbf{U}\chi_2)\} \subseteq H$

Например, гипотеза $\{p, \neg q, p\mathbf{U}\neg q, \mathbf{X}(p\mathbf{U}\neg q)\}$ совместна,
а $\{p, \neg q, \neg(p\mathbf{U}\neg q), \mathbf{X}(p\mathbf{U}\neg q)\}$ — нет

Состояниями автомата GA_φ объявим всевозможные совместные гипотезы для φ

Начальными состояниями автомата GA_φ объявим все вершины, в которых содержится φ

Гипотезы H_1 и H_2 назовём локально согласованными, если для любой формулы вида $\mathbf{X}\psi$ из $[\varphi]_{fl}$ верно

$$\mathbf{X}\psi \in H_1 \Leftrightarrow \psi \in H_2$$

В множество переходов автомата GA_φ

включим те и только те переходы $H_1 \xrightarrow{X} H_2$, для которых $X = H_1 \cap \mathrm{AP}$ и пара гипотез H_1, H_2 локально согласованна

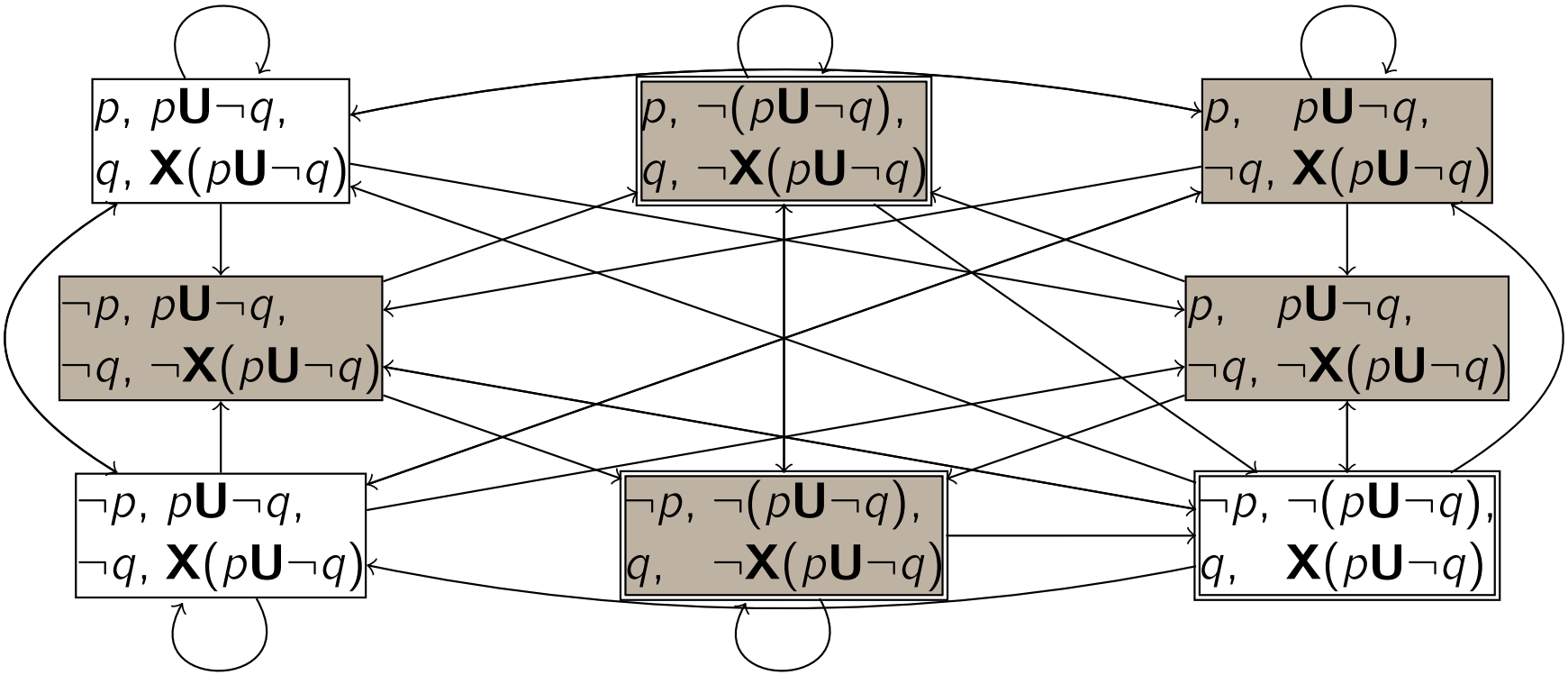
Будем говорить, что гипотеза H завершает формулу вида $\psi\mathbf{U}\chi$ из $[\varphi]_{fl}$, если верно хотя бы одно из двух:

1. $\chi \in H$
2. $\mathbf{X}(\psi\mathbf{U}\chi) \notin H$

Произвольно упорядочим все подформулы вида $\psi_1\mathbf{U}\psi_2$ из $[\varphi]_{fl}$:
$\chi_1, \chi_2, \ldots, \chi_k$ — и добавим в GA_φ
допускающие множества $F_1, \ldots, F_k$:
$H \in F_i \Leftrightarrow$ гипотеза H завершает формулу χ_i

Пример

Обобщённый автомат Бюхи $GA_{\neg(p\mathbf{U}\neg q)}$ может быть устроен так:



(*Метки дуг опущены: дуга, исходящая из H, помечена событием $H \cap \mathrm{AP}$*)

Можете попробовать самостоятельно доказать, что автомат, устроенный **согласно полужирному зелёному тексту**, действительно распознаёт свойство формулы (и это непросто!)

Математические методы верификации схем и программ

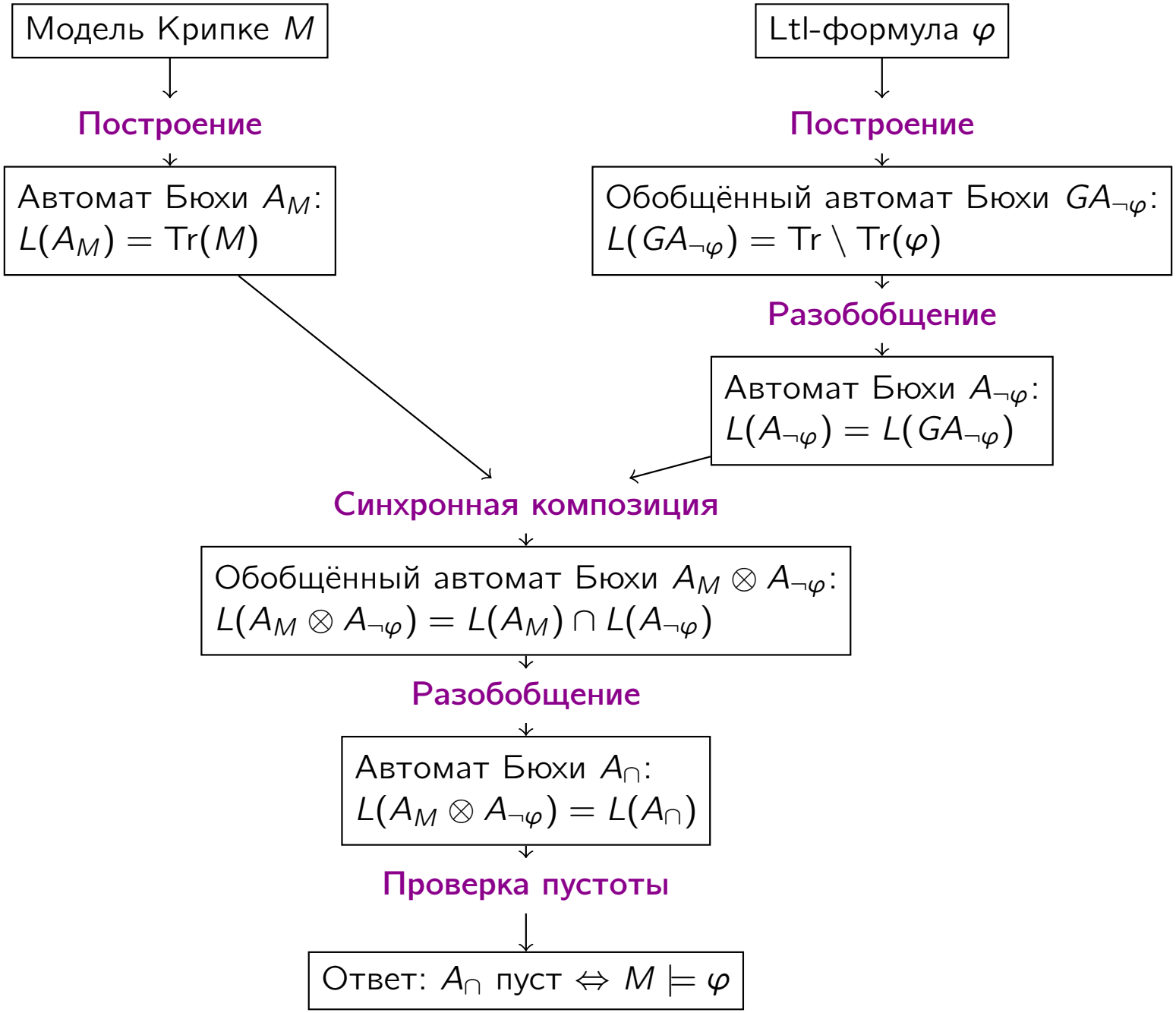
mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 20

Автоматный алгоритм
model checking для LTL:
уточнённая схема

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр


Модель Крипке M
Ltl-формула φ
Построение
Построение
Автомат Бюхи A_M:
$L(A_M) = \mathrm{Tr}(M)$
Обобщённый автомат Бюхи $GA_{\neg\varphi}$:
$L(GA_{\neg\varphi}) = \mathrm{Tr} \setminus \mathrm{Tr}(\varphi)$
Разобобщение
Автомат Бюхи $A_{\neg\varphi}$:
$L(A_{\neg\varphi}) = L(GA_{\neg\varphi})$
Синхронная композиция
Обобщённый автомат Бюхи $A_M \otimes A_{\neg\varphi}$:
$L(A_M \otimes A_{\neg\varphi}) = L(A_M) \cap L(A_{\neg\varphi})$
Разобобщение
Автомат Бюхи $A_\cap$:
$L(A_M \otimes A_{\neg\varphi}) = L(A_\cap)$
Проверка пустоты
Ответ: $A_\cap$ пуст $\Leftrightarrow$ $M \models \varphi$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 21

Логика деревьев вычислений (CTL)

Постановка задачи верификации моделей Крипке относительно CTL

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

LTL — это несложно устроенный язык формальной спецификации моделей Крипке

Запись $M \models \varphi$ для модели Крипке M и ltl-формулы φ читается так: **Каждое** вычисление модели M удовлетворяет свойству формулы φ

Естественно возникает желание уметь проверять, **существует** ли вычисление модели, удовлетворяющее свойству φ

Это можно попробовать явно отразить в формуле и условно изобразить так (*и это уже не LTL*):

- Каждое вычисление ...: $M \models \forall\varphi$
- Существует вычисление ...: $M \models \exists\varphi$

Вступление

$$M \models \forall\varphi \qquad\qquad M \models \exists\varphi$$

Желание поставить квантор относительно вычислений может возникнуть и «внутри» формулы — например:
Для любого начала вычисления **существует** способ его продолжить до правильного

Далее рассматривается логика деревьев вычислений (Computation Tree Logic, CTL; она же логика ветвящегося времени),

- похожая на LTL, но
- содержащая такие кванторы, как выше, и
- «двойственная» к LTL в том смысле, что
  - формулами LTL задаются свойства вычислений моделей Крипке,
  - а формулами CTL — свойства состояний

Логика деревьев вычислений: синтаксис

Формулы логики деревьев вычислений делятся на две категории:

- Формулы состояния: их истинностное значение задаётся **состоянием** модели Крипке
- Формулы пути: их истинностное значение задаётся **бесконечным путём** в модели Крипке

Краткий синтаксис этих формул над множеством атомарных высказываний AP:

$$\begin{aligned} \Phi &::= \mathbb{t} \mid p \mid (\Phi \,\&\, \Phi) \mid (\neg\Phi) \mid (\mathbf{A}\varphi) \mid (\mathbf{E}\varphi), \\ \varphi &::= (\mathbf{X}\Phi) \mid (\Phi \mathbf{U} \Phi), \end{aligned}$$

где Φ — формула состояния (её же будем называть ctl-формулой), φ — формула пути и $p \in \mathrm{AP}$

Логика деревьев вычислений: синтаксис

По сравнению с LTL в языке появились две новых буквы (кванторы пути):

- **A**φ: любой бесконечный путь, исходящий из текущего состояния, обладает свойством φ
- **E**φ: существует бесконечный путь, исходящий из текущего состояния и обладающий свойством φ

Остальные операции имеют тот же содержательный смысл, что и в LTL

В полный синтаксис включим те же операции, что и для LTL ($\vee$, $\to$, **F**, **G**) с тем же содержательным смыслом и способом введения, кроме способа введения **G** (из-за ограничений синтаксиса):

- **AG**$\Phi = \neg$**EF**$\neg\Phi$
- **EG**$\Phi = \neg$**AF**$\neg\Phi$

Приоритеты операций **A** и **E** одинаковы и такие же, как и $\neg$ и **X**, а в остальном — как в LTL

Логика деревьев вычислений: примеры

Примеры ctl-формул и выражаемых ими свойств вычислительных систем:

- Цель может быть достигнута
$$\mathbf{EF}\mathit{goal}$$
- Как бы ни работал компьютер, есть возможность в дальнейшем его выключить
$$\mathbf{AGEF}\mathit{off}$$
- Тех, кто много грешит, неотвратимо настигнет кара
$$\mathbf{AG}(\mathit{too\_many\_sins} \rightarrow \mathbf{AF}\mathit{punishment})$$
- Если я захочу всё бросить, то смогу сделать это на следующий день
$$\mathbf{AG}(\mathit{want} \rightarrow \mathbf{EX}\mathit{quit})$$
- Если я провинюсь, то меня обязательно накажут на следующий день
$$\mathbf{AG}(\mathit{guilty} \rightarrow \mathbf{AX}\mathit{punisment})$$

Логика деревьев вычислений: семантика

Отношение выполнимости ctl-формулы Φ в состоянии s модели Крипке $M = (S, S_0, \to, L)$ ($M, s \models \Phi$) и формулы пути φ на бесконечном пути π модели M ($M, \pi \models \varphi$) определяются так:

- Соотношение $M, s \models \mathbb{t}$ верно всегда
- $M, s \models p$, где $p \in \mathrm{AP} \Leftrightarrow p \in L(s)$
- $M, s \models \Phi_1 \,\&\, \Phi_2 \Leftrightarrow M, s \models \Phi_1$ и $M, s \models \Phi_2$
- $M, s \models \neg\Phi \Leftrightarrow M, s \not\models \Phi$
- $M, s \models \mathbf{A}\varphi \Leftrightarrow$ для любого бесконечного пути π в M, исходящего из s, верно $M, \pi \models \varphi$
- $M, s \models \mathbf{E}\varphi \Leftrightarrow$ в M существует бесконечный путь π, исходящий из s и такой что $M, \pi \models \varphi$
- $M, \pi \models \mathbf{X}\Phi \Leftrightarrow M, \pi[2] \models \Phi$
- $M, \pi \models \Phi_1 \mathbf{U} \Phi_2 \Leftrightarrow$ существует момент времени k, такой что
  - $M, \pi[k] \models \Phi_2$ и
  - для любого момента времени i, такого что $i < k$, верно $M, \pi[i] \models \Phi_1$

Логика деревьев вычислений: основные свойства

Утверждение. **Для любых модели Крипке M, её бесконечного пути π и ctl-формулы Φ верно:**
$M, \pi \models \mathbf{F}\Phi \Leftrightarrow$ **в π содержится состояние s, для которого верно** $M, s \models \Phi$

Утверждение. **Для любых модели Крипке M, её бесконечного пути π и ctl-формулы Φ верно:**
$M, \pi \models \mathbf{G}\Phi \Leftrightarrow$
для любого состояния s пути π верно $M, s \models \Phi$

Эти два утверждения обосновывать не будем ввиду их простоты

Утверждение
Для любых модели Крипке M, состояния s и ctl-формулы Φ верно:
$M, s \models \mathbf{AGAF}\Phi \Leftrightarrow$ **для любого бесконечного пути π в M, начинающегося в s, существует бесконечно много попарно различных моментов времени i, таких что** $M, \pi[i] \models \Phi$

Доказательство. Аналогично утверждению про $\mathbf{GF}\varphi$ для LTL

Логика деревьев вычислений: основные свойства

Развёрткой модели Крипке $M = (S, S_0, \to, L)$ относительно состояния $\mathfrak{s}$ называется бесконечное ориентированное дерево следующего вида

Дерево разбито на ярусы, пронумерованные моментами времени, и дуги из i-го яруса ведут только в $(i+1)$-й

Каждая вершина дерева помечена состоянием модели

0-й ярус состоит из одной вершины — корня, помеченного состоянием $\mathfrak{s}$

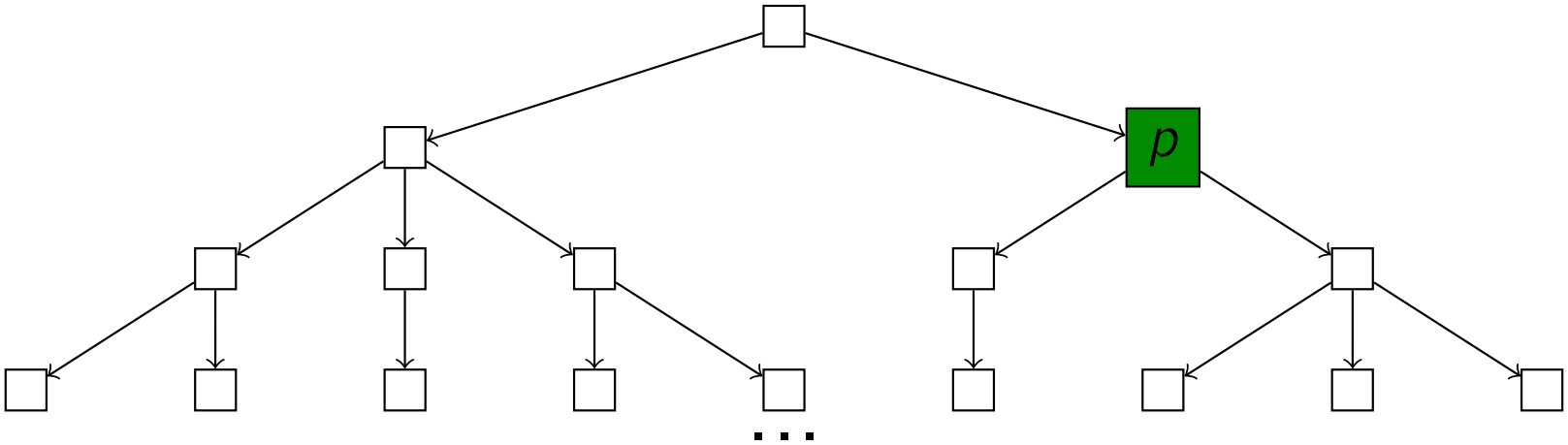
Если вершина v i-го яруса помечена состоянием s и $s \to s'$, то в развёртке из v исходит дуга в вершину, помеченную s'

В расширенном синтаксисе ctl-формул содержится 8 темпоральных комбинаций QO квантора пути Q и темпорального оператора O: **AX**, **EX**, **AF**, **EF**, **AG**, **EG**, **AU** и **EU**

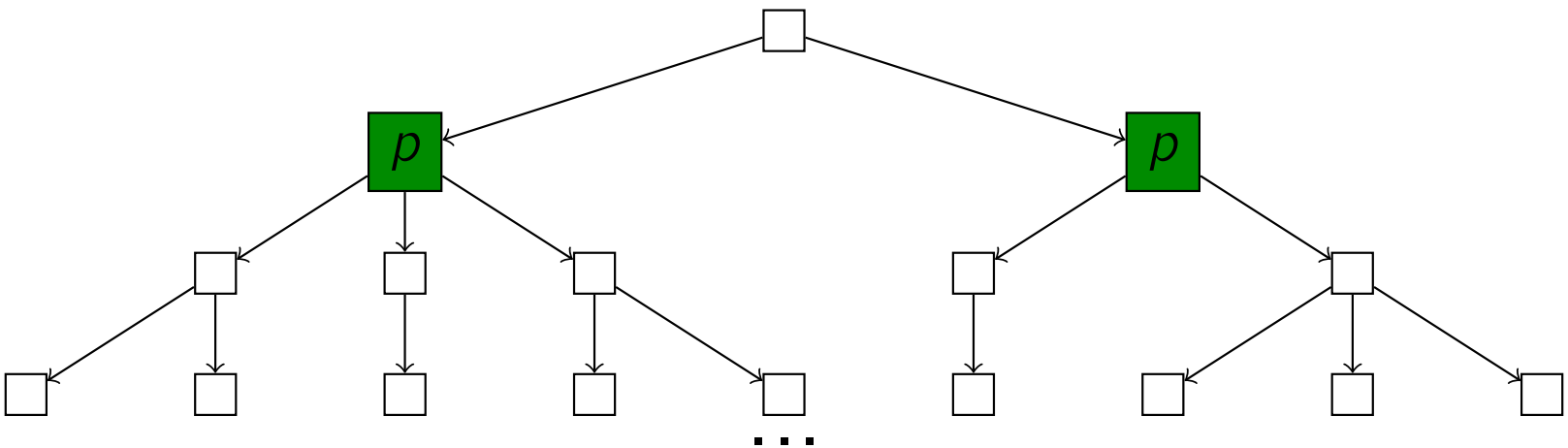
Можно проиллюстрировать эти сочетания на развёртке следующим образом

Логика деревьев вычислений: основные свойства

EXp

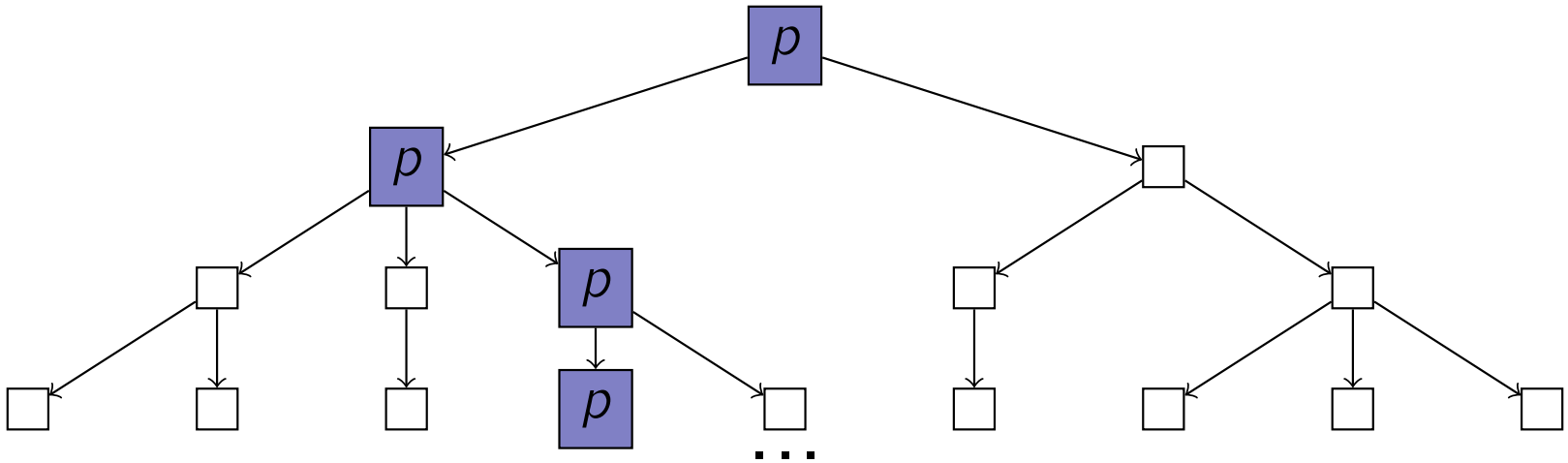


AXp

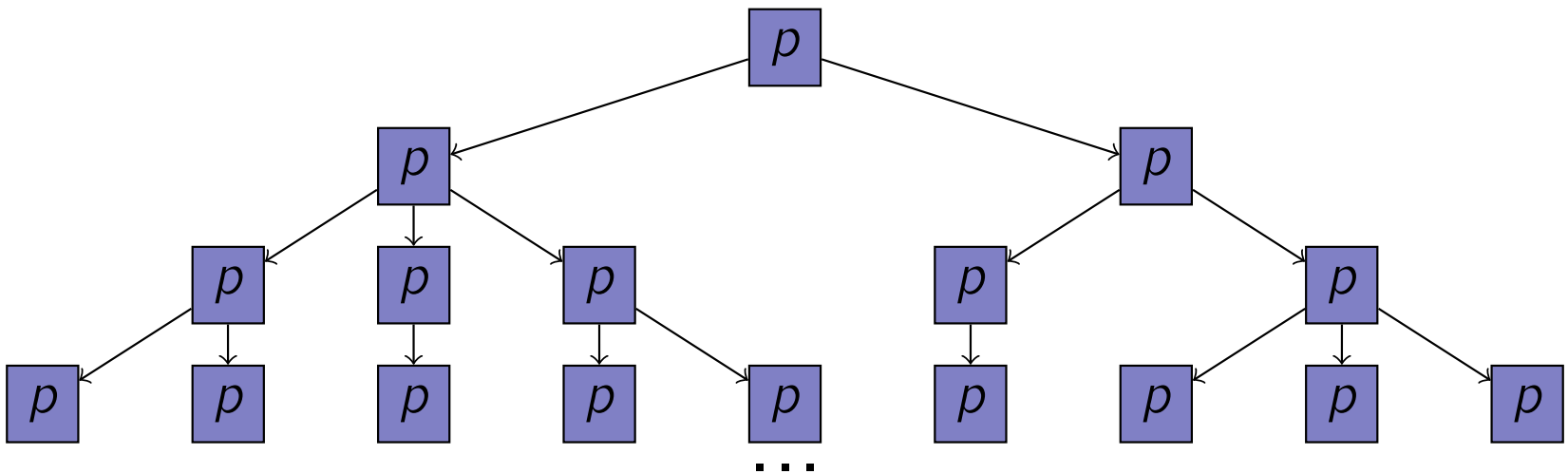


Логика деревьев вычислений: основные свойства

EGp

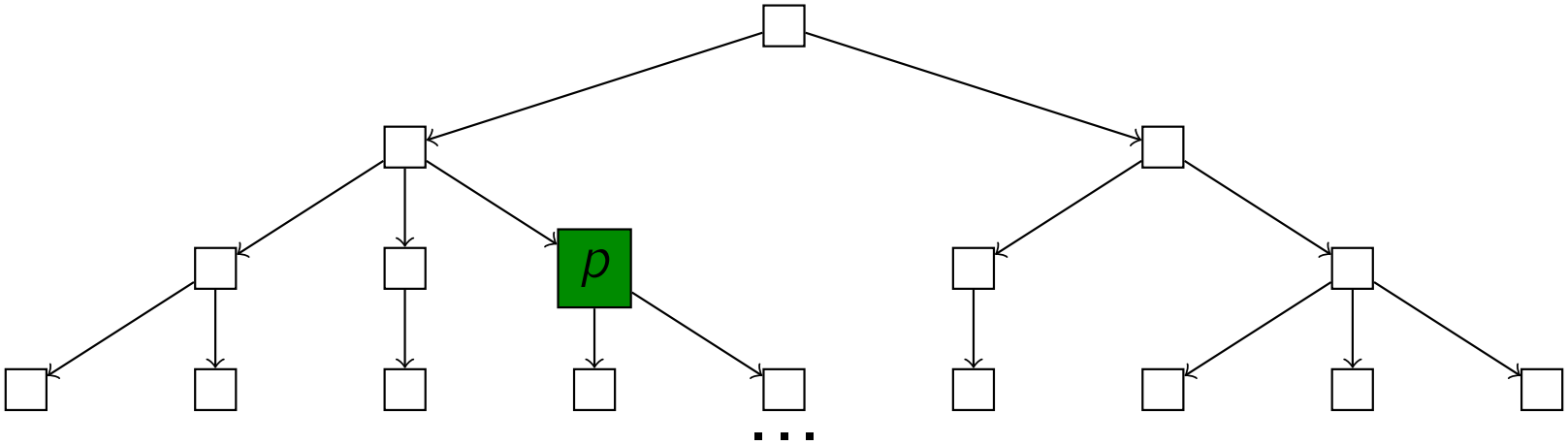


AGp

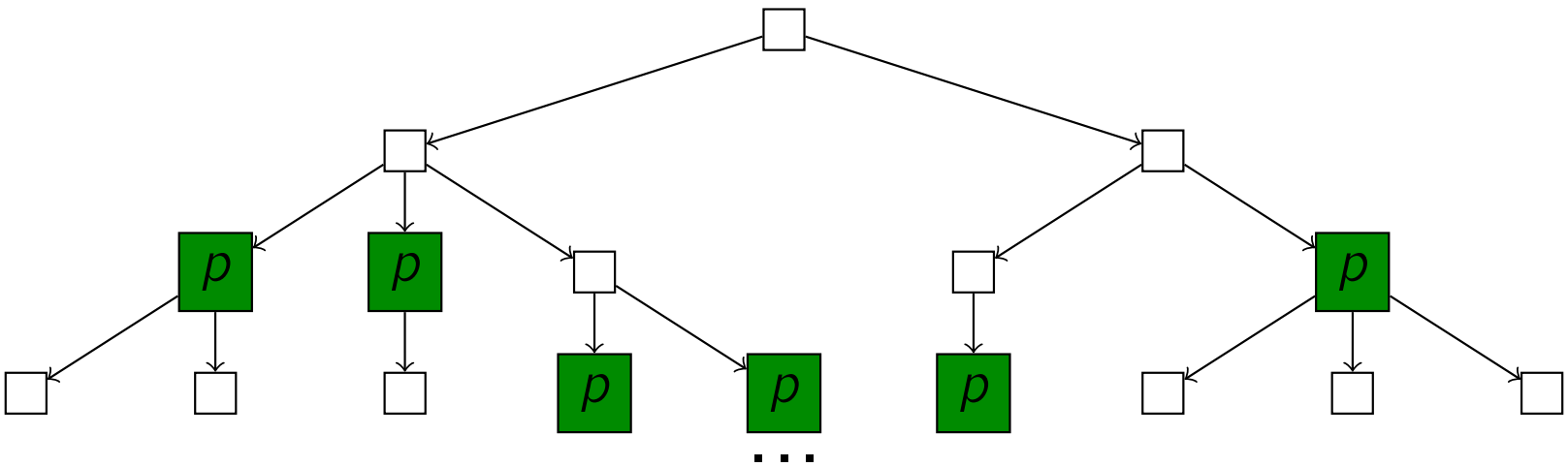


Логика деревьев вычислений: основные свойства

EFp

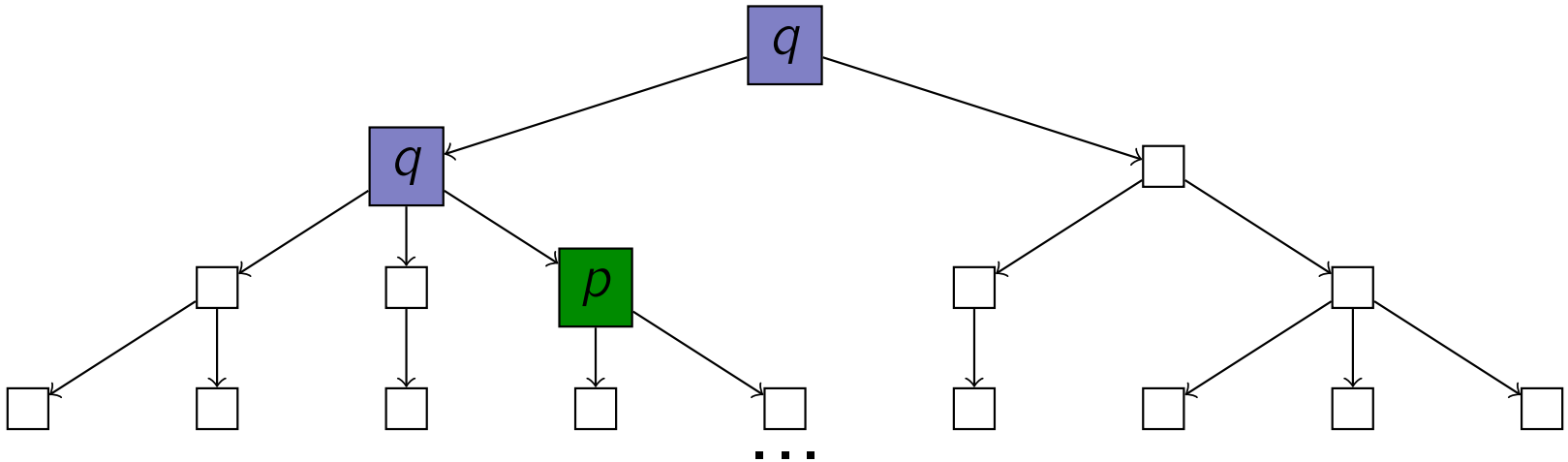


AFp

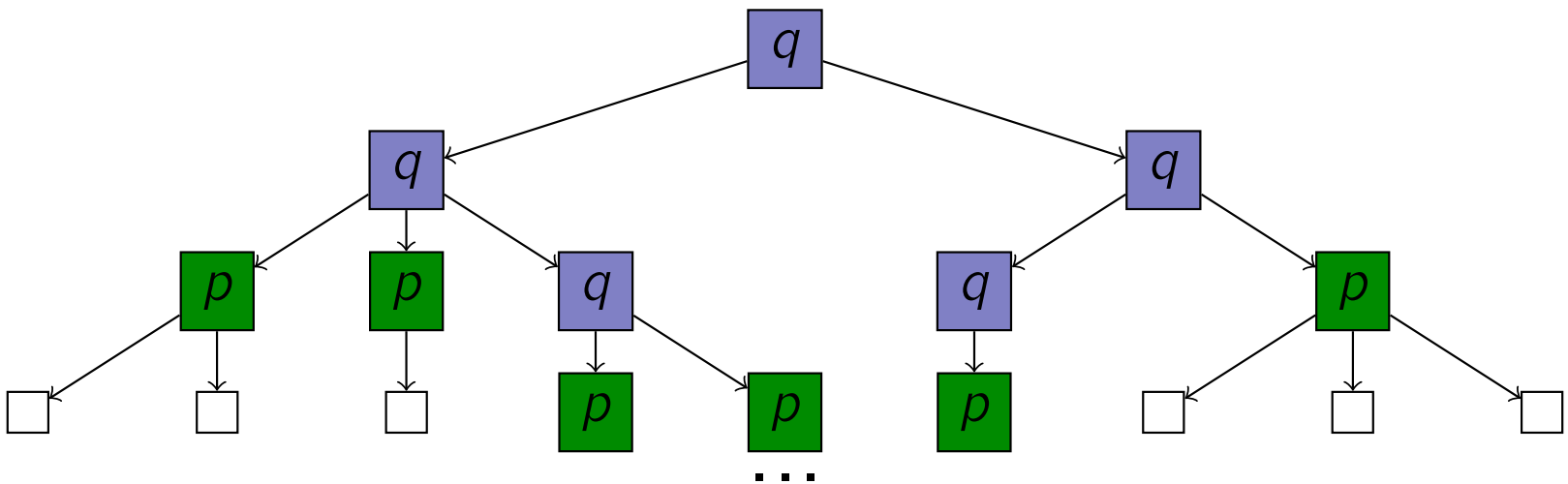


Логика деревьев вычислений: основные свойства

$\mathbf{E}(q\mathbf{U}p)$



$\mathbf{A}(q\mathbf{U}p)$



Логика деревьев вычислений: основные свойства

Будем говорить, что ctl-формулы Φ и Ψ равносильны ($\Phi \sim \Psi$), если для любой модели Крипке M и любого состояния s справедлива равносильность

$$M, s \models \Phi \Leftrightarrow M, s \models \Psi$$

Согласно краткому и полному синтаксисам, справедливы следующие равносильности, позволяющие *выразить* темпоральные комбинации **AF**, **EF**, **AG** и **EG** через комбинации краткого синтаксиса:

- $\mathbf{AF}\Phi \sim \mathbf{A}(\mathbb{t}\mathbf{U}\Phi)$
- $\mathbf{EF}\Phi \sim \mathbf{E}(\mathbb{t}\mathbf{U}\Phi)$
- $\mathbf{AG}\Phi \sim \neg\mathbf{EF}\neg\Phi$
- $\mathbf{EG}\Phi \sim \neg\mathbf{AF}\neg\Phi$

Кроме того, несложно убедиться в такой равносильности:

Утверждение. $\mathbf{AX}\Phi \sim \neg\mathbf{EX}\neg\Phi$

Значит, при составлении формул можно обойтись тремя комбинациями:

EX, **AU** и **EU**

Логика деревьев вычислений: основные свойства

EX, **AU** и **EU**

Такой набор комбинаций: достаточно маленький (*в идеале — минимальный*), через который выражаются остальные комбинации, — принято называть темпоральным базисом CTL

Утверждение. $\mathbf{A}(\Phi\mathbf{U}\Psi) \sim \neg\mathbf{E}(\neg\Psi\mathbf{U}(\neg\Phi \,\&\, \neg\Psi)) \,\&\, \neg\mathbf{EG}\neg\Psi$

Доказательство. Можете попробовать самостоятельно

Это значит, что базисом CTL является и такой набор:

EX, **EG** и **EU**

В теоретическом анализе CTL (доказательствах), как правило, будет использоваться один из двух упомянутых базисов

Задача model checking относительно CTL

Для модели Крипке M и ctl-формулы Φ записью $Sat(M, \Phi)$ будем обозначать множество всех состояний s этой модели, для которых верно $M, s \models \Phi$

Ctl-формула Φ выполняется на модели $M = (S, S_0, \to, L)$ ($M \models \Phi$), если справедливо включение $S_0 \subseteq Sat(M, \Phi)$

Небольшое пояснение:

- Ctl-формула делит все состояния модели на хорошие (в которых формула выполняется) и плохие (в которых формула не выполняется
- Соотношение $M \models \Phi$ означает, что все состояния, в которых система может начать своё выполнение, хорошие

Задача model checking для CTL (MC-CTL) формулируется так:

Для заданной модели Крипке M и заданной ctl-формулы Φ проверить справедливость соотношения

$$M \models \Phi$$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 22

Базовый алгоритм
model checking для CTL

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Дано:

- Конечная модель Крипке $M = (S, S_0, \to, L)$
- Ctl-формула Φ

Требуется проверить справедливость соотношения $M \models \Phi$

То есть требуется проверить включение $S_0 \subseteq Sat(M, \Phi)$

Базовый алгоритм работает с явным представлением модели Крипке как размеченного ориентированного графа

Алгоритм будет описан как набор рекурсивно вызываемых процедур

Основная процедура $\mathfrak{P}_{MC}$, отвечающая алгоритму, устроена так

Дано: конечная модель Крипке $M = (S, S_0, \to, L)$, ctl-формула Φ
Требуется проверить соотношение $M \models \Phi$

Устройство процедуры:

1. Вычислить множество $X = Sat(M, \Phi)$ при помощи описанной далее процедуры $\mathfrak{P}_{sat}$

2. Проверить включение $S_0 \subseteq X$

3. Вернуть результат проверки предыдущего пункта

Корректность основной процедуры обеспечивается

- определением выполнимости ctl-формулы на модели и
- обсуждающейся далее корректностью процедуры $\mathfrak{P}_{sat}$

Процедура $\mathfrak{P}_{sat}$

Дано: конечная модель Крипке $M = (S, S_0, \to, L)$, ctl-формула Φ
Требуется вычислить множество $Sat(M, \Phi)$

Устройство процедуры:

1. Используя известные равносильности, преобразовать Φ в равносильную *упрощённую* формулу Ψ в базисе **EX**, **EG**, **EU**:
$$\Psi \ ::= \ \mathbb{t} \ | \ p \ | \ \Psi \,\&\, \Psi \ | \ \neg\Psi \ | \ \mathbf{EX}\Psi \ | \ \mathbf{EG}\Psi \ | \ \mathbf{E}(\Psi \mathbf{U} \Psi)$$

2. $\mathfrak{P}_{sat}(M, \Phi) = \mathfrak{P}'_{sat}(M, \Psi)$
   - Процедура $\mathfrak{P}'_{sat}$ вычисления множества Sat для упрощённых формул будет описана дальше

Корректность этой процедуры обеспечивается равенством $Sat(M, \Phi) = Sat(M, \Psi)$, следующим из равносильности $\Phi \sim \Psi$

Процедура $\mathfrak{P}'_{sat}$

Дано: конечная модель Крипке $M = (S, S_0, \to, L)$, *упрощённая* ctl-формула Φ

Требуется вычислить множество $Sat(M, \Phi)$

Устройство процедуры:

- Если $\Phi = \mathbb{t}$, то $\mathfrak{P}'_{sat}(M, \Phi) = S$
- Если $\Phi = p \in \mathrm{AP}$, то $\mathfrak{P}'_{sat}(M, \Phi) = \{s \mid s \in S,\ p \in L(s)\}$
- Если $\Phi = \Psi_1 \,\&\, \Psi_2$, то $\mathfrak{P}'_{sat}(M, \Phi) = \mathfrak{P}'_{sat}(M, \Psi_1) \cap \mathfrak{P}'_{sat}(M, \Psi_2)$
- Если $\Phi = \neg\Psi$, то $\mathfrak{P}'_{sat}(M, \Phi) = S \setminus \mathfrak{P}'_{sat}(M, \Psi)$
- Если $\Phi = \mathbf{EX}\Psi$, то $\mathfrak{P}'_{sat}(M, \Phi) = \mathfrak{P}_{\mathbf{EX}}(M, \Psi)$
- Если $\Phi = \mathbf{EG}\Psi$, то $\mathfrak{P}'_{sat}(M, \Phi) = \mathfrak{P}_{\mathbf{EG}}(M, \Psi)$
- Если $\Phi = \mathbf{E}(\Psi_1 \mathbf{U} \Psi_2)$, то $\mathfrak{P}'_{sat}(M, \Phi) = \mathfrak{P}_{\mathbf{EU}}(M, \Psi_1, \Psi_2)$

Корректность этой процедуры для первых четырёх пунктов очевидна (обеспечивается семантикой ctl-формул)

Осталось предложить подходящие процедуры $\mathfrak{P}_{\mathbf{EX}}$, $\mathfrak{P}_{\mathbf{EG}}$ и $\mathfrak{P}_{\mathbf{EU}}$

Для ориентированного графа Γ и его вершины v и множества вершин V записями $Pre(\Gamma, v)$ и $Pre(\Gamma, V)$ обозначим множество вершин, из которых достижимы по одной дуге соответственно вершина v и хотя бы одна вершина множества V:

$$Pre(v) = \{v' \mid (v \to v') \in \Gamma\}$$
$$Pre(V) = \bigcup_{v \in V} Pre(\Gamma, v)$$

Утверждение. **Для любой модели Крипке M и любой ctl-формулы Φ справедливо равенство** $Sat(M, \mathbf{EX}\Phi) = Pre(M, Sat(M, \Phi))$

Доказательство.

$s \in Sat(M, \mathbf{EX}\Phi)$
$\Leftrightarrow M, s \models \mathbf{EX}\Phi$
$\Leftrightarrow$ существует состояние s', такое что $s \to s'$ и $M, s' \models \Phi$
$\Leftrightarrow$ хотя бы одно состояние (s') множества $Sat(M, \Phi)$ достижимо из s по одной дуге
$\Leftrightarrow s \in Pre(Sat(M, \Phi))$ ▼

Процедура $\mathfrak{P}_{\mathsf{EX}}$

Дано: конечная модель Крипке $M = (S, S_0, \to, L)$, *упрощённая* ctl-формула Φ
Требуется вычислить множество $Sat(M, \mathbf{EX}\Phi)$

Устройство процедуры:

1. Вычислить $X = \mathfrak{P}'_{sat}(M, \Phi)$
2. Вернуть множество $Pre(X)$

Корректность этой процедуры следует из

- последнего утверждения и
- предполагаемой (*по индукции*) корректности процедуры $\mathfrak{P}'_{sat}$

Утверждение. **Для любых конечной модели Крипке M, состояния s и ctl-формул Φ_1, Φ_2 верно следующее:**
$s \in Sat(M, \mathbf{E}(\Phi_1 \mathbf{U} \Phi_2)) \Leftrightarrow$ **в M существует путь $s_1 \to \cdots \to s_k$, такой что $s_1 = s$, $s_k \in Sat(M, \Phi_2)$ и $\{s_1, \ldots, s_{k-1}\} \subseteq Sat(M, \Phi_1)$**

Доказательство.

$s \in Sat(M, \mathbf{E}(\Phi_1 \mathbf{U} \Phi_2)$

$\Leftrightarrow M, s \models \mathbf{E}(\Phi_1 \mathbf{U} \Phi_2)$

$\Leftrightarrow$ существуют бесконечный путь π из s в M и момент времени k, такие что $M, \pi[k] \models \Phi_2$ и для любого момента времени i, меньшего k, верно $M, \pi[i] \models \Phi_1$

$\Leftrightarrow$ в M существует путь $s_1 \to \cdots \to s_k$ $(\pi[1] \to \cdots \to \pi[k])$, такой что $M, s_k \models \Phi_2$ и для всех $i \in \{1, \ldots, k-1\}$ верно $M, s_i \models \Phi_1$

$\Leftrightarrow$ в M существует путь $s_1 \to \cdots \to s_k$, такой что $s_k \in Sat(M, \Phi_2)$ и $\{s_1, \ldots, s_{k-1}\} \subseteq Sat(M, \Phi_1)$ ▼

Процедура $\mathfrak{P}_{\mathsf{EU}}$

Дано: конечная модель Крипке $M = (S, S_0, \to, L)$, *упрощённые* ctl-формулы Φ, Ψ

Требуется вычислить множество $Sat(M, \mathbf{E}(\Phi \mathbf{U} \Psi))$

Устройство процедуры:

1. Вычислить $Z = \mathfrak{P}'_{sat}(M, \Phi)$
2. Вычислить $X_0 = \mathfrak{P}'_{sat}(M, \Psi)$
3. Последовательно вычислять множества $X_1, X_2, \ldots$ по следующей схеме, пока для очередного вычисленного множества X_i не будет получено равенство $X_i = X_{i-1}$: $\qquad X_i = X_{i-1} \cup (Pre(X_{i-1}) \cap Z)$
4. Вернуть последнее вычисленное множество X_i

Корректность процедуры обосновывается

- последним утверждением,
- наблюдением «на грани очевидного» о том, что в множество X_i входят все вершины всех путей вида $s_1 \to \cdots \to s_j$, где $j \leq i$, $s_j \in Sat(M, \Phi_2)$ и $\{s_1, \ldots, s_{j-1}\} \subseteq Sat(M, \Phi_1)$, и
- гарантированным равенством $X_i = X_{i-1}$ хотя бы для одного i в связи с конечностью M

Утверждение. **В конечном ориентированном графе Γ из вершины s исходит хотя бы один бесконечный путь $\Leftrightarrow$ в Γ из s достижима хотя бы одна нетривиальная компонента сильной связности**

Доказательство этого утверждения несложно извлекается из теоремы о проверке пустоты автомата Бюхи

Для графа Γ и подмножества V его вершин записью $\Gamma|_V$ обозначим подграф графа Γ, порождённый множеством V:

- Множество вершин $\Gamma|_V$ — это V
- Дуга (s_1, s_2) входит в $\Gamma|_V \Leftrightarrow \{s_1, s_2\} \subseteq V$ и эта дуга входит в Γ
- Метки вершин и дуг переносятся из Γ в $\Gamma|_V$

Утверждение. **Для любой конечной модели Крипке M и любой ctl-формулы Φ верно следующее: $s \in Sat(M, \mathbf{EG}\Phi) \Leftrightarrow$ в графе $M|_{Sat(M,\Phi)}$ содержится вершина s и из неё достижима хотя бы одна нетривиальная компонента сильной связности**

Доказательство.

$s \in Sat(M, \mathbf{EG}\Phi) \Leftrightarrow M, s \models \mathbf{EG}\Phi$
$\Leftrightarrow$ в M существует бесконечный путь Φ, исходящий из s и такой что $M, \pi[i] \models \varphi$ для каждого момента времени i
$\Leftrightarrow$ в $\Gamma = M|_{Sat(M,\Phi)}$ существует бесконечный путь, исходящий из s
$\Leftrightarrow$ в Γ содержится s и из неё достижима хотя бы одна нетривиальная компонента сильной связности ▼

Процедура $\mathfrak{P}_{\mathbf{EG}}$

Дано: конечная модель Крипке $M = (S, S_0, \to, L)$, *упрощённая* ctl-формула Φ

Требуется вычислить множество $Sat(M, \mathbf{EG}\Phi)$

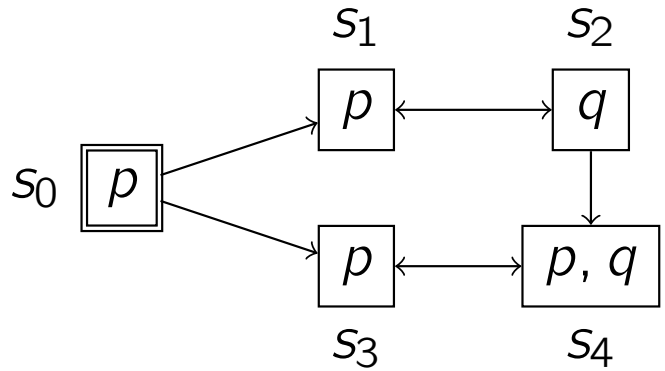
Устройство процедуры:

1. Вычислить множество $Z = Sat(M, \Phi)$
2. Вычислить граф $\Gamma = M|_Z$
3. Каким-либо известным эффективным алгоритмом вычислить множество X_0 всех вершин, входящих в какие-либо нетривиальные компоненты сильной связности графа Γ
4. Последовательно вычислять множества $X_1, X_2, \ldots$ по следующей схеме, пока не будет получено равенство $X_i = X_{i-1}$:
$$X_i = X_{i-1} \cup Pre(\Gamma, X_{i-1})$$
5. Вернуть последнее вычисленное множество X_i

Корректность этой процедуры показывается аналогично корректности процедуры $Sat_{\mathbf{EU}}$

Пример

M:



$$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$$

В процессе работы алгоритмом строятся следующие множества состояний

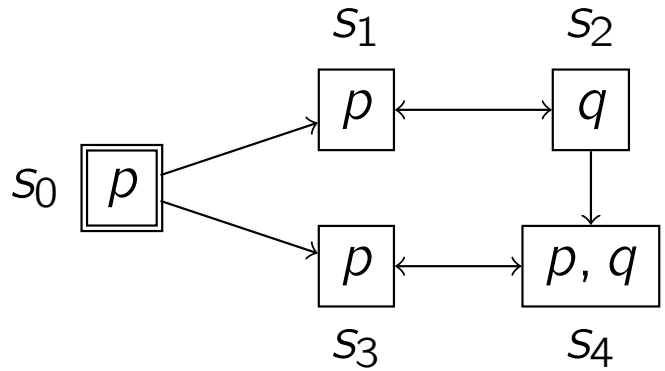
$$Sat(M, p) = \{s_0, s_1, s_3, s_4\}$$

$$Sat(M, \mathbf{EX}p) = \mathfrak{P}_{\mathbf{EX}}(M, p) = Pre(Sat(M, p)) = \{s_0, s_2, s_3, s_4\}$$

$$Sat(M, q) = \{s_2, s_4\}$$

Пример

M:



$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$

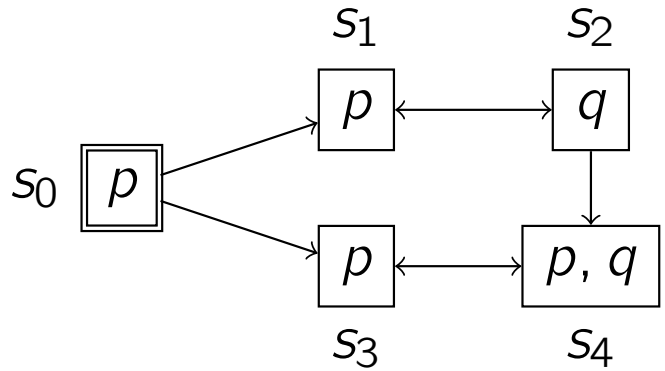
В процессе работы алгоритмом строятся следующие множества состояний

$Sat(M, \mathbf{EG}p) = \mathfrak{P}_{\mathbf{EG}}(M, p)$:

- $Z = Sat(M, p) = \{s_0, s_1, s_3, s_4\}$
- $X_0 = \{s_3, s_4\}$ (все вершины н.к.с.с. в $M|_Z$)
- $X_1 = X_0 \cup Pre(M|_X, X_0) = \{s_0, s_3, s_4\}$
- $X_2 = X_1 \cup Pre(M|_X, X_1) = X_1$
- $\mathfrak{P}_{\mathbf{EG}}(M, p) = X_2 = \{s_0, s_3, s_4\}$

Пример

M:



$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$

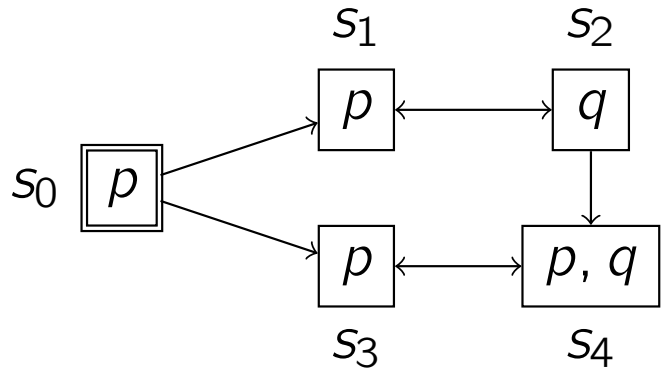
В процессе работы алгоритмом строятся следующие множества состояний

$Sat(M, \mathbf{E}(q\mathbf{UEG}p)) = \mathfrak{P}_{\mathbf{EU}}(M, q, \mathbf{EG}p)$:

- $Z = Sat(M, q) = \{s_2, s_4\}$
- $X_0 = Sat(M, \mathbf{EG}p) = \{s_0, s_3, s_4\}$
- $X_1 = X_0 \cup (Pre(X_0) \cap Z) = \{s_0, s_2, s_3, s_4\}$
- $X_2 = X_1 \cup (Pre(X_1) \cap Z) = X_1$
- $\mathfrak{P}_{\mathbf{EU}}(M, q, \mathbf{EG}p) = X_2 = \{s_0, s_2, s_3, s_4\}$

Пример

M:



$$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$$

В процессе работы алгоритмом строятся следующие множества состояний

$$Sat(M, \neg\mathbf{E}(q\mathbf{UEG}p)) = S \setminus Sat(M, \mathbf{E}(q\mathbf{UEG}p)) = \{s_1\}$$

$$Sat(M, \varphi) = Sat(M, \mathbf{EX}p) \cap Sat(M, \neg\mathbf{E}(q\mathbf{UEG}p)) = \emptyset$$

Так как $\{s_0\} \not\subseteq \emptyset$, можно заключить, что $M \not\models \varphi$

А какова сложность базового алгоритма относительно количества вершин и дуг в модели и количества операций в формуле?

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 23

Справедливость и CTL

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Напоминание

Система переходов над атомарными высказываниями AP и действиями Act — это модель Крипке, каждый переход которой помечен действием

Безусловная справедливость: действия заданного множества должны выполняться бесконечно часто

Сильная справедливость: если действия заданного множества могут выполняться бесконечно часто, то они должны выполняться бесконечно часто

Слабая справедливость: если действия заданного множества могут выполняться почти всегда, то они должны выполняться бесконечно часто

Ограничения справедливости $\mathcal{F} = (\mathcal{F}_u, \mathcal{F}_s, \mathcal{F}_w)$, где $\mathcal{F}_u, \mathcal{F}_s, \mathcal{F}_w \subseteq 2^{\text{Act}}$ — это тройка семейств множеств, составляющих систему ограничений справедливости: каждое множество из $\mathcal{F}_u$ — безусловной, каждое из $\mathcal{F}_s$ — сильной, каждое из $\mathcal{F}_w$ — слабой

Напоминание

В LTL ограничения справедливости укладывались в рамки языка: если можно записать в виде формулы фразы «сейчас может выполниться действие множества A» и «сейчас будет выполнено действие множества A», то ограничения справедливости можно было встроить в проверяемую формулу

В CTL так сделать нельзя — *из-за синтаксических ограничений на использование темпоральных операторов и, как будет показано дальше, невозможности записать на языке CTL словосочетание «почти всегда»*

Поэтому при внесении справедливости в CTL приходится изменять терминологический и алгоритмический фундамент: отношение выполнимости и алгоритм проверки выполнимости

Напоминание

Отношение выполнимости в ограничениях справедливости $\mathcal{F}$ ($\models_{\mathcal{F}}$) отличается от «обычного» отношения выполнимости для ctl-формул следующими пунктами определения:

- $M, s \models_{\mathcal{F}} \mathbf{A}\varphi \Leftrightarrow$ для любого $\mathcal{F}$-справедливого пути π в M, исходящего из s, верно $M, \pi \models \varphi$

- $M, s \models_{\mathcal{F}} \mathbf{E}\varphi \Leftrightarrow$ существует $\mathcal{F}$-справедливый путь π в M, исходящий из s и такой что $M, \pi \models \varphi$

Напоминание

Алгоритм model checking в ограничениях справедливости $\mathcal{F}$ может быть получен из базового внесением исправлений в процедуры $\mathfrak{P}_{\mathbf{EX}}$, $\mathfrak{P}_{\mathbf{EU}}$ и $\mathfrak{P}_{\mathbf{EG}}$:

- $\mathfrak{P}_{\mathbf{EX}}(M, \varphi)$: множество $\mathfrak{P}'_{sat}(M, \varphi)$ следует заменить на $\mathfrak{P}'_{sat}(M, \varphi) \cap \mathit{FairStates}(M, \mathcal{F})$
  - $\mathit{FairStates}(M, \mathcal{F})$ — множество всех состояний M, из которых в M исходит хотя бы один $\mathcal{F}$-справедливый путь
- $\mathfrak{P}_{\mathbf{EU}}(M, \varphi_1, \varphi_2)$: множество $\mathfrak{P}'_{sat}(M, \varphi_2)$ следует заменить на $\mathfrak{P}'_{sat}(M, \varphi_2) \cap \mathit{FairStates}(M, \mathcal{F})$
- $\mathfrak{P}_{\mathbf{EG}}(M, \varphi)$: н.к.с.с. следует заменить на $\mathcal{F}$-справедливые н.к.с.с., то есть такие, для которых возможен бесконечный обход, отвечающий $\mathcal{F}$-справедливому пути

Для самостоятельного размышления:

а как вычислять $\mathit{FairStates}(M, \mathcal{F})$ и справедливые н.к.с.с.?

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 24

Символьные представления моделей

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Базовый алгоритм model checking для CTL

- идеологически прост: сводит задачу проверки выполнимости формулы на модели к набору хорошо известных задач теории графов:
  - выделение компонент сильной связности,
  - проверка достижимости вершин
  - ...

- теоретически эффективен: его сложность полиномиальна с невысокой степенью (без учёта алгоритма вычисления н.к.с.с. — линейна) относительно размеров модели и формулы

Вступление

Но у базового алгоритма есть и недостатки

Для примера можно представить *относительно небольшую* систему из 20 процессов, в каждом из которых содержится 20 булевых переменных

В такой системе будет более 10^{120} состояний (*чтобы осознать, насколько это много: больше числá гугóл*)

В базовом алгоритме граф модели хранится **явно и полностью**, что приводит к неразумному и даже невозможному расходу памяти

А можно ли сократить расход памяти, представив модель более эффективно?

Вступление

Напоминание: конечная модель Крипке $M = (S, S_0, \to, L)$ — это

- конечное множество состояний S
- конечное множество начальных состояний S_0
- отношение переходов на паре конечных множеств состояний $\to$
- функция разметки, сопоставляющая каждому состоянию конечное множество L

Другое напоминание:

- $\mathbb{B} = \{0, 1\}$ (будем использовать 0 как синоним $\mathbb{f}$ и 1 как синоним $\mathbb{t}$)
- Булева функция местности n — это функция $f : \mathbb{B}^n \to \mathbb{B}$

Символьные представления

Символьные представления — это семейство эффективных представлений структур (графовых и не только), становящееся всё более популярным в практике «умного» программирования, и коротко можно описать его так:

1. Представляемый объект переписывается (кодируется) как совокупность множеств наборов нолей и единиц заданной длины
2. Каждое множество S наборов нолей и единиц длины n расценивается как характеристическое множество n-местной булевой функции $\mathfrak{f}_S$: $\mathfrak{f}_S(\widetilde{\alpha}) = 1 \Leftrightarrow \widetilde{\alpha} \in S$
3. Выбирается подходящее представление получающихся булевых функций в зависимости от природы исходных объектов и от того, как с ними предполагается взаимодействовать (*этот пункт необязателен, пока речь не идёт о программной реализации*)
4. Символьное представление объекта — это совокупность представлений булевых функций, кодирующих объект, и реализация требуемых операций над объектом в терминах булевых функций

Символьное представление графа

Начнём с простого **примера**: опишем символьное представление конечного ориентированного графа $G = (V, E)$, где V — это конечное множество вершин и $E \subseteq V \times V$ — множество дуг (двуместное отношение на множестве вершин)

Пронумеруем вершины графа: $V = \{v_0, v_1, , \ldots, v_n\}$

Сопоста́вим каждой вершине уникальное число — например, её номер: $V_{\mathbb{N}} = \{0, 1, \ldots, n\}$

Как-либо выберем такое количество битов k, в которое вмещается двоичная запись наибольшего рассматриваемого числа — например, $k = \lfloor \log_2(n) \rfloor + 1$

Символьное представление графа

Заме́ним каждое число i на его двоичную запись $(i)_2^k$ в k битах

В результате получим множество наборов нолей и единиц длины k, представляющее множество V:
$V_{\mathbb{B}} = \{(00\ldots00), (00\ldots01), (00\ldots10), \ldots, (n)_2^k\}$

Перейдём от этого множества к булевой функции: $\mathfrak{f}_{V_{\mathbb{B}}}((i)_2^k) \Leftrightarrow i \leq n$

Будем считать, что это функция над переменными $x_0, \ldots, x_{k-1}$, от младшего (правого) бита двоичной записи к старшему (левому)

Представим как-либо эту функцию — например (*но так обычно не делают*), в виде совершенной ДНФ: $\varphi_V = \neg x_{k-1} \,\&\, \ldots \,\&\, \neg x_1 \,\&\, \neg x_0 \vee \neg x_{k-1} \,\&\, \ldots \,\&\, \neg x_1 \,\&\, x_0 \vee \neg x_{k-1} \,\&\, \ldots \,\&\, x_1 \,\&\, \neg x_0 \vee \neg x_{k-1} \,\&\, \ldots \,\&\, x_1 \,\&\, x_0 \vee \ldots$

Такое представление множества (какая-либо форма задания соответствующей булевой функции) будем называть стандартным символьным

Символьное представление графа

Отношение $E \subseteq V \times V$ перепишем как соответствующее отношение $E_{\mathbb{B}} \subseteq \mathbb{B}^k \times \mathbb{B}^k$

Перейдём от этого отношения к его характеристической функции: $(2k)$-местной булевой функции $\mathfrak{f}_E$, такой что $f(\widetilde{\alpha}, \widetilde{\beta}) = 1 \Leftrightarrow (\widetilde{\alpha}, \widetilde{\beta}) \in E$

Будем считать, что это функция над двумя комплектами переменных: $x_0, \ldots, x_{k-1}$ для первого набора (начала дуги) и $x'_0, \ldots, x'_{k-1}$ для второго (конца дуги)

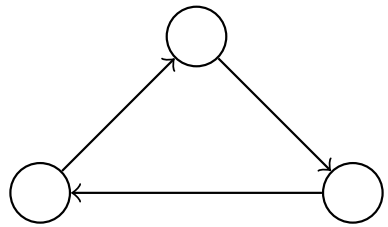
Например, дуга $(0010) \to (1011)$ может быть представлена как конъюнкция $\neg x_3 \,\&\, \neg x_2 \,\&\, x_1 \,\&\, \neg x_0 \,\&\, x'_3 \,\&\, \neg x'_2 \,\&\, x'_1 \,\&\, x'_0$

А отношение $E_{\mathbb{B}}$ можно представить как дизъюнкцию представлений дуг

Такое представление двуместных отношений (какая-либо форма задания соответствующей булевой функции над двумя комплектами переменных) будем называть стандартным символьным

Символьное представление графа

Например:



Закодируем вершины числами 0 (левая), 1 (верхняя) и 2 (правая)

Запишем эти числа двоично в двух битах: $(0)_2^2 = (00)$, $(1)_2^2 = (01)$, $(2)_2^2 = (10)$

Характеристическую функцию множества $\{(00), (01), (10)\}$, отвечающего вершинам, можно представить формулой

$$\neg(x_1 \,\&\, x_0)$$

Характеристическую функцию отношения

$$\{((00), (01)), ((01), (10)), ((10), (00))\},$$

отвечающего дугам, можно представить формулой

$$\neg x_1 \,\&\, \neg x_0 \,\&\, \neg x_1' \,\&\, x_0' \vee \neg x_1 \,\&\, x_0 \,\&\, x_1' \,\&\, \neg x_0' \vee x_1 \,\&\, \neg x_0 \,\&\, \neg x_1' \,\&\, \neg x_0'$$

Символьное представление модели Крипке

Конечную модель Крипке $M = (S, S_0, \to, L)$ над множеством AP можно представить как набор следующих конечных множеств и отношений:

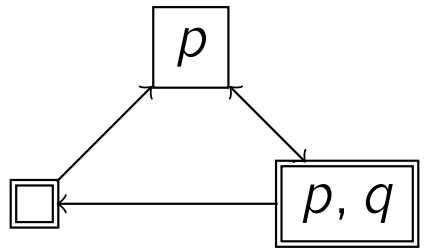
- S — конечное множество, самое большое по включению множество
- S_0 — конечное множество, $S_0 \subseteq S$
- $\to$ — двуместное отношение, $\to \subseteq S \times S$
- Для каждого атомарного высказывания p, $p \in \mathrm{AP}$, — множество всех вершин, в которых p истинно:

$$S_p = \{s \mid s \in S,\ p \in L(s)\}$$

Символьным представлением модели Крипке будем называть совокупность стандартных представлений для S, S_0, $\to$ и S_p для каждого $p \in \mathrm{AP}$ с использованием общего числа битов k

Символьное представление модели Крипке

Например ($\mathrm{AP} = \{p, q\}$)



Пример символьного представления этой модели Крипке для нумерации состояний 0 (левое), 1 (верхнее), 2 (правое) и двух битов:

- Множество состояний: $\neg(x_1 \,\&\, x_0)$
- Множество начальных состояний: $\neg x_0$
- Отношение переходов:
$$\neg x_1 \,\&(\neg x_0 \,\&\, x_1' \,\&\, \neg x_0' \vee \neg x_0 \,\&\, x_1' \,\&\, x_0') \vee x_1 \,\&\, \neg x_0 \,\&\, \neg x_1'$$
- Множество S_p: $x_0 \oplus x_1$
- Множество S_q: $x_1 \,\&\, \neg x_0$

Операции над символьными представлениями

Мало уметь записывать множества как булевы функции, обычно требуется уметь их строить и применять к ним теоретико-множественные операции

Стандартное представление $\emptyset$: $\mathbb{f}$

Стандартное представление $\{i\}$ — это элементарная конъюнкция, отвечающая i

Если φ_1 — представление множества S_1 и φ_2 — представление множества S_2, то:

- $S_1 \cup S_2$ представляется как $\varphi_1 \vee \varphi_2$
- $S_1 \cap S_2$ представляется как $\varphi_1 \,\&\, \varphi_2$
- $S_1 \setminus S_2$ представляется как $\varphi_1 \,\&\, \neg\varphi_2$

Откуда берутся символьные представления (пример)

Чтобы развеять впечатление о том, что символьное представление модели Крипке — это «неестественная» конструкция, необходимая только в технических целях, приведём пример, когда такое представление возникает прежде «явного» (*а ещё лучше это будет заметно в обязательном задании по средству NuSMV*)

Представим себе императивную программу над переменными $V = \{v_1, \ldots, v_n\}$, каждая из которых принимает значения из конечного множества D (домена)

Считая переменные упорядоченными по номерам, будем считать состоянием данных набор значений $(d_1, \ldots, d_n) \in D^n$

Состоянием управления будем считать значение счётчика команд — особой переменная pc, принимающей значения из конечного множества D_{pc} и значение которой — это *номер* команды, которая будет выполняться следующей

Тогда состояния вычисления — это элементы множества $D^n \times D_{pc}$

Откуда берутся символьные представления (пример)

Каждое «элементарное» утверждение о значениях переменных и счётчике команд: $v_i = k$, $v_i = v_j$, $pc = k$ — можно трактовать как сокращение для булевой формулы, описывающей такое равенство для двоичных записей значений переменных

Например, если переменной v_2 сопоставлены переменные x_3, x_4, x_5 (от младшего бита к старшему), то выражение $v_2 = 3$ — это сокращение для формулы $\neg x_5 \,\&\, x_4 \,\&\, x_3$

Остальные («производные») соотношения между значениями переменных и счётчика команд аналогично можно считать сокращениями для соответствующих булевых формул

Состоянию данных и состоянию вычисления естественно сопоставляются конъюнкции таких выражений

Например, состоянию данных $(2, 3, 5)$ для переменных v_1, v_2, v_3 соответствует формула $\varphi = (v_1 = 2) \,\&\, (v_2 = 3) \,\&\, (v_3 = 5)$, а этой оценке и значению счётчика команд 5 — формула $\varphi \,\&\, (pc = 5)$

Откуда берутся символьные представления (пример)

Если состояние данных, с которого программа начинает вычисление, однозначно определено, то множесто начальных состояний модели задаётся формулой, отвечающей этому состоянию и начальному значению счётчика команд (обычно — 0)

Если нет, то семейство допустимых оценок нередко *естественно* записывается в виде формулы (предусловия)

Самый простой и при этом полезный вид атомарных высказываний для рассматриваемой программы — это высказывания вида $v_i = k$

Множество состояний S_p, которые размечены высказыванием $p = (v_i = k)$, задаётся формулой $v_i = k$

Откуда берутся символьные представления (пример)

Множество переходов, определяемых для команды C операционной семантикой, как правило, легко выражается в виде формулы ψ_C над двумя комплектами переменных: «обычные» для текущих значений переменных и текущего значения счётчика команд, и «штрихованные» — для значений переменных и счётчика команд после выполнения C

Например, переходы для команды $\mathtt{x} := \mathtt{y} + 1$; программы над $\{\mathtt{x}, \mathtt{y}\}$ задаются формулой $(\mathtt{x}' = \mathtt{y} + 1)\,\&(\mathtt{y}' = \mathtt{y})\,\&(\mathtt{pc}' = \mathtt{pc} + 1)$:

- Следующее значение переменной $\mathtt{x}$ — это текущее значение переменной $\mathtt{y}$ плюс один
- Значение переменной $\mathtt{y}$ не изменяется
  - **Важно!** — если вычеркнуть множитель «$\mathtt{y}' = \mathtt{y}$», то это будет означать «выбор следующего значения $\mathtt{y}$ не влияет на истинность формулы», то есть «значение $\mathtt{y}$ может произвольно измениться при выполнении команды»
- Управление передаётся команде со следующим номером

Откуда берутся символьные представления (пример)

Для примера рассмотрим программу $\mathtt{x} := \mathtt{x} + 1;\ \mathtt{y} := \mathtt{x} + \mathtt{y};$ над переменными $\mathtt{x}$, $\mathtt{y}$ с доменом $\mathbb{B}$, и условимся, что значение счётчика команд 0 указывает на первое присваивание, 1 — на второе, и 2 означает, что программа завершилась

Счётчику команд $\mathtt{pc}$ сопоста́вим булевы переменные $\mathtt{pc}_0$ (младший бит) и $\mathtt{pc}_1$ (старший бит)

Тогда:

- Множество состояний представляется формулой $\neg(\mathtt{pc} = 3)$
- Множество переходов представляется формулой

$$\begin{aligned}&(\mathtt{pc} = 0)\,\&(\mathtt{x}' \oplus \mathtt{x})\,\&(\mathtt{y}' \leftrightarrow \mathtt{y})\,\&(\mathtt{pc}' = \mathtt{pc} + 1) \vee\\ &(\mathtt{pc} = 1)\,\&(\mathtt{x}' \leftrightarrow \mathtt{x})\,\&(\mathtt{y}' \leftrightarrow (\mathtt{x} \oplus \mathtt{y}))\,\&(\mathtt{pc}' = \mathtt{pc} + 1) \vee\\ &(\mathtt{pc} = 2)\,\&(\mathtt{x}' \leftrightarrow \mathtt{x})\,\&(\mathtt{y}' \leftrightarrow \mathtt{y})\,\&(\mathtt{pc}' = \mathtt{pc})\end{aligned}$$

- Утверждения о значениях $\mathtt{pc}$ «раскодируются» так:

$$\begin{aligned}&(\mathtt{pc} = 0) \sim (\neg\mathtt{pc}_1\,\&\,\neg\mathtt{pc}_0) \quad (\mathtt{pc} = 1) \sim (\neg\mathtt{pc}_1\,\&\,\mathtt{pc}_0)\\ &(\mathtt{pc} = 2) \sim (\mathtt{pc}_1\,\&\,\neg\mathtt{pc}_0) \quad (\mathtt{pc} = 3) \sim (\mathtt{pc}_1\,\&\,\mathtt{pc}_0)\\ &(\mathtt{pc}' = \mathtt{pc}) \sim ((\mathtt{pc}_1 \leftrightarrow \mathtt{pc}_1')\,\&(\mathtt{pc}_0 \leftrightarrow \mathtt{pc}_0'))\\ &(\mathtt{pc}' = \mathtt{pc} + 1) \sim ((\mathtt{pc}_1' \leftrightarrow (\mathtt{pc}_1 \oplus \mathtt{pc}_0))\,\&(\mathtt{pc}_0' \oplus \mathtt{pc}_0))\end{aligned}$$

Откуда берутся символьные представления (пример)

Для самостоятельного размышления:

а как выглядят формулы, задающие семантику всех других команд императивных программ из блока 3 для «естественной» арифметики с переполнением?

(*Это осознать не очень сложно, но весьма полезно*)

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 25

Двоичные решающие диаграммы:
BDD, OBDD, ROBDD

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Двоичная решающая диаграмма (Binary Decision Diagram; BDD) — это один из наиболее популярных способов эффективного представления булевых функций для таких задач, в которых удаётся символьно представить основные структуры данных решения

Существует несколько широко применяющихся русских вариантов названия этой структуры данных:

- Вместо «двоичная» иногда пишут «бинарная»
- Вместо «решающая диаграмма» иногда пишут «разрешающая диаграмма» или «диаграмма решений»

Вступление

Символьные представления нередко основываются на BDD или аналогичных структурах, и поэтому считается, что для более полного понимания тонкостей эффективной работы с символьными представлениями следует иметь общие знания об устройстве BDD

В эти знания входит *как минимум* то,

- как устроены BDD (синтаксис) и какие булевы функции ими реализуются (семантика)
- как строить BDD по другим представлениям (например, формулам)
- какие операции можно выполнять над BDD и как устроены соответствующие алгоритмы

BDD: синтаксис

Двоичная решающая диаграмма над упорядоченным набором переменных $x_1, \ldots, x_n$ — это конечный ациклический ориентированный граф, устроенный так

Вершины BDD обычно называют узлами

Особо выделены два терминальных узла: 0 и 1 — а остальные узлы называются внутренними

Каждый внутренний узел v помечен одной из переменных ($var(v)$)

Из каждого внутреннего узла исходят ровно две дуги́, одна помечена символом 0 ($\xrightarrow{0}$), другая — символом 1 ($\xrightarrow{1}$)

Из терминального узла не исходит ни одной дуги́

Узел, в который из v ведёт дуга $\xrightarrow{b}$, будем обозначать записью $v[b]$

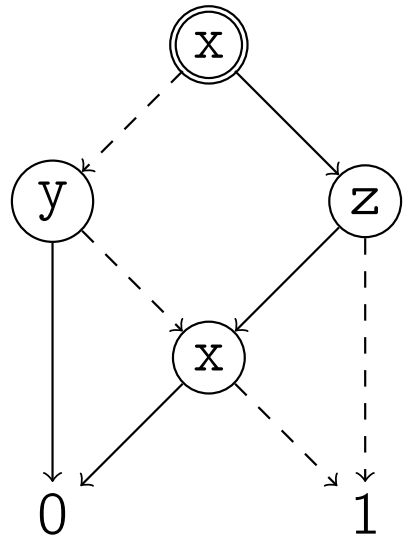
Узлы $v[0]$ и $v[1]$ будем называть соответственно младшим потомком ($low(v)$) и старшим потомком ($high(v)$) узла v

Ду́ги $\xrightarrow{1}$ и $\xrightarrow{0}$ также будем изображать соответственно как сплошную и пунктирную стрелки без помечающих чисел

Один (и только один) из узлов BDD объявлен корнем

BDD: синтаксис

Пример



Корень BDD изображён двойным контуром

Для узла (y) верно следующее:

- $high(v) = v[1] = 0$
- $var(low(v)) = var(v[0]) = \mathtt{x}$

BDD: семантика

Каждому узлу v BDD $\mathcal{D}$ над переменными $x_1, \ldots, x_n$ сопоста́вим n-местную булеву формулу Φ_v^D:

- $\Phi_0^D = 0$
- $\Phi_1^D = 1$
- Для внутреннего узла v: $\Phi_v^D = \neg var(v) \,\&\, \Phi_{low(v)}^D \vee var(v) \,\&\, \Phi_{high(v)}^D$

В узле v BDD $\mathcal{D}$ над переменными $x_1, \ldots, x_n$ реализуется n-местная булева функция $\mathfrak{f}_v^D$ — это функция, реализуемая формулой Φ_v^d

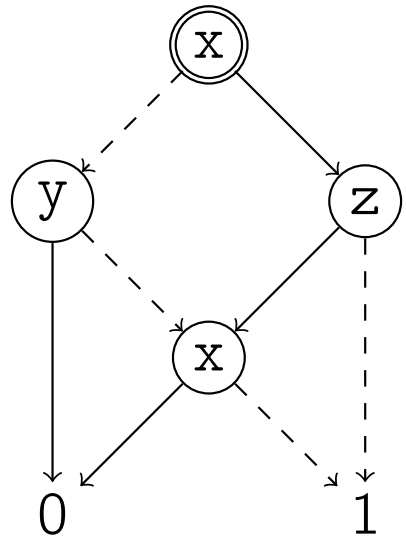
BDD $\mathcal{D}$ реализует булеву функцию $\mathfrak{f}^D$, реализуемую в её корне

Альтернативный способ определения значения $\mathfrak{f}^D(\alpha_1, \ldots, \alpha_n)$ (значения $\mathcal{D}$ на оценке переменных $\xi = [\mathrm{x}_1/\alpha_1, \ldots, \mathrm{x}_n/\alpha_n]$):

- Обойдём BDD как граф, начав в корне
- Если текущий узел v — внутренний, то следующим обходится узел $v[\xi(var(v))]$
- Если обход достиг узла 0 или 1, то этот достигнутый узел объявляется значением $\mathfrak{f}^D(\alpha_1, \ldots, \alpha_n)$

BDD: семантика

Пример диаграммы $\mathcal{D}$:



Пронумеруем внутренние узлы так: верхний, левый, правый, нижний — v_0, v_1, v_2, v_3

$\Phi^{\mathcal{D}}_{v_3} = \neg \mathrm{x} \,\&\, 1 \vee \mathrm{x} \,\&\, 0 \sim \neg \mathrm{x}$

$\Phi^{\mathcal{D}}_{v_2} = \neg \mathrm{z} \,\&\, 1 \vee \mathrm{z} \,\&\, \Phi^{\mathcal{D}}_{v_3} \sim \neg \mathrm{z} \vee \neg \mathrm{x}$

$\Phi^{\mathcal{D}}_{v_1} = \neg \mathrm{y} \,\&\, \Phi^{\mathcal{D}}_{v_3} \vee \mathrm{y} \,\&\, 0 \sim \neg \mathrm{y} \,\&\, \neg \mathrm{x}$

$\Phi^{\mathcal{D}}_{v_0} = \neg \mathrm{x} \,\&\, \Phi^{\mathcal{D}}_{v_1} \vee \mathrm{x} \,\&\, \Phi^{\mathcal{D}}_{v_2} \sim \neg x \,\&\, \neg \mathrm{y} \vee \mathrm{x} \,\&\, \neg \mathrm{z}$

То есть диаграммой $\mathcal{D}$ реализуется та же функция, что и формулой $\Phi^{\mathcal{D}}_{v_0}$

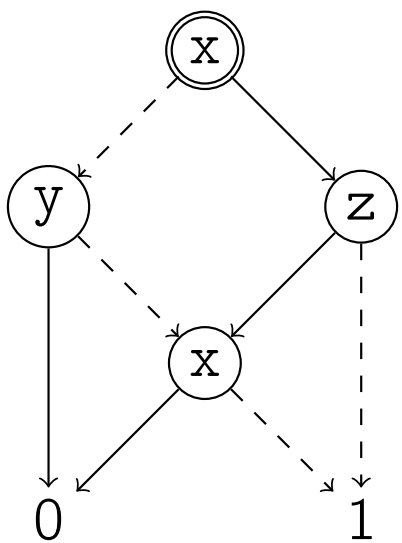
Это подтверждает и обход на оценке $[\mathrm{x}/1, \mathrm{y}/0, \mathrm{z}/1]$: $\textcircled{\textcircled{\mathrm{x}}} \xrightarrow{1} \textcircled{\mathrm{z}} \xrightarrow{1} \textcircled{\mathrm{x}} \xrightarrow{1} 0$

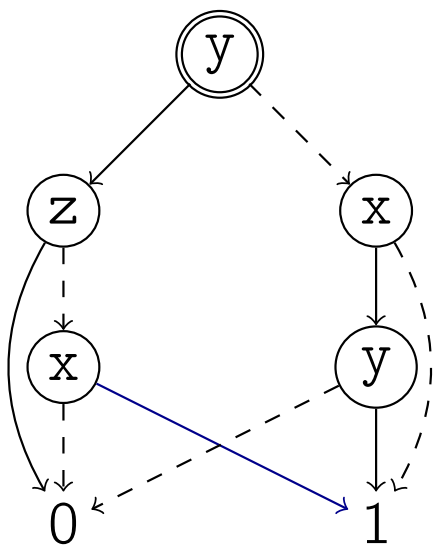
BDD: семантика

BDD $\mathcal{D}_1$, $\mathcal{D}_2$ (а также BDD $\mathcal{D}$ и формула φ) над переменными $x_1, \ldots, x_n$ называются эквивалентными ($\mathcal{D}_1 \sim \mathcal{D}_2$; $\mathcal{D} \sim \varphi$), если ими реализуются одинаковые функции над этим набором переменных

Существенная часть применения BDD — это проверка их эквивалентности, и способ такой проверки неочевиден: одна и та же функция может быть реализована существенно разными по структуре диаграммами

Например, такие две BDD эквивалентны:



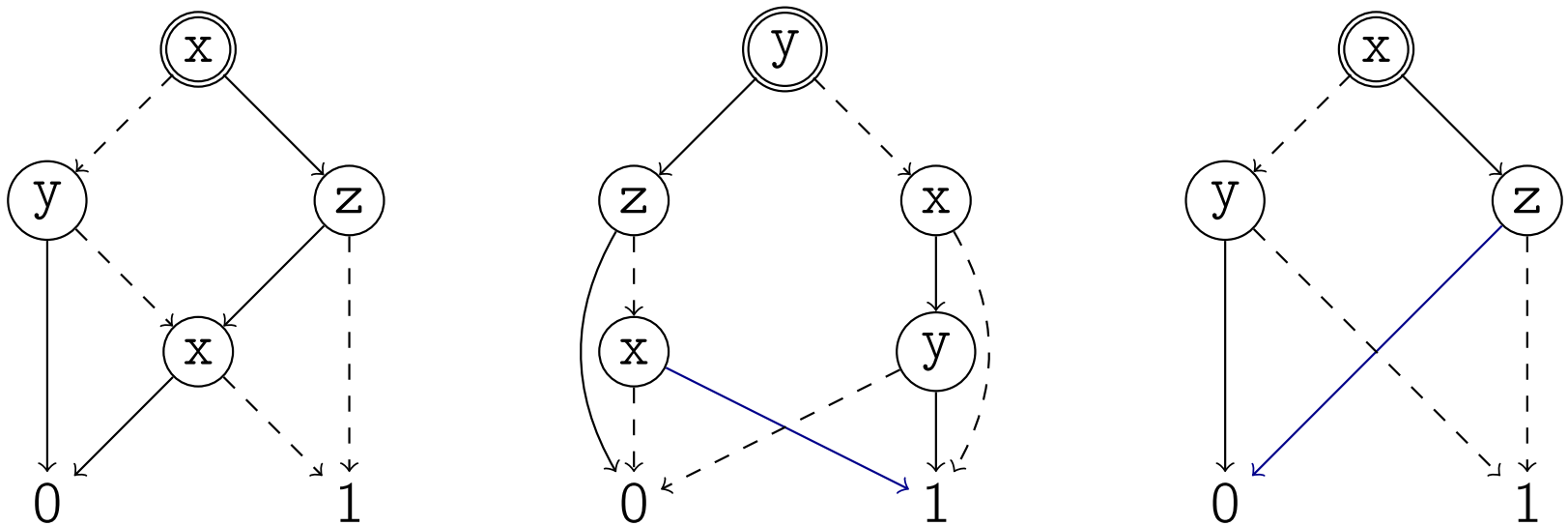


OBDD

Один из факторов, влияющих на трудность проверки эквивалентности BDD — это возможность записывать переменные в узлах «хаотично»

BDD $\mathcal{D}$ над набором переменных $x_1, \ldots, x_n$ называется упорядоченной (Ordered BDD; OBDD), если для любой дуги $v \to w$, ведущей во внутренний узел, верно $var(v) < var(w)$ для естественного порядка $<$ переменных: $x_i < x_j \Leftrightarrow i < j$

Например, среди изображённых ниже эквивалентных BDD над набором переменных $\mathtt{x}, \mathtt{y}, \mathtt{z}$ только самая правая упорядочена (является OBDD)

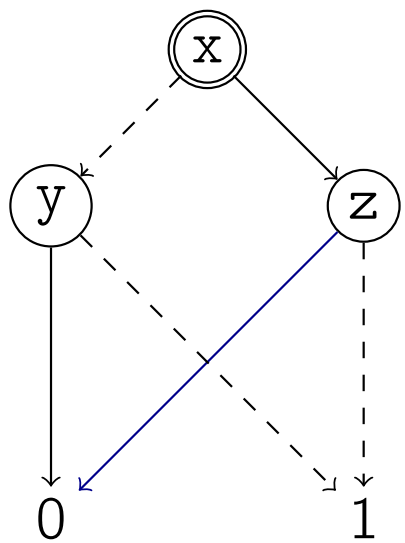


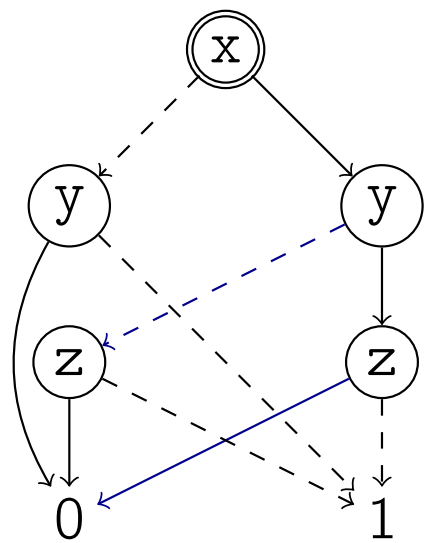
ROBDD

Для практических целей хотелось бы иметь **каноническое** представление булевых функций решающими диаграммами: единственное, и при этом такое, с которым было бы достаточно удобно работать (строить и преобразовывать)

OBDD не могут быть использованы в этом качестве: эквивалентные OBDD могут иметь заметно разную структуру

Например, следующие две OBDD над $\mathtt{x}, \mathtt{y}, \mathtt{z}$ эквивалентны, но имеют различное число узлов и хотя и не очень сильно, но всё же заметно различный внешний вид



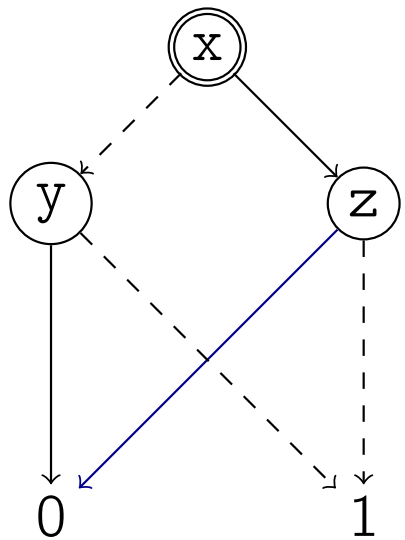


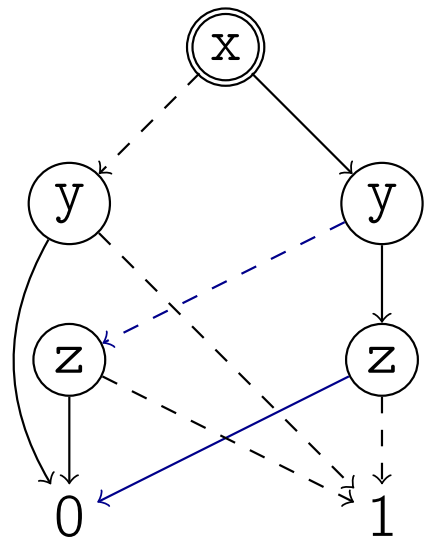
ROBDD

OBDD называется приведённой (или сокращённой, или редуцированной; Reduced OBDD; ROBDD), если для неё выполнены два условия:

1. Все внутренние узлы достижимы из корня
2. В любой паре различных узлов реализуются различные функции

Например, среди изображённых ниже OBDD над $\mathtt{x}, \mathtt{y}, \mathtt{z}$ левая является приведённой, а правая — нет

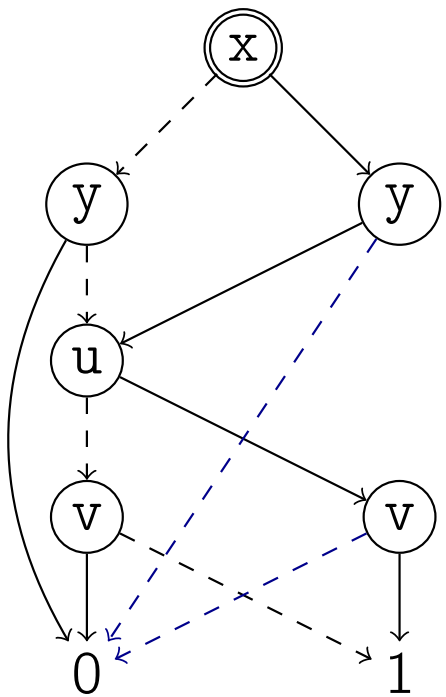


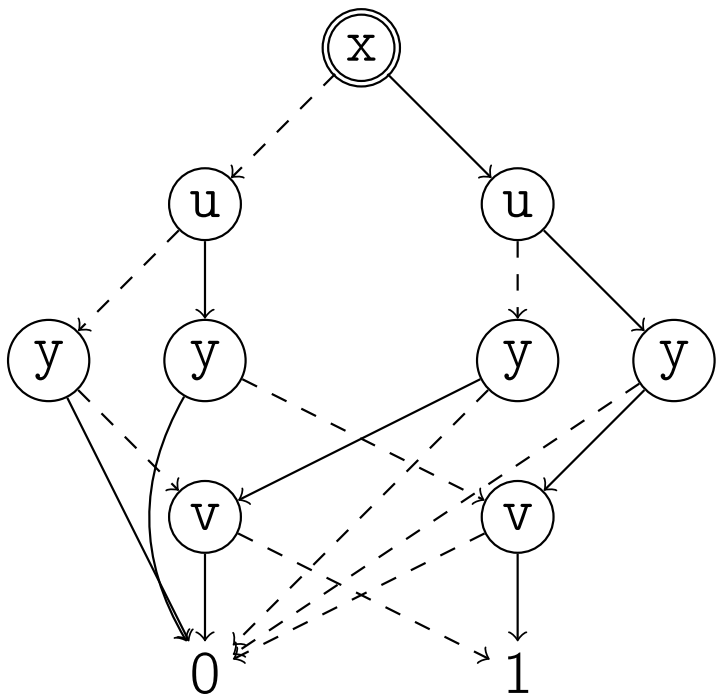


ROBDD

При использовании ROBDD очень важную роль играет выбор порядка переменных, над которой строится BDD

Например, ROBDD, эквивалентная формуле $(\mathtt{x} \leftrightarrow \mathtt{y}) \,\&\, (\mathtt{u} \leftrightarrow \mathtt{v})$ для порядка $\mathtt{x} < \mathtt{y} < \mathtt{u} < \mathtt{v}$ имеет заметно меньше узлов, чем для порядка $\mathtt{x} < \mathtt{u} < \mathtt{y} < \mathtt{v}$:





ROBDD

Так как BDD обсуждаются только как вспомогательный инструмент, основные их свойства и алгоритмы будут приводиться без обоснования

Утверждение. **Если ROBDD D_1 и D_2 над общим набором переменных эквивалентны, то они изоморфны как размеченные графы**

Изоморфизм ROBDD проверить несложно:

- В изоморфизм ϕ (как двуместное отношение на вершинах) входит пара корней диаграмм
- Если $(v, w) \in \phi$, то и $(low(v), low(w)), (high(v), high(w)) \in \phi$
- Используя два предыдущих пункта, можно вычислить множество всех пар узлов, которые обязаны входить в изоморфизм ϕ
- Диаграммы изоморфны $\Leftrightarrow$ в каждой паре $(v, w) \in \phi$
  - либо оба узла внутренние и $var(v) = var(w)$,
  - либо оба узла терминальные и $v = w$

То есть ROBDD гарантируют единственность (с точностью до изоморфизма) представления булевой функции

А насколько эффективны основные операции над ROBDD?

ROBDD: приведение OBDD

Дано: OBDD $\mathcal{D}$ над $x_1, \ldots, x_n$

Требуется построить ROBDD $\mathcal{D}^*$ над $x_1, \ldots, x_n$, эквивалентную $\mathcal{D}$

Алгоритм устроен несложно — пока это возможно, выполнять следующие три преобразования:

1. Если в диаграмме есть внутренний узел v, отличный от корня и не имеющий ни одной входящей дуги́, то удалить v и исходящие ду́ги
2. Если в диаграмме есть внутренний узел v, такой что $\mathit{low}(v) = \mathit{high}(v)$, то удалить v и перенаправить все ду́ги, входившие в v, в $\mathit{low}(v)$; если v был корнем, то объявить $\mathit{low}(v)$ корнем
3. Если в диаграмме есть различные внутренние узлы v, w, такие что $\mathit{var}(v) = \mathit{var}(w)$, $\mathit{low}(v) = \mathit{low}(w)$, $\mathit{high}(v) = \mathit{high}(w)$ и w не корень, то удалить узел w и перенаправить все ду́ги, входившие в w, в v

ROBDD: простейшие диаграммы

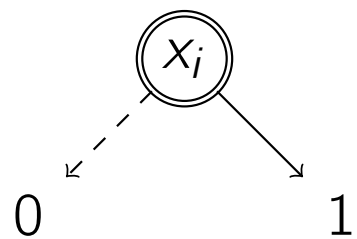
$\mathcal{D}^{\varphi}$ — так обозна́чим ROBDD, эквивалентную формуле φ

ROBDD для простейших функций устроены очень просто

$\mathcal{D}^{0}$:

$\mathcal{D}^{1}$:

$\mathcal{D}^{x_i}$:



ROBDD: отрицание

Дано: ROBDD $\mathcal{D}^{\varphi}$ над $x_1, \ldots, x_n$

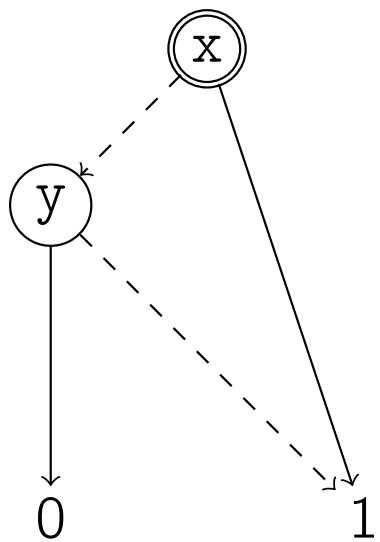
Требуется построить ROBDD $\neg D^{\varphi} = \mathcal{D}^{\neg\varphi}$ над $x_1, \ldots, x_n$

Алгоритм устроен несложно:

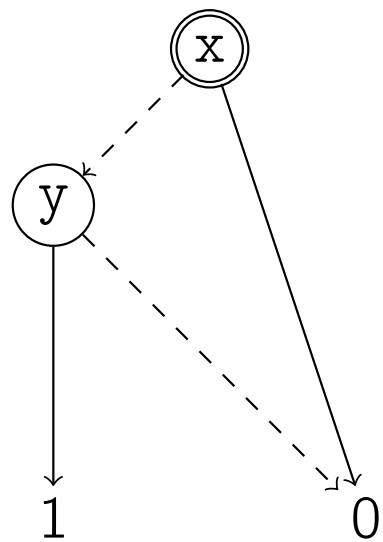
1. Поменять местами терминальные узлы

Пример

$\mathcal{D}^{\mathrm{x}\vee\neg\mathrm{y}}$:



$\neg\mathcal{D}^{\mathrm{x}\vee\neg\mathrm{y}}$:



ROBDD: подстановка константы

$\varphi[\mathrm{x}/e]$ — формула, получающаяся из φ подстановкой выражения e на место каждого вхождения переменной x

Дано: ROBDD $\mathcal{D}^{\varphi}$ над $x_1, \ldots, x_n$, значение $b \in \{0, 1\}$

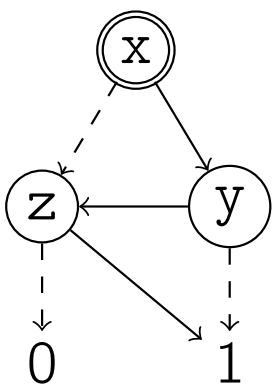
Требуется построить ROBDD $\mathcal{D}^{\varphi}[\mathrm{x}/b] = \mathcal{D}^{\varphi[\mathrm{x}/b]}$ над $x_1, \ldots, x_n$

Алгоритм и тут устроен несложно:

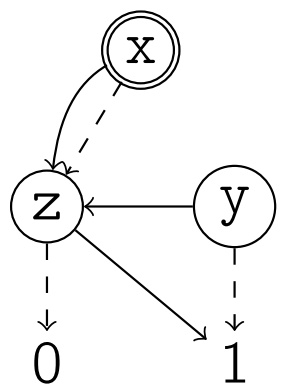
1. Для каждого узла v, такого что $var(v) = \mathrm{x}$, перенаправить входящие дуги в узел $v[b]$
2. Привести получившуюся OBDD

Пример ($\varphi = \mathrm{z}\,\&(\neg\mathrm{x} \vee \mathrm{y}) \vee \mathrm{x}\,\&\,\neg\mathrm{y}$)

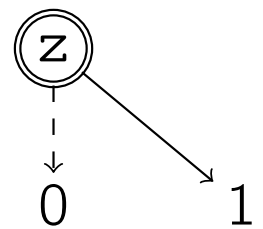
$\mathcal{D}^{\varphi}$:



OBDD $\mathcal{D} \sim \varphi[\mathrm{y}/1]$:



$\mathcal{D}^{\varphi}[\mathrm{y}/1]$:



ROBDD: применение двуместной операции

Дано: ROBDD $\mathcal{D}^\varphi$, $\mathcal{D}^\psi$ над $x_1, \ldots, x_n$, двуместная булева операция $\circ$

Требуется построить ROBDD $\mathcal{D}^\varphi \circ \mathcal{D}^\psi = \mathcal{D}^{\varphi \circ \psi}$ над $x_1, \ldots, x_n$

Алгоритм устроен сложнее всех предыдущих, но тоже достаточно просто:

- Если $n = 0$, то обе формулы φ, ψ — это константы 0, 1, и результат — простейшая диаграмма $\mathcal{D}^{\varphi \circ \psi}$
- Иначе:
  - *Рекурсивно* построить диаграммы $\mathcal{D}_0 = \mathcal{D}^\varphi[\mathrm{x}_1/0] \circ \mathcal{D}^\psi[\mathrm{x}_1/0]$ и $\mathcal{D}_1 = \mathcal{D}^\varphi[\mathrm{x}_1/1] \circ \mathcal{D}^\psi[\mathrm{x}_1/1]$ над $x_2, \ldots, x_n$
  - Объединить $\mathcal{D}_0$ и $\mathcal{D}_1$, считая все внутренние узлы попарно различными, добавить узел x_1, объявить его корнем и направить из него дуги $\xrightarrow{0}$, $\xrightarrow{1}$ в бывшие корни диаграмм $\mathcal{D}_0$, $\mathcal{D}_1$ соответственно
  - Привести полученную OBDD

А почему это работает?

ROBDD: построение по формуле

Формулу φ *над* $\neg$ *и двуместными операциями* можно трактовать как схему применения операций к функциям, реализуемым простейшими формулами 0, 1, $\mathtt{x}_i$

Диаграмму $\mathcal{D}^{\varphi}$ можно получить как результат применения операций согласно той же схеме к соответствующим простейшим диаграммам $\mathcal{D}^0$, $\mathcal{D}^1$, $\mathcal{D}^{\mathtt{x}_i}$

Например, $\mathcal{D}^{\mathtt{x}\,\&\,\neg\mathtt{y}\vee\mathtt{z}} = \mathcal{D}^{\mathtt{x}}\,\&\,\neg\mathcal{D}^{\mathtt{y}} \vee \mathcal{D}^{\mathtt{z}}$

ROBDD: производные операции

На практике к ROBDD применяются и другие удобные «производные» операции:

- Для формулы φ: $\exists \mathrm{x}\ \varphi = \varphi[\mathrm{x}/0] \vee \varphi[\mathrm{x}/1]$
  Для диаграммы: $\exists \mathrm{x}\ \mathcal{D}^{\varphi} = \mathcal{D}^{\exists \mathrm{x}\ \varphi}$
- Для формулы φ: $\forall \mathrm{x}\ \varphi = \varphi[\mathrm{x}/0] \,\&\, \varphi[\mathrm{x}/1]$
  Для диаграммы: $\forall \mathrm{x}\ \mathcal{D}^{\varphi} = \mathcal{D}^{\forall \mathrm{x}\ \varphi}$
- Для формулы φ: $\mathit{relprod}(\varphi, \psi, \mathrm{y}_1, \ldots, \mathrm{y}_k) = \exists \mathrm{y}_1\ \ldots \exists \mathrm{y}_k\ (\varphi \,\&\, \psi)$
  Для диаграммы: $\mathit{relprod}(\mathcal{D}^{\varphi}, \mathcal{D}^{\psi}, \mathrm{y}_1, \ldots, \mathrm{y}_k) = \mathcal{D}^{\mathit{relprod}(\varphi, \psi, \mathrm{y}_1, \ldots, \mathrm{y}_k)}$
  - Пусть формулой φ задаётся двуместное отношение R_1 над комплектами переменных $\widetilde{x}$ и $\widetilde{y}$, а формулой ψ — одноместное отношение R_2 над комплектом $\widetilde{y}$ или двуместное над $\widetilde{y}$ и $\widetilde{z}$
    Тогда $\mathit{relprod}(\varphi, \psi, \widetilde{y})$ задаёт отношение
    - $\exists \widetilde{y}(R_1(\widetilde{x}, \widetilde{y}) \,\&\, R_2(\widetilde{y}))$: множество всех $\widetilde{x}$, входящих в отношение R_1 с хотя бы одним $\widetilde{y}$ из R_2 — или
    - $\exists \widetilde{y}(R_1(\widetilde{x}, \widetilde{y}) \,\&\, R_2(\widetilde{y}, \widetilde{z}))$: множество всех пар $(\widetilde{x}, \widetilde{z})$, соединяющихся посредством R_1 и R_2 через хотя бы один $\widetilde{y}$

ROBDD: производные операции

Задача QBF (выполнимости квантифицированных булевых формул): для заданной произвольной формулы вида $Q_1 x_1 \dots Q_n x_n(\varphi)$, где $Q_i \in \{\exists, \forall\}$ и φ — КНФ над $x_1, \dots, x_n$, проверить соотношение $Q_1 x_1 \dots Q_n x_n(\varphi) \not\sim 0$

Эта задача **весьма трудна**: является PSPACE-полной

Но её можно решить при помощи малого числа операций над BDD:

- Построить ROBDD по заданной формуле (с кванторами)
- Проверить изоморфизм полученной BDD и $\mathcal{D}^0$

Сложность работы с ROBDD «скрыта» в возрастании их размера при применении операций:

- При применении одной операции размер возрастает несильно (*полиномиально с низкой степенью*)
- При многократном применении операций размер может возрастать весьма существенно

Но хотя использование ROBDD и неэффективно в теории, оно всё же считается эффективным на практике: при должном выборе порядка переменных нередко получаются достаточно компактные диаграммы

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 26

Символьный алгоритм model checking для CTL (начало)

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Основная идея символьного алгоритма

Символьным алгоритмом решения задачи обычно называют решающий алгоритм, в котором для основных («трудоёмких») структур данных используются символьные представления в предположении о том, что работать с такими представлениями можно существенно более эффективно, чем с явными

Алгоритм model checking для CTL линеен относительно как размера формулы, так и размера модели

Но на практике возникает необходимость применять алгоритм к настолько большим моделям, что даже такая (казалось бы совсем невысокая) сложность оказывается неприемлемой

Основная идея символьного алгоритма

Например, если в системе независимо асинхронно выполняется n одинаковых процессов, в каждом из которых k состояний, то суммарное число состояний системы (k^n) экспоненциально относительно числа процессов

Такой эффект экспоненциального роста числа состояний относительно параметров системы называется комбинаторным взрывом числа состояний, и это основная проблема, возникающая в области model checking на практике

Символьный алгоритм model checking для CTL — это базовый алгоритм, адаптированный для работы с моделями Крипке в символьном представлении

Стандартное символьное представление (например, в виде ROBDD) множества или отношения X далее будем обозначать записью Φ_X

Представление Φ множества над переменными $\vec{x}$ будем обозначать записью $\Phi(\vec{x})$, и особо для двуместных отношений над комплектами переменных $\vec{x}$ для первого аргумента и $\vec{y}$ для второго — $\Phi(\vec{x}, \vec{y})$

Основная идея символьного алгоритма

Записью $\vec{x}$ будем обозначать набор переменных $x_1, \ldots, x_m$ для некоторого m, *заданного контекстом*

$\exists\vec{x}$ — так будем сокращать запись $\exists x_1 \exists x_2 \ldots \exists x_m$

Основные операции над множествами и отношениями, используемые в базовом алгоритме, естественно переформулируются на языке символьных представлений:

- Объединение множеств: $A = B \cup C \mapsto \Phi_A = \Phi_B \vee \Phi_C$
- Пересечение множеств: $A = B \cap C \mapsto \Phi_A = \Phi_B \,\&\, \Phi_C$
- Разность множеств: $A = B \setminus C \mapsto \Phi_A = \Phi_B \,\&\, \neg\Phi_C$
- Образ отношения: $A = \{y \mid \exists x : (x, y) \in R\} \mapsto \Phi_A(\vec{y}) = \exists\vec{x}\Phi_R(\vec{x}, \vec{y})$
- Прообраз отношения: $A = \{x \mid \exists y : (x, y) \in R\} \mapsto$ $\Phi_A(\vec{x}) = \exists\vec{y}\Phi_R(\vec{x}, \vec{y})$

Для «стыковки» переменных множеств понадобится также уметь переименовывать переменные в формулах: $\Phi[\vec{x}/\vec{y}] = \Phi[x_1/y_1, \ldots, x_m/y_m]$

Символьный алгоритм (начало)

Дано:

- Стандартное символьное представление конечной модели Крипке $M = (S, S_0, \to, L)$ над $\{p_1, \ldots, p_k\}$:
  $\mathfrak{M} = (\Phi_S(\vec{x}), \Phi_{S_0}(\vec{x}), \Phi_{\to}(\vec{x}, \vec{y}), \Phi_{p_1}(\vec{x}), \ldots, \Phi_{p_k}(\vec{x}))$
- Ctl-формула φ

Требуется: проверить справедливость соотношения $M \models \varphi$

Базовый алгоритм, основная процедура:

1. Вычислить множество $X = Sat(M, \varphi) = \mathfrak{P}_{sat}(M, \varphi)$
2. Проверить включение $S_0 \subseteq X$
   - То есть проверить соотношение $S_0 \setminus X = \emptyset$
3. Вернуть результат проверки предыдущего пункта

Символьный алгоритм, основная процедура:

1. Вычислить $\Phi_X(\vec{x}) = \mathfrak{F}_{sat}(\mathfrak{M}, \varphi)$ ($\mathfrak{F}_{sat}$ описана далее)
2. Проверить соотношение $\Phi_{S_0} \,\&\, \neg\Phi_X \sim \Phi_\emptyset$
3. Вернуть результат проверки предыдущего пункта

Символьный алгоритм (начало)

Базовый алгоритм, процедура $\mathfrak{P}_{sat}(M, \varphi)$:

1. Используя известные равносильности, преобразовать φ в равносильную *упрощённую* формулу ψ в базисе **EX**, **EG**, **EU**:
$$\psi \ ::= \ \mathbb{t} \ | \ p \ | \ \psi \,\&\, \psi \ | \ \neg\psi \ | \ \mathbf{EX}\psi \ | \ \mathbf{EG}\psi \ | \ \mathbf{E}(\psi \mathbf{U} \psi)$$

2. $\mathfrak{P}_{sat}(M, \varphi) = \mathfrak{P}'_{sat}(M, \psi)$

Символьный алгоритм, процедура $\mathfrak{F}_{sat}(\mathfrak{M}, \varphi)$:

1. Дословно как выше

2. $\mathfrak{F}_{sat}(\mathfrak{M}, \varphi) = \mathfrak{F}'_{sat}(\mathfrak{M}, \psi)$

Символьный алгоритм (начало)

Базовый алгоритм, процедура $\mathfrak{P}'_{sat}(M, \varphi)$:

- Если $\varphi = \mathbb{t}$, то $\mathfrak{P}'_{sat}(M, \varphi) = S$
- Если $\varphi = p \in \mathrm{AP}$, то $\mathfrak{P}'_{sat}(M, \varphi) = \{s \mid s \in S,\ p \in L(s)\}$
- Если $\varphi = \psi_1 \,\&\, \psi_2$, то $\mathfrak{P}'_{sat}(M, \varphi) = \mathfrak{P}'_{sat}(M, \psi_1) \cap \mathfrak{P}'_{sat}(M, \psi_2)$
- Если $\varphi = \neg\psi$, то $\mathfrak{P}'_{sat}(M, \varphi) = S \setminus \mathfrak{P}'_{sat}(M, \psi)$
- Если $\varphi = \mathbf{EX}\psi$, то $\mathfrak{P}'_{sat}(M, \varphi) = \mathfrak{P}_{\mathbf{EX}}(M, \psi)$
- Если $\varphi = \mathbf{EG}\psi$, то $\mathfrak{P}'_{sat}(M, \varphi) = \mathfrak{P}_{\mathbf{EG}}(M, \psi)$
- Если $\varphi = \mathbf{E}(\psi_1 \mathbf{U} \psi_2)$, то $\mathfrak{P}'_{sat}(M, \varphi) = \mathfrak{P}_{\mathbf{EU}}(M, \psi_1, \psi_2)$

Символьный алгоритм, процедура $\mathfrak{F}'_{sat}(\mathfrak{M}, \varphi)$:

- Если $\varphi = \mathbb{t}$, то $\mathfrak{F}'_{sat}(\mathfrak{M}, \varphi) = \Phi_S$
- Если $\varphi = p \in \mathrm{AP}$, то $\mathfrak{F}'_{sat}(\mathfrak{M}, \varphi) = \Phi_p$
- Если $\varphi = \psi_1 \,\&\, \psi_2$, то $\mathfrak{F}'_{sat}(\mathfrak{M}, \varphi) = \mathfrak{F}'_{sat}(\mathfrak{M}, \psi_1) \,\&\, \mathfrak{F}'_{sat}(\mathfrak{M}, \psi_2)$
- Если $\varphi = \neg\psi$, то $\mathfrak{F}'_{sat}(\mathfrak{M}, \varphi) = \Phi_S \,\&\, \neg\mathfrak{F}'_{sat}(\mathfrak{M}, \psi)$
- Если $\varphi = \mathbf{EX}\psi$, то $\mathfrak{F}'_{sat}(\mathfrak{M}, \varphi) = \mathfrak{F}_{\mathbf{EX}}(\mathfrak{M}, \psi)$
- Если $\varphi = \mathbf{EG}\psi$, то $\mathfrak{F}'_{sat}(\mathfrak{M}, \varphi) = \mathfrak{F}_{\mathbf{EG}}(\mathfrak{M}, \psi)$
- Если $\varphi = \mathbf{E}(\psi_1 \mathbf{U} \psi_2)$, то $\mathfrak{F}'_{sat}(\mathfrak{M}, \varphi) = \mathfrak{F}_{\mathbf{EU}}(\mathfrak{M}, \psi_1, \psi_2)$

Символьный алгоритм (начало)

Базовый алгоритм, множество $Pre(M, X) = \{s \mid \exists s' : s \to s',\ s' \in X\}$

Символьный алгоритм: $\mathfrak{F}_{pre}(\mathfrak{M}, \Phi_X) = \exists \vec{y}(\Phi_{\to} \,\&\, \Phi_X[\vec{x}/\vec{y}])$

Базовый алгоритм, процедура $\mathfrak{P}_{\mathbf{EX}}(M, \varphi)$:

- Вычислить $X = \mathfrak{P}'_{sat}(M, \varphi)$
- Вернуть множество $Pre(M, X)$

Символьный алгоритм, процедура $\mathfrak{F}_{\mathbf{EX}}(\mathfrak{M}, \varphi)$:

- Вычислить $\Phi_X = \mathfrak{F}'_{sat}(\mathfrak{M}, \varphi)$
- Вернуть $\mathfrak{F}_{pre}(\mathfrak{M}, \Phi_X)$

Перед описанием процедур $\mathfrak{F}_{\mathbf{EU}}$ и $\mathfrak{F}_{\mathbf{EG}}$ символьного алгоритма сформулируем ещё несколько понятий и утверждений, позволяющих устроить эти процедуры «более умно»

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 27

Преобразователи предикатов
и их неподвижные точки

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Преобразователи предикатов

В базовом алгоритме вычисление множеств $Sat(M, \mathbf{EX}\varphi)$, $Sat(M, \mathbf{EG}\varphi)$ и $Sat(M, \mathbf{E}(\varphi_1 \mathbf{U} \varphi_2))$ при помощи процедур $\mathfrak{P}_{\mathbf{EX}}$, $\mathfrak{P}_{\mathbf{EU}}$, $\mathfrak{P}_{\mathbf{EG}}$ основывалось на преобразовании множеств состояний — или, по-другому, одноместных предикатов на множестве S

Способ преобразования предиката на S можно представить как преобразователь предикатов на S: функцию вида $\mathbb{f} : 2^S \to 2^S$

В частности, темпоральную комбинацию **EX** в контексте заданной модели M можно расценивать как преобразователь предикатов:

$$Sat(M, \mathbf{EX}\varphi) = \{s \mid \exists s' : s \to s',\ s' \in Sat(M, \varphi)\}$$

$$Sat_M(\mathbf{EX}\varphi) = \{s \mid \exists s' : s \to s',\ s' \in Sat_M(\varphi)\}$$

$$\mathbf{EX}_M(Sat_M(\varphi)) = \{s \mid \exists s' : s \to s',\ s' \in Sat_M(\varphi)\}$$

$$\mathbf{EX}_M(A) = \{s \mid \exists s' : s \to s',\ s' \in A\}$$

Преобразователи предикатов

Совокупность $(2^S, \subseteq)$ — это **решётка**, в которой:

- Точная верхняя грань $sup(A, B)$ множеств A и B — это их объединение
- Точная нижняя грань $inf(A, B)$ множеств A и B — это их пересечение
- Наибольший элемент — это можество S
- Наименьший элемент — это $\emptyset$

Таким образом, преобразователь предикатов может расцениваться как функция преобразования элементов решётки, и из этого далее будут вытекать терминология и свойства преобразователей

$\subseteq$ в такой решётке — это отношение нестрогого частичного порядка, и для него будем применять соответствующую терминологию:

- $A \subseteq B \Leftrightarrow A$ не больше B и B не меньше A
- $A \subset B \Leftrightarrow A$ меньше B и B больше A
- ...

Преобразователи предикатов

Преобразователь предикатов $\mathbb{f} : 2^S \to 2^S$ называется

- монотонным, если для любых предикатов A, B справедлива импликация
$$A \subseteq B \Rightarrow \mathbb{f}(A) \subseteq \mathbb{f}(B)$$
- $\cup$-непрерывным, если для любой бесконечной монотонно неубывающей последовательности предикатов
$$A_1 \subseteq A_2 \subseteq \ldots$$
верно $\mathbb{f}(\bigcup\limits_{i=1}^{\infty} A_i) = \bigcup\limits_{i=1}^{\infty} \mathbb{f}(A_i)$
- $\cap$-непрерывным, если для любой бесконечной монотонно невозрастающей последовательности предикатов
$$A_1 \supseteq A_2 \supseteq \ldots$$
верно $\mathbb{f}(\bigcap\limits_{i=1}^{\infty} A_i) = \bigcap\limits_{i=1}^{\infty} \mathbb{f}(A_i)$

Преобразователи предикатов

Лемма. Любой монотонный преобразователь предикатов $\mathbb{f}$ на конечном множестве S $\cup$-непрерывен и $\cap$-непрерывен

Доказательство.

Рассмотрим бесконечную последовательность предикатов $A_1 \subseteq A_2 \subseteq \ldots$

Так как множество S конечно, для некоторого k верно $A_k = A_{k+1} = A_{k+2} = \ldots$

Так как $A_1 \subseteq \cdots \subseteq A_k$, то верно и $\bigcup\limits_{i=1}^{k} A_i = A_k$

Значит, $\bigcup\limits_{i=1}^{\infty} A_i = A_k$

Так как $\mathbb{f}$ монотонен, верно $\mathbb{f}(A_1) \subseteq \cdots \subseteq \mathbb{f}(A_k) = \mathbb{f}(A_{k+1}) = \ldots$, и аналогично верно $\bigcup\limits_{i=1}^{\infty} \mathbb{f}(A_i) = \mathbb{f}(A_k)$

Следовательно, $\mathbb{f}(\bigcup\limits_{i=1}^{\infty} A_i) = \mathbb{f}(A_k) = \bigcup\limits_{i=1}^{\infty} \mathbb{f}(A_i)$, то есть $\mathbb{f}$ $\cup$-непрерывен

$\cap$-непрерывность обосновывается аналогично ▼

Неподвижные точки

Неподвижной точкой преобразователя $\mathbb{f} : 2^S \to 2^S$ называется предикат A, такой что $f(A) = A$

Неподвижная точка A преобразователя $\mathbb{f}$ называется наименьшей ($A = \mu Z.\mathbb{f}(Z)$), если она наименьшая по включению среди всех неподвижных точек $\mathbb{f}$, и наибольшей ($A = \nu Z.\mathbb{f}(Z)$), если она наибольшая по включению среди всех неподвижных точек

$\mathbb{f}^i(A)$ — так обозначим i-кратное применение преобразователя $\mathbb{f}$ к предикату A:

- $\mathbb{f}^0(A) = A$
- $\mathbb{f}^i(A) = \mathbb{f}(\mathbb{f}^{i-1}(A))$, если $i > 0$

Неподвижные точки

Лемма. Для любого монотонного $\cup$-непрерывного преобразователя $\mathbb{f}$ верно $\mu Z.\mathbb{f}(Z) = \bigcup\limits_{i=0}^{\infty} \mathbb{f}^i(\emptyset)$

Доказательство. Пусть $A = \bigcup\limits_{i=0}^{\infty} \mathbb{f}^i(\emptyset)$

По определению наименьшей неподвижной точки, достаточно обосновать два факта:

1. $\mathbb{f}(A) = A$
2. Если $\mathbb{f}(B) = B$, то $A \subseteq B$

1. По $\cup$-непрерывности $\mathbb{f}$: $\mathbb{f}(A) = \bigcup_{i=1}^{\infty} \mathbb{f}^i(\emptyset) = \mathbb{f}^0(\emptyset) \cup \bigcup_{i=1}^{\infty} \mathbb{f}^i(\emptyset) = A$

2. Верно $\mathbb{f}^0(\emptyset) = \emptyset \subseteq B$

По монотонности $\mathbb{f}$, для любого $i \in \{0, 1, 2, \dots\}$ верно следующее: если $\mathbb{f}^i(\emptyset) \subseteq B$, то верно и $\mathbb{f}^{i+1}(\emptyset) = \mathbb{f}(\mathbb{f}^i(\emptyset)) \subseteq \mathbb{f}(B) = B$

Значит, для любого $i \in \{0, 1, 2, \dots\}$ верно $\mathbb{f}^i(\emptyset) \subseteq B$, и следовательно, $\bigcup\limits_{i=0}^{\infty} \mathbb{f}^i(\emptyset) \subseteq B$ ▼

Неподвижные точки

Лемма. **Для любого монотонного $\cap$-непрерывного преобразователя $\mathbb{f}$ на S верно $\nu Z.\mathbb{f}(Z) = \bigcap_{i=0}^{\infty} \mathbb{f}^i(S)$**

Доказательство. Аналогично доказательству предыдущей леммы

Лемма. **Для любого монотонного преобразователя и любого i, $i \in \mathbb{N}_0$, верно $\mathbb{f}^i(\emptyset) \subseteq \mathbb{f}^{i+1}(\emptyset)$**

Доказательство.

$\mathbb{f}^0(\emptyset) = \emptyset \subseteq \mathbb{f}^1(\emptyset)$

Если $\mathbb{f}^{i-1}(\emptyset) \subseteq \mathbb{f}^i(\emptyset)$ $(i \geq 1)$, то $\mathbb{f}^i(\emptyset) = \mathbb{f}(\mathbb{f}^{i-1}(\emptyset)) \subseteq \mathbb{f}(\mathbb{f}^i(\emptyset)) = \mathbb{f}^{i+1}(\emptyset)$

Значит, *согласно принципу математической индукции*, для любого $i \in \{0, 1, 2, \dots\}$ верно $\mathbb{f}^i(\emptyset) \subseteq \mathbb{f}^{i+1}(\emptyset)$ ▼

Лемма. **Для любого монотонного преобразователя на S и любого i, $i \in \mathbb{N}_0$, верно $\mathbb{f}^i(S) \supseteq \mathbb{f}^{i+1}(S)$**

Доказательство. Аналогично доказательству предыдущей леммы

Неподвижные точки

Лемма. **Для любого монотонного преобразователя $\mathbb{f}$ на конечном множестве S существует k, $k \in \mathbb{N}_0$, такое что $\mathbb{f}^k(\emptyset) = \mathbb{f}^{k+1}(\emptyset)$**

Доказательство. *Предположим от противного*, что это не так

По доказанной ранее лемме, для любого $i \in \{0, 1, \dots\}$ верно $\mathbb{f}^i(\emptyset) \subseteq \mathbb{f}^{i+1}(\emptyset)$

По предположению, $\mathbb{f}^i(\emptyset) \neq \mathbb{f}^{i+1}(\emptyset)$, а значит, $\mathbb{f}^i(\emptyset) \subset \mathbb{f}^{i+1}(\emptyset)$

Следовательно, $|\mathbb{f}^0(\emptyset)| < |\mathbb{f}^1(\emptyset)| < |\mathbb{f}^2(\emptyset)| < \dots$

Но тогда существует m, такое что $|\mathbb{f}^m(\emptyset)| > |S|$, что *противоречит* включению $\mathbb{f}^m(\emptyset) \subseteq S$ ▼

Лемма. **Для любого монотонного преобразователя $\mathbb{f}$ на конечном множестве S существует k, $k \in \mathbb{N}_0$, такое что $\mathbb{f}^k(S) = \mathbb{f}^{k+1}(S)$**

Доказательство. Аналогично доказательству предыдущей леммы

Неподвижные точки

Лемма. **Для любого преобразователя $\mathbb{f}$, любого k, $k \in \mathbb{N}_0$, и любого предиката A верно следующее: если $\mathbb{f}^k(A) = \mathbb{f}^{k+1}(A)$, то для любого m, $m \in \mathbb{N}$, верно $\mathbb{f}^k(A) = \mathbb{f}^{k+m}(A)$**

Доказательство (индукцией по m).

База ($m = 1$): верно по условию леммы

Индуктивный переход: если $\mathbb{f}^k(A) = \mathbb{f}^{k+(m-1)}(A)$, то
$\mathbb{f}^{k+m}(A) = \mathbb{f}^{k+1}(\mathbb{f}^{m-1}(A)) = \mathbb{f}^k(\mathbb{f}^{m-1}(A)) = \mathbb{f}^{k+m-1}(A) = \mathbb{f}^k(A)$ ▼

Неподвижные точки

Объединив результаты всех предложенных лемм, можно легко получить следующие теорему и процедуру вычисления наименьшей неподвижной точки ($\mathfrak{P}_{lfp}$) для преобразователя предикатов на конечном множестве S

Теорема (о поиске наименьшей неподвижной точки). Для любого монотонного преобразователя $\mathbb{f}$ на конечном множестве S существует k, $k \in \mathbb{N}_0$, такое что

$$\mathbb{f}^0(\emptyset) \subset \mathbb{f}^1(\emptyset) \subset \cdots \subset \mathbb{f}^k(\emptyset) = \mathbb{f}^{k+1}(\emptyset),$$

и верно $\mu Z.\mathbb{f}(Z) = \mathbb{f}^k(\emptyset)$

Процедура $\mathfrak{P}_{lfp}(M, \mathbb{f})$:

- Положить $X_0 = \emptyset$
- Последовательно для $i \in \{1, 2, \dots\}$:
  - Вычислить $X_i = \mathbb{f}(X_{i-1})$
  - Если $X_i = X_{i-1}$, то завершить процедуру и вернуть X_i

Для использования преобразователей в качестве аргументов процедур требуется подходящее представление — оно будет введено чуть позже

Неподвижные точки

Символьным представлением преобразователя $\mathbb{f}$ назовём отображение $\mathbb{f}_\mathfrak{s}$ множества символьных представлений предикатов в него же, такое что $\mathbb{f}_\mathfrak{s}(\Phi_A) = \Phi_{\mathbb{f}(A)}$

Процедура вычисления наименьшей неподвижной точки очевидным образом переформулируется в терминах символьных представлений

Процедура $\mathfrak{F}_{lfp}(\mathfrak{M}, \mathbb{f}_\mathfrak{s})$:

- Положить $\Phi^0 = \Phi_\emptyset$
- Последовательно для $i \in \{1, 2, \dots\}$:
  - Вычислить $\Phi^i = \mathbb{f}_\mathfrak{s}(\Phi^{i-1})$
  - Если $\Phi^i \sim \Phi^{i-1}$, то завершить процедуру и вернуть Φ^i

Неподвижные точки

Аналогичные теорема и алгоритмы получаются и для наибольшей неподвижной точки

Теорема (о поиске наибольшей неподвижной точки). Для любого монотонного преобразователя $\mathbb{f}$ на конечном множестве S существует k, $k \in \mathbb{N}_0$, такое что

$$\mathbb{f}^0(S) \supset \mathbb{f}^1(S) \supset \cdots \supset \mathbb{f}^k(S) = \mathbb{f}^{k+1}(S),$$

и верно $\nu Z.\mathbb{f}(Z) = \mathbb{f}^k(S)$

Процедура $\mathfrak{P}_{gfp}(M, \mathbb{f})$:

- Положить $X_0 = S$
- Последовательно для $i \in \{1, 2, \dots\}$:
  - Вычислить $X_i = \mathbb{f}(X_{i-1})$
  - Если $X_i = X_{i-1}$, то завершить процедуру и вернуть X_i

Процедура $\mathfrak{F}_{gfp}(\mathfrak{M}, \mathbb{f}_\mathfrak{s})$:

- Положить $\Phi^0 = \Phi_S$
- Последовательно для $i \in \{1, 2, \dots\}$:
  - Вычислить $\Phi^i = \mathbb{f}_\mathfrak{s}(\Phi^{i-1})$
  - Если $\Phi^i \sim \Phi^{i-1}$, то завершить процедуру и вернуть Φ^i

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 28

Символьный алгоритм model checking для CTL (окончание)

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Преобразователь $\mathbf{EX}_M$

Каждой ctl-формуле φ для предзаданной модели Крипке M можно сопоставить предикат, задающий множество всех состояний M, в которых выполнена φ: $Sat_M(\varphi) = Sat(M, \varphi)$

Этот предикат можно представить символьно: $\Phi_\varphi^M = \Phi_{Sat_M(\varphi)}$

Для заданной модели Крипке комбинация **EX** может расцениваться как преобразователь таких предикатов:

$$\mathbf{EX}_M(Sat_M(\varphi)) = Sat_M(\mathbf{EX}\varphi)$$

Согласно устройству процедуры $\mathfrak{P}_{\mathbf{EX}}$, преобразователь $\mathbf{EX}_M$ можно задать так:

$$\mathbf{EX}_M(Z) = Pre(M, Z)$$

Преобразователь $\mathbb{f}$, задающийся равенством $\mathbb{f}(Z) = E$ для произвольного предиката Z и выражения E (вообще говоря зависящего от Z), можно представить записью $\lambda Z.E$: это запись функции, принимающей на вход значение Z и возвращающей значение выражения E для этого значения

Тогда, в частности, $\mathbf{EX}_M = \lambda Z.Pre(M, Z)$, и символьное представление этого преобразователя можно устроить так: $\mathbf{EX}_{\mathfrak{M}} = \lambda Z.\mathfrak{F}_{pre}(\mathfrak{M}, Z)$

Преобразователь $\mathbf{EX}_M$

Лемма. **Для любых модели Крипке M и предиката C преобразователь $\mathbb{f} = \lambda Z.\, C \cap \mathbf{EX}_M(Z)$ является монотонным**

Доказательство.

Рассмотрим предикаты A и B, такие что $A \subseteq B$, и покажем, что $\mathbb{f}(A) \subseteq \mathbb{f}(B)$, то есть что для любого состояния $s \in \mathbb{f}(A)$ верно $s \in \mathbb{f}(B)$

Так как $s \in \mathbb{f}(A)$, верно $s \in C$ и $s \in \mathbf{EX}_M(A)$

$s \in \mathbf{EX}_M(A)$ означает, что существует состояние s', такое что $s' \in A$ и $s \to s'$

Так как $A \subseteq B$, верно и $s' \in B$

Значит, $s \in \mathbf{EX}_M(B)$, и следовательно, $s \in C \cap \mathbf{EX}_M(B) = \mathbb{f}(B)$ ▼

Преобразователь $\mathbf{EG}_M$

Лемма. **Для любых модели Крипке M и ctl-формулы φ предикат $Sat_M(\mathbf{EG}\varphi)$ является неподвижной точкой преобразователя $\lambda Z.Sat_M(\varphi) \cap \mathbf{EX}_M(Z)$**

Доказательство.

По определению неподвижной точки, достаточно показать равенство

$$Sat_M(\mathbf{EG}\varphi) = Sat_M(\varphi) \cap \mathbf{EX}_M(Sat_M(\mathbf{EG}\varphi))$$

По определениям Sat_M и $\mathbf{EX}_M$, достаточно показать равносильность

$$M, s \models \mathbf{EG}\varphi \Leftrightarrow M, s \models \varphi \text{ и } M, s \models \mathbf{EXEG}\varphi$$

По семантике ctl-формул,

- «$M, s \models \mathbf{EG}\varphi$» $\Leftrightarrow$ в M из s исходит хотя бы один бесконечный путь $s_1, s_2, \ldots$, такой что $M, s_1 \models \varphi$, $M, s_2 \models \varphi$, ...
- «$M, s \models \varphi$ и $M, s \models \mathbf{EXEG}\varphi$» $\Leftrightarrow$ $M, s \models \varphi$ и в M из s исходит хотя бы один бесконечный путь $s_1, s_2, \ldots$, такой что $M, s_2 \models \varphi$, $M, s_3 \models \varphi$, ...

Легко видеть, что последние два пункта равносильны ▼

Преобразователь $\mathbf{EG}_M$

Лемма. **Для любых конечной модели Крипке M и ctl-формулы φ предикат $Sat_M(\mathbf{EG}\varphi)$ является наибольшей неподвижной точкой преобразователя $\lambda Z.Sat_M(\varphi) \cap \mathbf{EX}_M(Z)$**

Доказательство. Можете попробовать самостоятельно

Из последней леммы естественно вытекает альтернативный (по сравнению с базовым алгоритмом) вариант процедуры $\mathfrak{P}_{\mathbf{EG}}(M, \varphi)$:

- Вычислить $X = Sat(M, \varphi) = \mathfrak{P}'_{sat}(M, \varphi)$
- Вернуть предикат $\mathfrak{P}_{gfp}(M, \lambda Z.X \cap \mathbf{EX}_M(Z))$

Эту процедуру несложно представить символьно ($\mathfrak{F}_{\mathbf{EG}}(\mathfrak{M}, \varphi)$):

- Вычислить $\Phi_X = \mathfrak{F}'_{sat}(\mathfrak{M}, \varphi)$
- Вернуть представление $\mathfrak{F}_{gfp}(\mathfrak{M}, \lambda Z.\Phi_X \,\&\, \mathbf{EX}_{\mathfrak{M}}(Z))$

Преобразователь $\mathbf{EU}_M$

Лемма. **Для любых модели Крипке M и ctl-формул φ и ψ предикат $Sat_M(\mathbf{E}(\varphi\mathbf{U}\psi))$ является неподвижной точкой преобразователя $\lambda Z.Sat_M(\psi) \cup (Sat_M(\varphi) \cap \mathbf{EX}_M(Z))$**

Доказательство.

По определению неподвижной точки, достаточно показать равенство

$$Sat_M(\mathbf{E}(\varphi\mathbf{U}\psi)) = Sat_M(\psi) \cup (Sat_M(\varphi) \cap \mathbf{EX}_M(Sat_M(\mathbf{E}(\varphi\mathbf{U}\psi))))$$

По определениям Sat_M и $\mathbf{EX}_M$, достаточно показать равносильность

$$M, s \models \mathbf{E}(\varphi\mathbf{U}\psi) \Leftrightarrow M, s \models \psi \text{ или } (M, s \models \varphi \text{ и } M, s \models \mathbf{EXE}(\varphi\mathbf{U}\psi))$$

Аналогично доказательству такой же леммы для $\mathbf{EG}\varphi$, легко видеть, что эта равносильность действительно справедлива ▼

Лемма. **Для любых <u>конечной</u> модели Крипке M и ctl-формул φ и ψ предикат $Sat_M(\mathbf{E}(\varphi\mathbf{U}\psi))$ является <u>наименьшей</u> неподвижной точкой преобразователя $\lambda Z.Sat_M(\psi) \cup (Sat_M(\varphi) \cap \mathbf{EX}_M(Z))$**

Доказательство. Можете попробовать самостоятельно

Преобразователь $\mathbf{EU}_M$

Из последней леммы естественно вытекает альтернативный (по сравнению с базовым алгоритмом) вариант процедуры $\mathfrak{P}_{\mathbf{EU}}(M, \varphi, \psi)$:

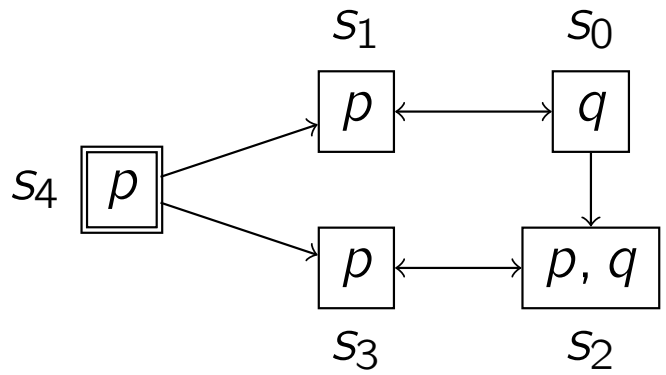
- Вычислить $X = Sat(M, \varphi) = \mathfrak{P}'_{sat}(M, \varphi)$ и $Y = Sat'(m, \psi) = \mathfrak{P}'_{sat}(M, \psi)$
- Вернуть предикат $\mathfrak{P}_{lfp}(M, \lambda Z.Y \cup (X \cap \mathbf{EX}_M(Z)))$

Эту процедуру несложно представить символьно ($\mathfrak{F}_{\mathbf{EU}}(\mathfrak{M}, \varphi, \psi)$):

- Вычислить $\Phi_X = \mathfrak{F}'_{sat}(\mathfrak{M}, \varphi)$ и $\Phi_Y = \mathfrak{F}'_{sat}(\mathfrak{M}, \psi)$
- Вернуть представление $\mathfrak{F}_{lfp}(\mathfrak{M}, \lambda Z.\Phi_Y \vee (\Phi_X \,\&\, \mathbf{EX}_{\mathfrak{M}}(Z)))$

Символьный алгоритм: пример

M:



$$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$$

Для начала проиллюстрируем модифицированный базовый алгоритм

Начало — такое же, как в исходном базовом алгоритме:

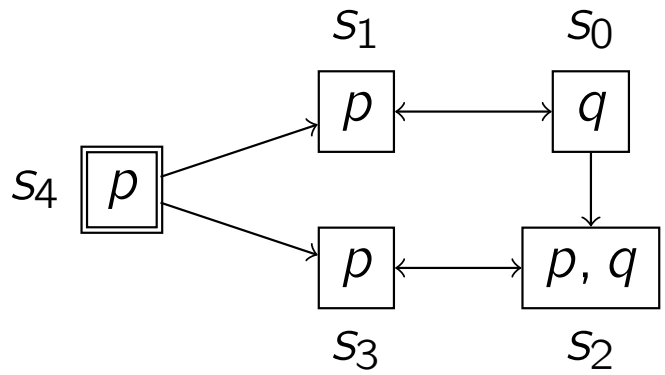
$Sat_M(p) = \{s_1, s_2, s_3, s_4\}$

$Sat_M(\mathbf{EX}p) = Pre(Sat(M, p)) = \{s_0, s_2, s_3, s_4\}$

$Sat_M(q) = \{s_0, s_2\}$

Символьный алгоритм: пример

M:



$$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$$

Преобразователь для $\mathbf{EG}p$:

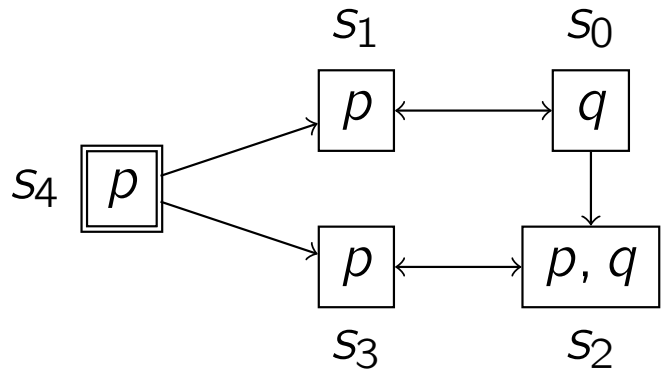
$$\mathbb{f}_1 = \lambda Z.Sat_M(p) \cap \mathbf{EX}_M(Z) = \lambda Z.\{s_1, s_2, s_3, s_4\} \cap Pre(M, Z)$$

Вычисление наибольшей неподвижной точки $\mathbb{f}_1$:

- $X_0 = S = \{s_0, s_1, s_2, s_3, s_4\}$
- $X_1 = \mathbb{f}_1(X_0) = \{s_1, s_2, s_3, s_4\} \cap Pre(M, S) = \{s_1, s_2, s_3, s_4\} \cap S = \{s_1, s_2, s_3, s_4\}$
- $X_2 = \mathbb{f}_1(X_1) = \{s_1, s_2, s_3, s_4\} \cap \{s_0, s_2, s_3, s_4\} = \{s_2, s_3, s_4\}$
- $X_3 = \mathbb{f}_1(X_2) = \{s_1, s_2, s_3, s_4\} \cap \{s_0, s_2, s_3, s_4\} = \{s_2, s_3, s_4\} = X_2$
- $Sat_M(\mathbf{EG}p) = \nu Z.\mathbb{f}_1(Z) = X_3 = \{s_2, s_3, s_4\}$

Символьный алгоритм: пример

M:



$$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$$

Преобразователь для $\mathbf{E}(q\mathbf{UEG}p)$:

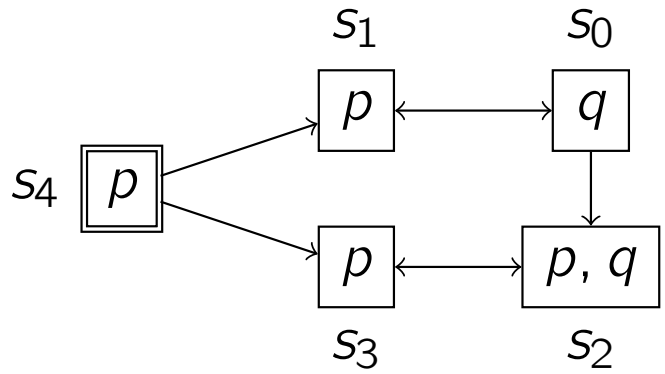
$\mathbb{f}_2 = \lambda Z.Sat_M(\mathbf{EG}p) \cup (Sat_M(q) \cap \mathbf{EX}_M(Z)) =$
$\lambda Z.\{s_2, s_3, s_4\} \cup (\{s_0, s_2\} \cap \mathbf{EX}_M(Z))$

Вычисление наименьшей неподвижной точки $\mathbb{f}_2$:

- $X_0 = \emptyset$
- $X_1 = \mathbb{f}_2(X_0) = \{s_2, s_3, s_4\} \cup (\{s_0, s_2\} \cap \emptyset) = \{s_2, s_3, s_4\}$
- $X_2 = \mathbb{f}_2(X_1) = \{s_2, s_3, s_4\} \cup (\{s_0, s_2\} \cap \{s_0, s_2, s_3, s_4\}) = \{s_0, s_2, s_3, s_4\}$
- $X_3 = \mathbb{f}_2(X_1) = \{s_2, s_3, s_4\} \cup (\{s_0, s_2\} \cap \{s_0, s_1, s_2, s_3, s_4\}) =$
  $\{s_0, s_2, s_3, s_4\} = X_2$
- $Sat_M(\mathbf{E}(q\mathbf{UEG}p)) = \mu Z.\mathbb{f}_2(Z) = \{s_0, s_2, s_3, s_4\}$

Символьный алгоритм: пример

M:



$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$

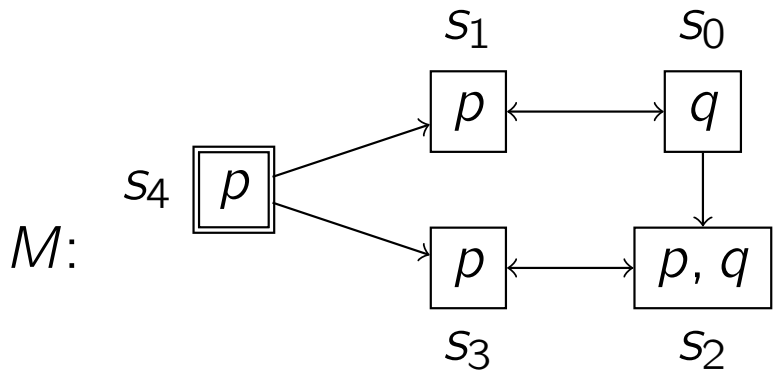
Конец — такой же, как в исходном базовом алгоритме

$Sat_M(\neg\mathbf{E}(q\mathbf{UEG}p)) = S \setminus Sat_M(\mathbf{E}(q\mathbf{UEG}p)) = \{s_1\}$

$Sat_M(\varphi) = Sat_M(\mathbf{EX}p) \cap Sat_M(\neg\mathbf{E}(q\mathbf{UEG}p)) = \emptyset$

Так как $\{s_4\} \not\subseteq \emptyset$, можно заключить, что $M \not\models \varphi$

Символьный алгоритм: пример



$$\varphi = \mathbf{EX}p \,\&\, \neg\mathbf{E}(q\mathbf{UEG}p)$$

Символьное представление модели M, основанное на формулах, для трёх разрядов, отвечающих переменным x_0, x_1, x_2, с естественным кодированием состояний согласно номерам выше:

- $\Phi_S = x_2 \to \neg x_1 \,\&\, \neg x_0$
- $\Phi_{S_0} = x_2 \,\&\, \neg x_1 \,\&\, \neg x_0$
- $\Phi_{\to} = \Phi_S \,\&\, \neg x_2' \,\&((x_0 \leftrightarrow x_0') \,\&(x_1 \oplus x_1') \vee \neg x_1 \,\&\, \neg x_0 \,\&\, x_1' \,\&\, \neg x_0')$
- $\Phi_p = \Phi_S \,\&(x_1 \vee x_0)$
- $\Phi_q = \neg x_2 \,\&\, \neg x_0$

А переписать результаты работы модифицированного базового алгоритма в символьном виде можете попробовать сами

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 29

CTL*
CTL и LTL как фрагменты CTL*
Сравнение выразительности CTL и LTL

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

БНФ для ctl-формул (Φ) и ltl-формул (φ) устроены так:

$$\Phi ::= \mathbb{t} \mid p \mid (\Phi \,\&\, \Phi) \mid (\neg\Phi) \mid (\mathbf{A}\chi) \mid (\mathbf{E}\chi),$$
$$\chi ::= (\mathbf{X}\Phi) \mid (\Phi\mathbf{U}\Phi)$$

$$\varphi ::= \mathbb{t} \mid p \mid (\varphi \,\&\, \varphi) \mid (\neg\varphi) \mid (\mathbf{X}\varphi) \mid (\varphi\mathbf{U}\varphi)$$

В языке CTL использование темпоральных операторов ограничивается так, чтобы каждый оператор (**U**, **X**) обязательно был предварён квантором пути (**A**, **E**)

В LTL можно произвольно комбинировать темпоральные операторы, но использование кванторов пути крайне ограничено:

- В синтаксисе этих кванторов нет
- В задаче model checking неявно предполагается квантор **A** в качестве внешней операции формулы:

  $M \models \varphi \Leftrightarrow$ любое вычисление M удовлетворяет формуле φ

А насколько сильно отличаются выразительные возможности этих двух языков?

CTL*

$$\Phi ::= \mathbb{t} \mid p \mid (\Phi \,\&\, \Phi) \mid (\neg\Phi) \mid (\mathbf{A}\chi) \mid (\mathbf{E}\chi),$$
$$\chi ::= (\mathbf{X}\Phi) \mid (\Phi \mathbf{U} \Phi)$$

$$\varphi ::= \mathbb{t} \mid p \mid (\varphi \,\&\, \varphi) \mid (\neg\varphi) \mid (\mathbf{X}\varphi) \mid (\varphi \mathbf{U} \varphi)$$

Язык CTL* — это расширение языка CTL, в котором в качестве формул пути (χ) разрешены произвольные ltl-формулы

Синтаксис ctl*-формул задаётся следующей БНФ:

$$\Phi ::= \mathbb{t} \mid p \mid (\Phi \,\&\, \Phi) \mid (\neg\Phi) \mid (\mathbf{A}\varphi) \mid (\mathbf{E}\varphi),$$
$$\varphi ::= \Phi \mid \varphi \,\&\, \varphi \mid \neg\varphi \mid (\mathbf{X}\varphi) \mid (\varphi \mathbf{U} \varphi),$$

где Φ — формула состояния (она же ctl*-формула) и φ — формула пути

Для ctl*-формул и их подформул, как и для ctl-формул, определяются два вида выполнимости:

- Выполнимость ctl*-формулы Φ в состоянии s модели Крипке M ($M, s \models \Phi$)
  - Эта часть семантики дословно переносится из языка CTL
- Выполнимость формулы пути φ на бесконечном пути π модели Крипке M ($M, \pi \models \varphi$)

CTL*

Выполнимость формулы пути на бесконечном пути π модели Крипке M в CTL* задаётся *почти так же, как и для LTL*:

- $M, \pi \models \Phi$ для ctl*-формулы Φ $\Leftrightarrow$ $M, \pi[1] \models \Phi$
- $M, \pi \models \varphi_1 \,\&\, \varphi_1$ $\Leftrightarrow$ $M, \pi \models \varphi_1$ и $M, \pi \models \varphi_2$
- $M, \pi \models \neg\varphi$ $\Leftrightarrow$ $M, \pi \not\models \varphi$
- $M, \pi \models \mathbf{X}\varphi$ $\Leftrightarrow$ $M, \pi^{\geq 2} \models \varphi$
- $M, \pi \models \varphi_1 \mathbf{U} \varphi_2$ $\Leftrightarrow$ существует момент времени k, такой что
  - $M, \pi^{\geq k} \models \varphi_2$ и
  - для любого момента времени i, $i < k$, верно $M, \pi^{\geq i} \models \varphi_1$

CTL*-формула φ выполняется на модели M ($M \models \varphi$), если она выполняется в каждом начальном состоянии этой модели

Задача model checking для CTL* (MC-CTL*) формулируется так:

Для заданной модели Крипке M и заданной ctl*-формулы φ проверить справедливость соотношения $M \models \varphi$

CTL и LTL как фрагменты CTL*

Из устройства семантических правил в языке CTL и для ctl-формул как фрагмента языка CTL* немедленно вытекают следующие утверждения

Утверждение. **Для любых модели Крипке M, её состояния s и ctl-формулы φ верно:**

$$M, s \models \varphi \text{ **в языке CTL**} \quad \Leftrightarrow \quad M, s \models \varphi \text{ **в языке CTL***}$$

Утверждение. **Для любой модели Крипке M и любой ctl-формулы φ верно:**

$$M \models \varphi \text{ **в языке CTL**} \quad \Leftrightarrow \quad M \models \varphi \text{ **в языке CTL***}$$

Таким образом, любая ctl-формула может расцениваться как ctl*-формула частного вида

CTL и LTL как фрагменты CTL*

Из устройства семантических правил в языках LTL и CTL* немедленно вытекают следующие утверждения

Утверждение. **Для любых модели Крипке M и её пути π, трассы τ этого пути и любой ltl-формулы φ верно:**

$$\tau \models \varphi \text{ **в языке LTL**} \quad \Leftrightarrow \quad M, \pi \models \varphi \text{ **в языке CTL***}$$

Утверждение. **Для любых модели крипке M и ltl-формулы φ верно:**

$$M \models \varphi \text{ **в языке LTL**} \quad \Leftrightarrow \quad M \models \mathbf{A}\varphi \text{ **в языке CTL***}$$

Таким образом, любая ltl-формула φ может расцениваться как формула $\mathbf{A}\varphi$ языка CTL* с тем же смыслом

Сравнение выразительности CTL и LTL

Теперь CTL и LTL можно полноценно рассматривать как фрагменты «объемлющего» языка CTL*, а значит, можно сравнивать широту возможностей выражения тех или иных свойств моделей на этих языках

Ctl*-формулы φ и ψ будем называть эквивалентными ($\varphi \sim \psi$), если для любой модели Крипке M и любого состояния s справедлива равносильность

$$M, s \models \varphi \quad \Leftrightarrow \quad M, s \models \psi$$

Для фрагментов $\mathfrak{L}_1$, $\mathfrak{L}_2$ языка CTL* будем говорить, что

- $\mathfrak{L}_1$ не менее выразителен, чем $\mathfrak{L}_2$ ($\mathfrak{L}_2 \preceq \mathfrak{L}_1$), если для любой формулы из $\mathfrak{L}_2$ существует эквивалентная формула из $\mathfrak{L}_1$
- $\mathfrak{L}_1$ и $\mathfrak{L}_2$ эквивалентны ($\mathfrak{L}_1 \sim \mathfrak{L}_2$), если $\mathfrak{L}_1 \preceq \mathfrak{L}_2$ и $\mathfrak{L}_2 \preceq \mathfrak{L}_1$
- $\mathfrak{L}_1$ строго менее выразителен, чем $\mathfrak{L}_2$ ($\mathfrak{L}_1 \prec \mathfrak{L}_2$), если $\mathfrak{L}_1 \preceq \mathfrak{L}_2$ и $\mathfrak{L}_1 \not\sim \mathfrak{L}_2$
- $\mathfrak{L}_1$ и $\mathfrak{L}_2$ несравнимы ($\mathfrak{L}_1 \perp \mathfrak{L}_2$), если $\mathfrak{L}_1 \not\preceq \mathfrak{L}_2$ и $\mathfrak{L}_1 \not\preceq \mathfrak{L}_2$

Сравнение выразительности CTL и LTL

Утверждение. **Не существует ctl-формулы, эквивалентной формуле AFGp, где p — атомарное высказывание**

Утверждение. **Не существует ltl-формулы φ, такой что формула Aφ эквивалентна формуле AGEFp, где p — атомарное высказывание**

Утверждение. **Не существует ни ctl-формулы, ни формулы вида Aφ, где φ — ltl-формула, эквивалентной формуле AFG$p \vee$ AGEFp, где p — атомарное высказывание**

Доказать эти утверждения можете попробовать самостоятельно (и каждое из этих утверждений трудно)

Следствие. **Справделивы следующие соотношения:**

- **CTL $\perp$ LTL**
- **CTL $\cup$ LTL $\prec$ CTL***

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 30

Системы реального времени (СРВ)
Временны́е автоматы

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Системы реального времени

Система реального времени (СРВ) — это система, поведение которой существенно зависит не только от порядка выполнения действий и изменения состояний, но и от того, **за какое время** выполняются действия и изменяются состояния

Для СРВ характерны директивные сроки выполнения действий компонентами: интервалы (*действительных чисел*), которым должна принадлежать длительность выполнения действий

Но в СРВ, даже разработанной в том или ином смысле «правильно», не всегда соблюдаются директивные сроки, хотя и «должны»

Для сравнения: если «правильным» считать студента, который в итоге успешно выпускается, то по уставу университета такой студент должен посещать лекции, но это не значит, что он действительно их посещает

Системы реального времени

В зависимости от того, к каким последствиям приводит несоблюдение директивных сроков, СРВ принято причислять к одному из двух классов:

- В мягких СРВ последствия хотя и нежелательны (ухудшают качество выполнения системы), но в целом допустимы
- В жёстких СРВ сорванный директивный срок считается недопустимым, приводящим к фатальному сбою с бессмысленностью продолжения выполнения

Системы реального времени

Примеры сорванных сроков в мягких СРВ:

- Поспал на два часа меньше нормы $\Rightarrow$ будешь вялым, но если не злоупотреблять, то жить будешь
- Почта задержалась на год $\Rightarrow$ печально, но все к этому привыкли
- Процесс долго освобождал память $\Rightarrow$ операционная система «подвиснет», но потом восстановится

Примеры сорванных сроков в жёстких СРВ:

- Парашют раскрылся на минуту позже документации $\Rightarrow$ смерть
- Схемные сигналы не стабилизировались в процессоре за такт $\Rightarrow$ процессор отправляется на свалку
- Светофор стал зелёным раньше положенного $\Rightarrow$ авария

Далее (*так или иначе, явно или неявно*) будут рассматриваться **только жёсткие** СРВ

Системы реального времени

Пример

Рассмотрим СРВ $\mathfrak{S}$, предназначенную для распознавания одинарного и двойного нажатия кнопки мыши:

- В $\mathfrak{S}$ выполняются (происходят в окружении и порождаются системой) действия
  - *click*: нажата кнопка мыши
  - *single*: произошло одинарное нажатие
  - *double*: произошло двойное нажатие
- Если в режиме ожидания нажатия выполнилось *click*, то через *единицу времени* $\mathfrak{S}$ принимает решение о том, какое нажатие произошло, одинарное или двойное, и порождает соответствующее действие
  - Если после первого *click* до вынесения решения ещё раз выполнилось *click*, то нажатие двойное
  - Иначе — одинарное

Системы реального времени

Пример

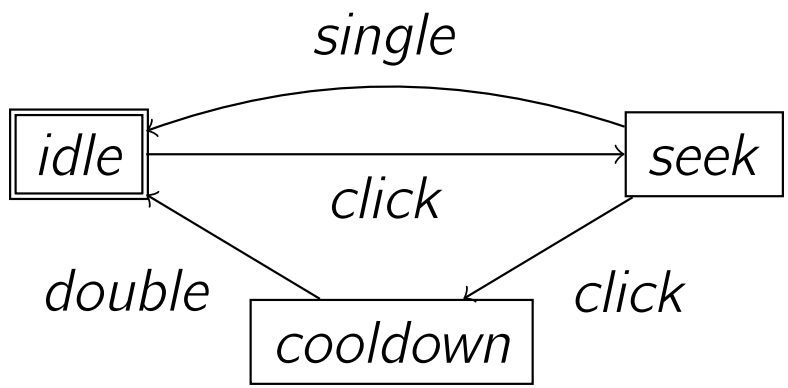
Начнём формализацию $\mathfrak{S}$ с системы переходов, отвечающей всем возможностям выполнения действий без учёта директивных сроков

Начальное состояние (*idle*) отвечает режиму ожидания первого *click*

По первому *click* перейдём в состояние *seek* ожидания второго *click*

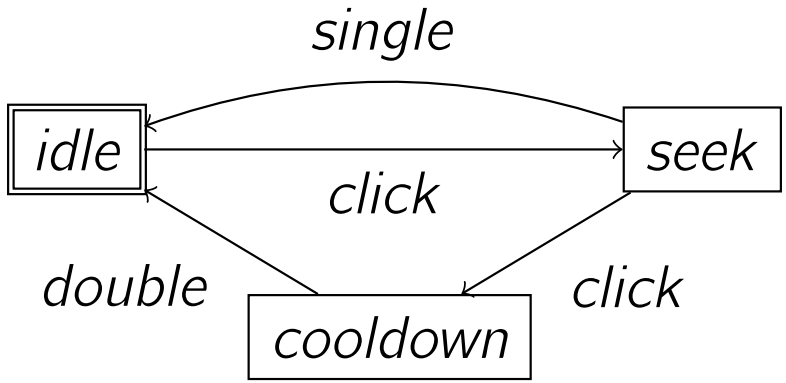
Если второе *click* не обнаружено в заданный срок, то выносится решение об одинарном нажатии: перейдём в *idle*, породив действие *single*

Если второе *click* обнаружено, то перейдём в режим *cooldown*, и через некоторое время вынесем решение о двойном нажатии: перейдём в *idle*, породив действие *double*



Системы реального времени

Пример



Чтобы следить за директивными сроками, добавим в модель часы-секундомеры:

- Значение часов — это действительное число, показывающее, сколько времени прошло с их последнего сброса
- Сбрасывать часы будем по желанию при выполнении переходов

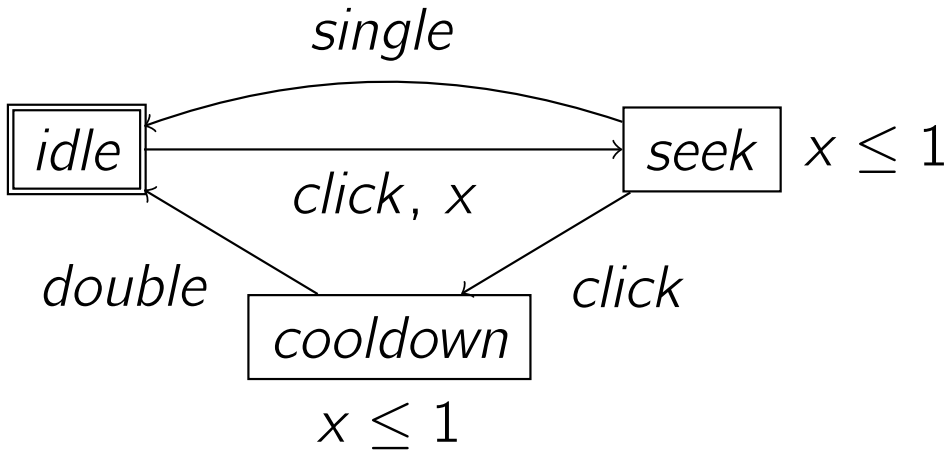
В этом примере достаточно одних часов x

При выполнении перехода $idle \to seek$ сбросим часы x, чтобы следить за тем, не пора ли выносить решение о нажатии

Чтобы обозначить сброс часов на переходе, пометим переход этими часами

Системы реального времени

Пример



В некоторых состояниях $\mathfrak{S}$ не может находиться сколь угодно долго

В состояниях *seek* и *cooldown* значение 1 часов x означает, что пришла пора выносить решение о нажатии и переходить в *idle*

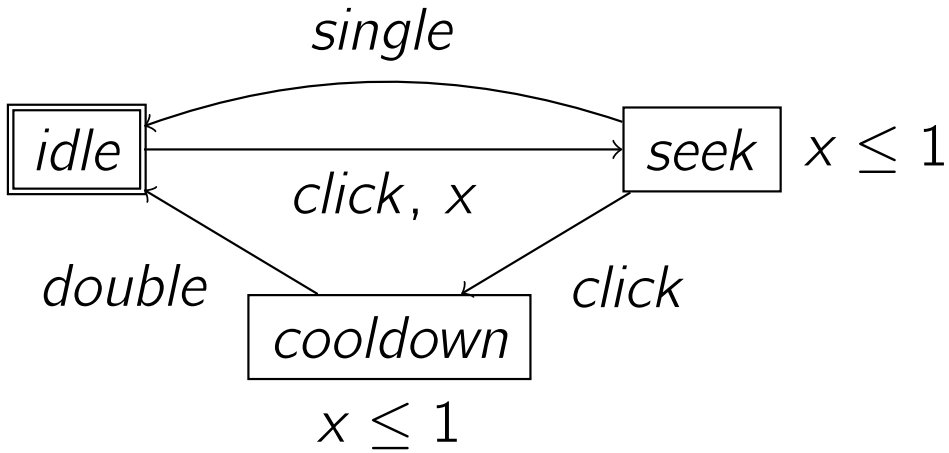
Значит, в этих состояниях значение x не может быть больше 1, то есть верно $x \leq 1$

Пометим *seek* и *cooldown* этим неравенством

Ограничение на длительность нахождение в состоянии называется инвариантом

Системы реального времени

Пример



Для каждого значения x каждый переход либо открыт (может быть выполнен), либо закрыт (не может быть выполнен)

Переходы $seek \to idle$ и $cooldown \to idle$ открыты $\Leftrightarrow x = 1$

Переход $seek \to cooldown$ открыт $\Leftrightarrow x < 1$

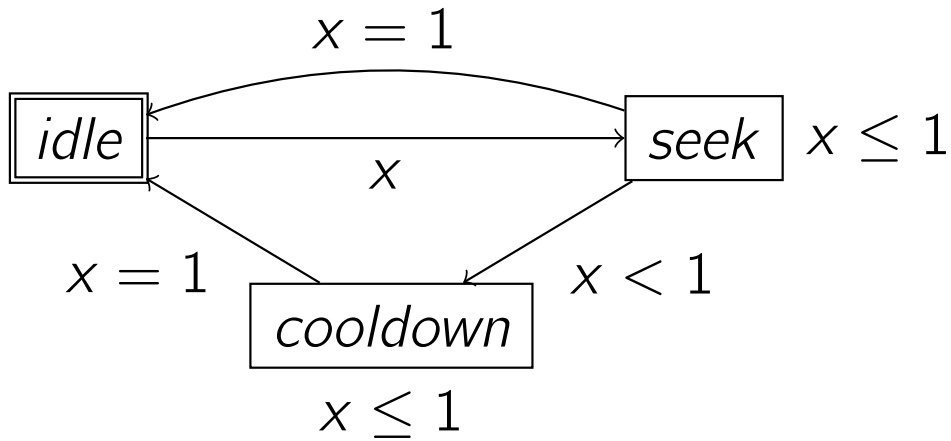
Остальные переходы открыты всегда

Пометим переходы выражениями, истинность которых равносильна открытости этих переходов

Такие выражения называются предусловиями переходов (англ. guard; иногда переводится как «охрана», «охранник» и «страж»)

Системы реального времени

Пример



Теперь «забудем» (*по крайней мере на время*) о действиях, ограничившись только тем, какими свойствами обладают состояния системы

(*По аналогии с тем, как «забываются» действия системы переходов, чтобы из неё получилась модель Крипке*)

В результате получилась модель, похожая на модель Крипке, но содержащая часы (реального времени) и предназначенная для моделирования СРВ: временной автомат

Временны́е автоматы: временны́е ограничения

Синтаксис временны́х ограничений над множеством часов $\mathcal{C}$ задаётся следующей БНФ:

$$\begin{array}{lcl} g & ::= & ag \mid (g \,\&\, g) \mid (\neg g), \\ ag & ::= & \mathbb{t} \mid (x < k) \mid (x \leq k), \end{array}$$

где g — временное ограничение, ag — атомарное временное ограничение, $x \in \mathcal{C}$ и $k \in \mathbb{N}_0$

$\mathbb{R}$, $\mathbb{R}_{\geq 0}$ и $\mathbb{R}_{>0}$ — так будем обозначать множества действительных, неотрицательных действительных и положительных действительных чисел соответственно

Временны́е автоматы: временны́е ограничения

Оценка часов множества $\mathcal{C}$ — это отображение вида $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$

Выполнимость временно́го ограничения g на оценке часов ν ($\nu \models g$) определяется естественным образом:

- Всегда верно $\nu \models \mathbb{t}$
- $\nu \models (x < k) \quad \Leftrightarrow \quad \nu(x) < k$
- $\nu \models (x \leq k) \quad \Leftrightarrow \quad \nu(x) \leq k$
- $\nu \models (g_1 \,\&\, g_2) \quad \Leftrightarrow \quad \nu \models g_1$ и $\nu \models g_2$
- $\nu \models (\neg g) \quad \Leftrightarrow \quad \nu \not\models g$

Временно́е ограничение будем называть инвариантным, если в нём не содержится $\neg$

$CC(\mathcal{C})$, $AC(\mathcal{C})$ и $IC(\mathcal{C})$ — так будем обозначать множества временны́х ограничений над $\mathcal{C}$ — всевозможных, атомарных и инвариантных соответственно

Временны́е автоматы: временны́е ограничения

В синтаксисе временны́х ограничений будут использоваться и другие булевы операции и арифметические отношения, расценивающиеся как естественные сокращения:

- $\mathbb{f} \quad = \quad \neg\mathbb{t}$
- $g_1 \vee g_2 = \neg((\neg g_1)\,\&(\neg g_2))$
- $g_1 \to g_2 = (\neg g_1) \vee g_2$
- $x \geq k = \neg(x < k)$
- $x > k = \neg(x \leq k)$
- $x = k = (x \leq k)\,\&(x \geq k)$
- $x \neq k = \neg(x = k)$

Во всех этих сокращениях используется $\neg$, а значит, их **нельзя** использовать в инвариантных ограничениях

В записи временны́х ограничений будем опускать скобки согласно обычным приоритетам булевых операций

Временны́е автоматы: синтаксис

Временной автомат над конечными множествами атомарных высказываний AP и часов $\mathcal{C}$ — это система $\mathcal{A} = (S, s_0, \mathcal{I}, T, L)$, где:

- S — конечное множество состояний
- $s_0 \in S$ — начальное состояние
- $\mathcal{I} : S \to IC(\mathcal{C})$ — разметка состояний инвариантами
- $T \subseteq S \times CC(\mathcal{C}) \times 2^{\mathcal{C}} \times S$ — отношение переходов
- $L : S \to 2^{\mathrm{AP}}$ — разметка состояний событиями

Переход вида (s_1, g, X, s_2) называется переходом из состояния s_1 в состояние s_2 с предусловием g и сбросом всех часов из X и будет изображаться так: $s_1 \xrightarrow{g,X} s_2$

Временны́е автоматы: синтаксис

Автомат $\mathcal{A}$ представляет собой размеченный конечный ориентированный граф:

- Вершины — это состояния автомата
- Ду́ги, помеченные предусловиями и множествами часов — это переходы автомата
- Остальные компоненты автомата — это метки вершин

В связи с этим к временны́м автоматам будет применяться графовая терминология

Временно́е ограничение $\mathbb{t}$ и множество часов $\emptyset$ в изображениях иногда будут опускаться, и множество часов $\{x_1, \ldots, x_n\}$ иногда будет записываться без фигурных скобок: $x_1, \ldots, x_n$

Временны́е автоматы: синтаксис

Замечание для любопытных

При разработке временного автомата может возникнуть желание записать ограничение «$x < k$» и «$x \leq k$», где $k \in \mathbb{R}_{\geq 0}$ или $k \in \mathbb{Q}_{\geq 0}$ (это множество всех неотрицательных рациональных чисел)
Принято полагать, что более строгое ограничение «$k \in \mathbb{N}_0$» на самом деле не ограничивает выразительные возможности:

- Любое число из $\mathbb{R}_{\geq 0}$ может быть приближено числом из $\mathbb{Q}_{\geq 0}$ с любой наперёд заданной точностью
- Замеры времени выполнения СРВ возможны только с некоторой погрешностью (точностью)
  - $\Rightarrow$ множество $\mathbb{R}_{\geq 0}$ излишне, достаточно использовать $\mathbb{Q}_{\geq 0}$
- Любой конечный набор рациональных чисел можно привести к общему знаменателю
- Домножением всех рациональных чисел в записи автомата на их общий знаменатель N («замедлением» модельного времени в N раз) можно преобразовать все эти числа в целые неотрицательные

Временны́е автоматы: семантика

Вычислительная конфигурация автомата $\mathcal{A}$ — это пара (s, ν), где s — состояние автомата и ν — оценка часов

Для технической простоты иногда будем полагать, что часы автомата упорядочены ($\mathcal{C} = \{x_1, \ldots, x_n\}$), и записывать оценку ν как набор значений часов: $(\nu(x_1), \ldots, \nu(x_n))$

Начальная конфигурация автомата $\mathcal{A}$ с начальным состоянием s_0 имеет вид $(s_0, (0, 0, \ldots, 0))$

Автомат $\mathcal{A}$ выполняется пошагово согласно двуместному отношению шага вычисления $\to_{\mathcal{A}}$

Это отношение зададим как объединение двух отношений, отвечающих двум возможностям шага вычисления автомата (строгое определение будет дальше):

- $\mapsto_{\mathcal{A}}$ — продвижение времени: время течёт, автомат бездействует
- $\hookrightarrow_{\mathcal{A}}$ — выполнение перехода: время не течёт, переход выполняется (мгновенно)

Иногда будем опускать индекс $\mathcal{A}$ в отношениях $\to_{\mathcal{A}}$, $\hookrightarrow_{\mathcal{A}}$ и $\mapsto_{\mathcal{A}}$, если автомат $\mathcal{A}$ однозначно задаётся контекстом или неважен

Временны́е автоматы: семантика

Продвижение времени

Для оценки часов ν, конфигураций $\sigma = (s, \nu)$ и σ' и числá $d \in \mathbb{R}_{\geq 0}$ будем использовать такие обозначения:

- $\nu + d$ — это оценка часов, такая что $(\nu + d)(x) = \nu(x) + d$ для любых часов x
- $\sigma + d = (s, \nu + d)$
- $\sigma \overset{d}{\mapsto} \sigma'$ означает, что $\sigma' = \sigma + d$

$\sigma \mapsto_{\mathcal{A}} \sigma'$ для автомата $\mathcal{A} = (S, s_0, \mathcal{I}, T, L)$ и конфигураций $\sigma = (s, \nu)$ и σ', если существует константа $d \in \mathbb{R}_{>0}$, для которой верно:

1. $\sigma \overset{d}{\mapsto} \sigma'$
2. $\nu + d \models \mathcal{I}(s)$

Временны́е автоматы: семантика

Выполнение перехода

Для оценки часов ν, множества часов X, конфигураций $\sigma = (s, \nu)$ и σ', состояния s' и перехода $t = (s \xrightarrow{g,X} s')$ будем использовать такие обозначения:

- $\nu[X]$ — оценка часов, такая что
$$\nu[X](x) = 0, \quad \text{если } x \in X, \text{ и}$$
$$\nu[X](x) = \nu(x) \quad \text{иначе}$$
- $\sigma[X] = (s, \nu[X])$
- $\sigma[s'] = (s', \nu)$
- $\sigma \overset{t}{\hookrightarrow} \sigma'$ означает, что $\sigma' = \sigma[X][s']$

$\sigma \hookrightarrow_{\mathcal{A}} \sigma'$ для автомата $A = (S, s_0, \mathcal{I}, T, L)$ и конфигураций $\sigma = (s, \nu)$ и σ', если существует переход $t = (s \xrightarrow{g,X} s') \in T$, для которого верно:

1. $\nu \models g$
2. $\sigma \overset{t}{\hookrightarrow} \sigma'$
3. $\nu[X] \models \mathcal{I}(s')$

Временны́е автоматы: семантика

Трассой временно́го автомата $\mathcal{A}$ из конфигурации σ (или, коротко, — σ-трассой) назовём последовательность конфигураций вида

$$\sigma_1 \to_{\mathcal{A}} \sigma_2 \to_{\mathcal{A}} \sigma_3 \to_{\mathcal{A}} \dots,$$

где $\sigma_1 = \sigma$

σ-трассу автомата $\mathcal{A}$ назовём начальной, если σ — начальная конфигурация $\mathcal{A}$

Конфигурацию σ автомата $\mathcal{A}$ назовём тупиковой, если не существует конфигурации σ', такой что $\sigma \to_{\mathcal{A}} \sigma'$

Трассу назовём

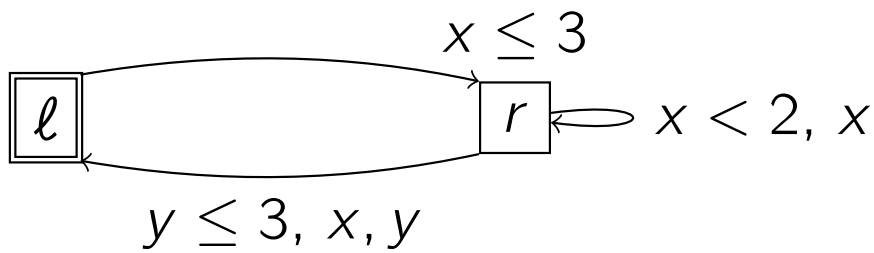
- тупиковой, если она конечна и оканчивается тупиковой конфигурацией, и
- полной, если она бесконечная или тупиковая

Вычислением автомата $\mathcal{A}$ назовём полную начальную трассу

Временны́е автоматы: семантика

Пример

Рассмотрим такой временной автомат $\mathcal{A}$ над множеством часов $\{x, y\}$ (атомарные высказывания опущены за ненадобностью):



Пример тупикового вычисления $\mathcal{A}$ *для порядка часов* (x, y):

$$(\ell, 0, 0) \hookrightarrow (r, 0, 0) \mapsto (r, 1, 1) \mapsto (r, \sqrt{2}, \sqrt{2}) \hookrightarrow (r, 0, \sqrt{2}) \mapsto (r, 3, \sqrt{2} + 3)$$

Пример бесконечного вычисления:

$$(\ell, 0, 0) \mapsto (\ell, 1, 1) \mapsto (\ell, 2, 2) \mapsto \ldots \mapsto (\ell, n, n) \mapsto \ldots$$

Другой пример бесконечного вычисления:

$$(\ell, 0, 0) \mapsto (\ell, 1.2, 1.2) \hookrightarrow (r, 1.2, 1.2) \hookrightarrow (\ell, 0, 0) \mapsto (\ell, 1.2, 1.2) \hookrightarrow \ldots$$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 31

Неправдоподобные вычисления временны́х автоматов

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Длительностью шага вычисления $\sigma \to \sigma'$ назовём число $\Delta(\sigma, \sigma')$, равное

- d, если $\sigma \overset{d}{\mapsto} \sigma'$, где $d > 0$
- 0, если $\sigma \hookrightarrow \sigma'$

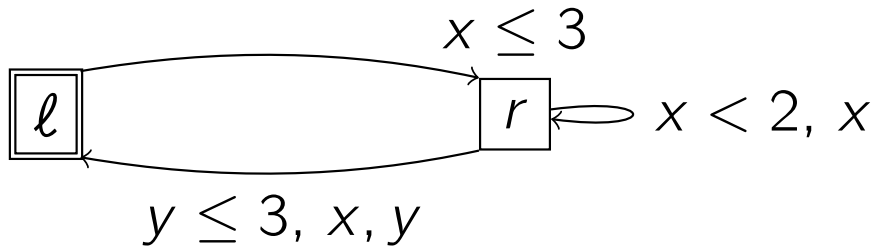
Длительностью трассы $\sigma_0 \to \sigma_1 \to \ldots$ назовём

- сумму $\sum_{i=0}^{k} \Delta(\sigma_i, \sigma_{i+1})$, если эта трасса конечна и имеет длину $(k+1)$
- сумму ряда $\sum_{i=0}^{\infty} \Delta(\sigma_i, \sigma_{i+1})$, если трасса бесконечна

Трассу назовём конвергентной, если её длительность конечна, и дивергентной иначе

Вычислением Зенона, или, по-другому, зеноновским вычислением, назовём конвергентное вычисление, содержащее бесконечно много шагов выполнения перехода

Пример



Конвергентные незеноновские вычисления:

Все тупиковые вычисления конечны, а значит, попадают в эту категорию — например:

$$(\ell, 0, 0) \hookrightarrow (r, 0, 0) \mapsto (r, 1, 1) \mapsto (r, \sqrt{2}, \sqrt{2}) \hookrightarrow (r, 0, \sqrt{2}) \mapsto (r, 3, \sqrt{2} + 3)$$

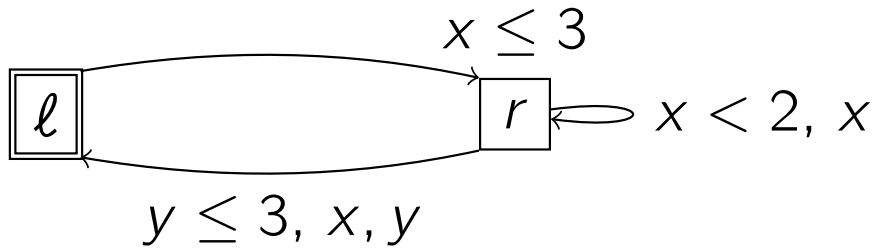
Длительность этого вычисления — $\sqrt{2} + 3$, выполнение перехода встретилось 2 раза

Бывают и бесконечные вычисления такого вида — например:

$$(\ell, 0, 0) \mapsto (\ell, \tfrac{1}{2}, \tfrac{1}{2}) \mapsto (\ell, \tfrac{2}{3}, \tfrac{2}{3}) \mapsto \ldots \mapsto (\ell, \tfrac{n-1}{n}, \tfrac{n-1}{n}) \mapsto \ldots$$

В этом вычислении ни разу не выполняется переход, и длительность вычисления — 1

Пример



Дивергентные вычисления:

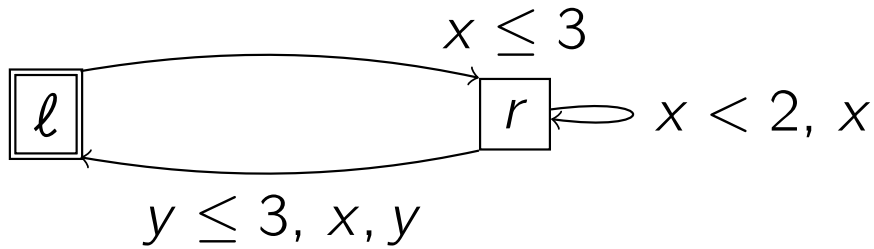
$$(\ell, 0, 0) \mapsto (\ell, 1.2, 1.2) \hookrightarrow (r, 1.2, 1.2) \hookrightarrow (\ell, 0, 0) \mapsto (\ell, 1.2, 1.2) \hookrightarrow \ldots$$

$$(\ell, 0, 0) \mapsto (\ell, 1, 1) \mapsto (\ell, 2, 2) \mapsto \ldots \mapsto (\ell, n, n) \mapsto \ldots$$

$$(\ell, 0, 0) \mapsto (\ell, \tfrac{1}{2}, \tfrac{1}{2}) \hookrightarrow (r, \tfrac{1}{2}, \tfrac{1}{2}) \hookrightarrow (\ell, 0, 0) \mapsto (\ell, \tfrac{1}{3}, \tfrac{1}{3}) \hookrightarrow (r, \tfrac{1}{3}, \tfrac{1}{3}) \hookrightarrow \ldots$$
$$\hookrightarrow (\ell, 0, 0) \mapsto (\ell, \tfrac{1}{n}, \tfrac{1}{n}) \hookrightarrow (r, \tfrac{1}{n}, \tfrac{1}{n}) \hookrightarrow \ldots$$

Длительность последнего вычисления равна $\sum_{i=1}^{\infty} \frac{1}{i}$, и известно что этот ряд расходится

Пример



Зеноновские вычисления:

$$(\ell, 0, 0) \hookrightarrow (r, 0, 0) \hookrightarrow (\ell, 0, 0) \hookrightarrow (r, 0, 0) \hookrightarrow \dots$$

Длительность этого вычисления — 0, и бесконечное число раз выполняются переходы

$$(\ell, 0, 0) \mapsto (\ell, \tfrac{1}{2}, \tfrac{1}{2}) \hookrightarrow (r, \tfrac{1}{2}, \tfrac{1}{2}) \hookrightarrow (\ell, 0, 0) \mapsto (\ell, \tfrac{1}{4}, \tfrac{1}{4}) \hookrightarrow (r, \tfrac{1}{4}, \tfrac{1}{4}) \hookrightarrow \dots$$
$$\hookrightarrow (\ell, 0, 0) \mapsto (\ell, \tfrac{1}{2^n}, \tfrac{1}{2^n}) \hookrightarrow (r, \tfrac{1}{2^n}, \tfrac{1}{2^n}) \hookrightarrow \dots$$

Длительность этого вычисления — 1, и бесконечно часто (хотя и не всегда) выполняются переходы

«В реальности» длительность выполнения СРВ потенциально бесконечна: сколько бы времени ни происходило наблюдение, всегда можно подождать ещё минуту

Поэтому все конвергентные вычисления следует считать нереалистичными:

- Тупиковое вычисление означает, что время принципиально не может больше течь, чего не бывает в реальности
- Бесконечное конвергентное вычисление означает, что за конечное время с системы было снято бесконечное число «снимков», а актуальной бесконечности не существует

При этом в любом автомате, содержащем хотя бы одно дивергентное вычисление, содержится и бесконечно много конвергентных: достаточно заменить шаг $\sigma \overset{d}{\longmapsto} \sigma'$ на трассу $\sigma \overset{\frac{d}{2}}{\longmapsto} \sigma_1 \overset{\frac{d}{4}}{\longmapsto} \ldots \overset{\frac{d}{2^n}}{\longmapsto} \sigma_n \overset{\frac{d}{2^{n+1}}}{\longmapsto} \ldots$ или любую аналогичную с суммой d ряда длительностей

Конвергентные вычисления имеют разную природу:

- некоторые из них (как упомянутые на предыдущем слайде) неизбежно следуют из устройства модели временны́х автоматов,
- но бывают и такие, которые свидетельствуют о некорректности конкретного автомата

«Неизбежные» конвергентные вычисления принято исключать из рассмотрения в семантике языка спецификаций СРВ

А остальные придётся исключить, поделив автоматы на «хорошо» и «плохо» построенные при помощи следующего определения

Временной автомат $\mathcal{A}$ будем называть корректным (построенным «хорошо»), если верно следующее:

- Не существует ни одного зеноновского вычисления $\mathcal{A}$
- Любая начальная трасса $\mathcal{A}$ может быть продолжена до дивергентного вычисления $\mathcal{A}$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 32

Логика ветвящегося реального времени (TCTL)

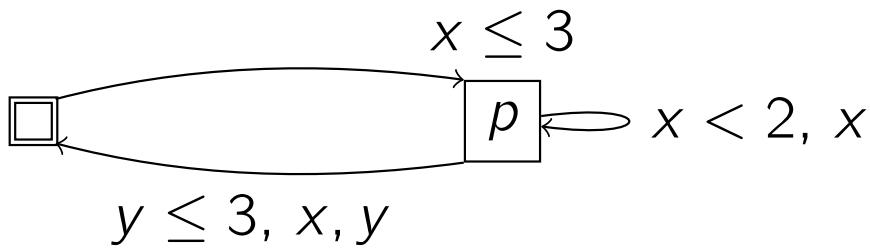
Задача model checking для TCTL

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Напоминание

Временной автомат над множеством часов $\{x, y\}$ и множеством атомарных высказываний $\{p\}$:



Дивергентное вычисление (*для порядка часов* (x, y)):

$$(\boxdot, 0, 0) \mapsto (\boxdot, 1, 1) \mapsto (\boxdot, 2, 2) \mapsto \ldots \mapsto (\boxdot, n, n) \mapsto \ldots$$

Вычисление Зенона (конвергентное и с бесконечным числом $\hookrightarrow$):

$$(\boxdot, 0, 0) \mapsto (\boxdot, 1.2, 1.2) \hookrightarrow (\square, 1.2, 1.2) \hookrightarrow (\boxdot, 0, 0) \mapsto (\boxdot, 1.2, 1.2) \hookrightarrow \ldots$$

Тупиковое конвергентное вычисление:

$$(\boxdot, 0, 0) \hookrightarrow (\square, 0, 0) \mapsto (\square, 1, 1) \mapsto (\square, \sqrt{2}, \sqrt{2}) \hookrightarrow (\square, 0, \sqrt{2}) \mapsto (\square, 3, \sqrt{2} + 3)$$

Автомат корректен, если не имеет зеноновских вычислений и любая его начальная трасса может быть продолжена до дивергентной

TCTL: синтаксис

Логика ветвящегося реального времени (Timed CTL; TCTL) — это аналог CTL, предназначенный для записи требований, предъявляемых к СРВ

Будем использовать такой краткий синтаксис tctl-формул над конечными множествами часов $\mathcal{C}$ и атомарных высказываний AP:

$$\Phi \quad ::= \quad \mathbb{t} \mid p \mid ag \mid (\Phi \,\&\, \Phi) \mid (\neg\Phi) \mid (\mathbf{A}\varphi) \mid (\mathbf{E}\varphi),$$
$$\varphi \quad ::= \quad (\Phi \mathbf{U} \Phi),$$

где Φ — формула состояния (она же tctl-формула), φ — формула пути, $p \in \mathrm{AP}$ и $ag \in AC(\mathcal{C})$

Отличия от синтаксиса ctl-формул:

1. Наряду с атомарными высказываниями (p) можно записывать и атомарные временны́е ограничения (ag)
2. В синтаксисе нет оператора **X**, так как для реального времени понятие «следующий момент времени» бессмысленно

TCTL: семантика

Содержательная трактовка кванторов пути (**A**, **E**) и темпорального оператора (**U**) — это их трактовка в CTL, адаптированная к особенностям поведения СРВ:

- **E**φ: существует **дивергентная** трасса, для которой верно Φ
- **A**φ: для любой **дивергентной** трассы верно Φ
- Φ**U**Ψ: в **реальном** будущем станет верным Ψ, а до тех пор будет верно Φ

Чтобы адекватно учесть реальное время в операторе **U**, следует «реально» переосмыслить дискретные (пошаговые) трассы автоматов

TCTL: семантика

Например, трассу

$$(\ell, 0) \mapsto (\ell, 1) \hookrightarrow (r, 0)$$

следует понимать так:

- Автомат начинает выполнение в состоянии ℓ со значением часов 0
- Затем автомат ожидает единицу времени, и во время ожидания значение часов непрерывно возрастает от 0 до 1, проходя через все значения интервала $[0, 1]$
- После этого автомат мгновенно переходит в состояние r, и одовременно с этим значение часов сбрасывается

Таким образом, между конфигурациями $(\ell, 0)$ и $(\ell, 1)$ в этой трассе *неявно подразумевается* континуум промежуточных конфигураций для всех значений часов интервала $(0, 1)$

TCTL: семантика

Будем говорить, что конфигурация σ

- покрывается парой конфигураций σ_1, σ_2, если верно хотя бы одно из двух:
  - $\sigma = \sigma_2$
  - $\sigma_1 \overset{d}{\mapsto} \sigma_2$ и $\sigma_1 \overset{d'}{\longmapsto} \sigma$, где $d' \in (0, d)$
- покрывается k-м шагом трассы
$$\sigma_1 \to \sigma_2 \to \ldots,$$
где $k \geq 1$, если она порождается парой σ_k, σ_{k+1}
- покрывается 0-м шагом трассы
$$\sigma_1 \to \sigma_2 \to \ldots,$$
если $\sigma = \sigma_1$
- покрывается трассой, если она покрывается каком-либо шагом этой трассы

TCTL: семантика

Как и для CTL, для TCTL задаются два вида выполнимости:

- Выполнимость tctl-формулы Φ в конфигурации σ автомата $\mathcal{A}$: $\mathcal{A}, \sigma \models \Phi$
- Выполнимость формулы пути φ на дивергентной трассе τ автомата $\mathcal{A}$: $\mathcal{A}, \tau \models \varphi$

Эти отношения для автомата $\mathcal{A} = (S, s_0, \mathcal{I}, T, L)$ и формул над атомарными высказываниями AP и часами $\mathcal{C}$ задаются так:

- Всегда верно $\mathcal{A}, \sigma \models \mathtt{t}$
- $\mathcal{A}, (s, \nu) \models p$, где $p \in \mathrm{AP} \quad \Leftrightarrow \quad p \in L(s)$
- $\mathcal{A}, (s, \nu) \models ag$, где $ag \in AC(\mathcal{C}) \quad \Leftrightarrow \quad \nu \models ag$
- $\mathcal{A}, \sigma \models \Phi_1 \,\&\, \Phi_2 \quad \Leftrightarrow \quad \mathcal{A}, \sigma \models \Phi_1$ и $\mathcal{A}, \sigma \models \Phi_2$
- $\mathcal{A}, \sigma \models \neg\Phi \quad \Leftrightarrow \quad \mathcal{A}, \sigma \not\models \Phi$

TCTL: семантика

Как и для CTL, для TCTL задаются два вида выполнимости:

- Выполнимость tctl-формулы Φ в конфигурации σ автомата $\mathcal{A}$: $\mathcal{A}, \sigma \models \Phi$
- Выполнимость формулы пути φ на дивергентной трассе τ автомата $\mathcal{A}$: $\mathcal{A}, \tau \models \varphi$

Эти отношения для автомата $\mathcal{A} = (S, s_0, \mathcal{I}, T, L)$ и формул над атомарными высказываниями AP и часами $\mathcal{C}$ задаются так:

- $\mathcal{A}, \sigma \models \mathbf{A}\varphi \quad \Leftrightarrow \quad$ для любой дивергентной σ-трассы τ автомата $\mathcal{A}$ верно $\mathcal{A}, \tau \models \varphi$
- $\mathcal{A}, \sigma \models \mathbf{E}\varphi \quad \Leftrightarrow \quad$ существует дивергентная σ-трасса τ автомата $\mathcal{A}$, такая что верно $\mathcal{A}, \tau \models \varphi$

TCTL: семантика

Как и для CTL, для TCTL задаются два вида выполнимости:

- Выполнимость tctl-формулы Φ в конфигурации σ автомата $\mathcal{A}$: $\mathcal{A}, \sigma \models \Phi$
- Выполнимость формулы пути φ на дивергентной трассе τ автомата $\mathcal{A}$: $\mathcal{A}, \tau \models \varphi$

Эти отношения для автомата $\mathcal{A} = (S, s_0, \mathcal{I}, T, L)$ и формул над атомарными высказываниями AP и часами $\mathcal{C}$ задаются так:

- $\mathcal{A}, \tau \models \Phi \mathbf{U} \Psi$, где $\tau = (\sigma_1 \to \sigma_2 \to \dots)$ $\Leftrightarrow$ существуют номер k, $k \in \mathbb{N}_0$, и конфигурация σ, покрываемая k-м шагом трассы τ, такие что
  - $\mathcal{A}, \sigma \models \Psi$
  - Если $k > 0$, то для любой конфигурации σ', покрываемой трассой
    $$\sigma_1 \to \sigma_2 \to \dots \to \sigma_k - 1 \to \sigma,$$
    верно $\mathcal{A}, \sigma' \models \Phi \vee \Psi$

Tctl-формула φ выполняется на автомате $\mathcal{A}$ ($\mathcal{A} \models \varphi$), если она выполняется в начальной конфигурации этого автомата

TCTL: семантика, особенность U

«Если $k > 0$, то для любой конфигурации σ' ... верно $\mathcal{A}, \sigma' \models \Phi \vee \Psi$»

В CTL в соответствующем месте семантики **U** вместо $\Phi \vee \Psi$ записывалось просто Φ

В содержательной трактовке **U** в TCTL говорится «а до тех пор верно Φ» (не $\Phi \vee \Psi$)

Можно легко убедиться в том, что семантика **U** в CTL не изменится, если в соответствующем месте написать «$\Phi \vee \Psi$» вместо «Φ»

При этом для реального времени различие существенно, и в этом можно убедиться на таком примере

TCTL: семантика, особенность U

«Если $k > 0$, то для любой конфигурации σ' ... верно $\mathcal{A}, \sigma' \models \Phi \vee \Psi$»

Пример автомата $\mathcal{A}$ и tctl-формулы φ:

$$\mathbf{A}((x \leq 1)\mathbf{U}(x > 1))$$

Все дивергентные вычисления этого автомата имеют вид

$$(\square, 0) \mapsto (\square, d_1) \mapsto (\square, d_2) \mapsto \dots$$

для неограниченно возрастающей последовательности чисел $d_1, d_2, \dots$

Содержательное прочтение формулы: «в любом вычислении автомата становится верным $x > 1$, и до тех пор верно $x \leq 1$»

Согласно прочтению формулы и согласно семантике **U**, верно $\mathcal{A} \models \varphi$

Если в семантике **U** записать «Φ» вместо «$\Phi \vee \Psi$», то формула не будет выполняться на $\mathcal{A}$:

- «становится верным $x > 1$»: выберем какую-либо конфигурацию со значением d часов x, где $d > 1$
- этой конфигурации в вычислении предшествуют покрытые конфигурации со значениями x из $(1, d)$, и для этих значений не выполняется неравенство $x \leq 1$

TCTL

В полном синтаксисе tctl-формул встречаются и другие привычные логические связки и темпоральные операторы:

- $\varphi \vee \psi = \neg(\neg\varphi \,\&\, \neg\psi)$
- $\varphi \to \psi = \neg\varphi \vee \psi$
- $\mathbf{F}\varphi = \mathbb{t}\mathbf{U}\varphi$
- $\mathbf{AG}\varphi = \neg\mathbf{EF}\neg\varphi$
- $\mathbf{EG}\varphi = \neg\mathbf{AF}\neg\varphi$

Содержательная трактовка операторов **F** и **G** тоже привычна с поправкой на реальное время:

- $\mathbf{F}\varphi$: в **реальном** будущем рано или поздно станет верным φ
- $\mathbf{G}\varphi$: всегда в **реальном** будущем будет верно φ

TCTL

Примеры tctl-формул напоследок:

- Как бы ни работал компьютер (*в реальном времени*), если он включен, то есть возможность его выключить

$$\mathbf{AG}(on \to \mathbf{EF}\neg on)$$

- Задача не может выполняться больше минуты (x — *часы, в которых отмеряется реальное время выполнения задачи в минутах*)

$$\mathbf{AG}(executing \to (x \leq 1))$$

- Если послан запрос, то неотвратимо не более чем за 5 минут на него будет получен отклик (x — *часы, в которых отмеряется реальное время с отсылки запроса в минутах*)

$$\mathbf{AG}(request \to \mathbf{AF}(response \,\&\, (x \leq 5)))$$

- Существует вычисление СРВ, в котором действия, выполнение которых сопровождается сбросом часов x, выполняются не реже чем раз в 5 *единиц времени*

$$\mathbf{EG}(x \leq 5)$$

Задача model checking для TCTL (MC-TCTL)

Для заданного корректного временно́го автомата $\mathcal{A}$
и заданной tctl-формулы Φ
над теми же множествами атомарных высказываний и часов
проверить справедливость соотношения

$$\mathcal{A} \models \Phi$$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 33

Сéти временны́х автоматов

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Для моделей Крипке существуют способы автоматического построения параллельной композиции (синхронной, асинхронной и смешанной)

Такое построение позволяет избежать незаметных, но при этом критичных ошибок, которые пользователь может внести в модель, пытаясь описать такую композицию вручную, и тем самым упрощает разработку модели и повышает уверенность в её правильности

СРВ, как и другие системы, как правило тоже состоят из параллельно выполняющихся компонентов

Ручное построение параллельной композиции временны́х автоматов, моделирующих компоненты СРВ — это процесс даже более подверженный ошибкам, чем то же для моделей Крипке

Поэтому критичными оказываются и средства автоматического построения параллельной композиции временны́х автоматов

Автомат с синхронизацией

Временно́й автомат с синхронизацией $(S, s_0, \mathcal{I}, T, L)$ над множествами атомарных высказываний AP, часов $\mathcal{C}$ и каналов синхронизации $\mathfrak{C}$ отличается от «обычного» временно́го автомата только тем, как устроены его переходы:

- $T \subseteq S \times CC(\mathcal{C}) \times 2^{\mathcal{C}} \times Sync(\mathfrak{C}) \times S$, где $Sync(\mathfrak{C})$ — множество действий синхронизации, устроенное так:
$$Sync(\mathfrak{C}) = \{\mathfrak{c}! \mid \mathfrak{c} \in \mathfrak{C}\} \cup \{\mathfrak{c}? \mid \mathfrak{c} \in \mathfrak{C}\} \cup \{\lambda\}$$

Действие λ означает, что переход автомата выполняется так же, как и в обычном временно́м автомате, асинхронно относительно остальных автоматов сети

Переход (s, g, X, α, s') будем изображать так: $s \xrightarrow{g,X,\alpha} s'$

Если переход не помечен действием синхронизации, то это означает действие λ

Автомат с синхронизацией

$$T \subseteq S \times CC(\mathcal{C}) \times 2^{\mathcal{C}} \times Sync(\mathfrak{C}) \times S$$

$$Sync(\mathfrak{C}) = \{\mathfrak{c}! \mid \mathfrak{c} \in \mathfrak{C}\} \cup \{\mathfrak{c}? \mid \mathfrak{c} \in \mathfrak{C}\} \cup \{\lambda\}$$

Переход с действием $\mathfrak{c}!$ обязан выполняться одновременно с переходом с парным действием $\mathfrak{c}?$, то есть два автомата с выбранными переходами выполняются синхронно, но при этом асинхронно относительно остальных автоматов сети

Такой тип синхронизации принято называть рандеву (синонимы: точка-точка, рукопожатие, handshake, peer-to-peer)

Действия $\mathfrak{c}!$ и $\mathfrak{c}?$ будут «симметричны» с точки зрения семантики, но тем не менее в связи с тем, как на практике разрабатываются автоматы с синхронизацией, будем называть действие $\mathfrak{c}!$ посылкой сигнала в канал $\mathfrak{c}$, а действие $\mathfrak{c}?$ — приёмом сигнала из канала $\mathfrak{c}$

Сеть временны́х автоматов: синтаксис

Сеть временны́х автоматов (далее для краткости просто сеть) над множествами атомарных высказываний AP, часов $\mathcal{C}$ и каналов синхронизации $\mathfrak{C}$ — это набор $\mathcal{N} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$, где:

- $\mathcal{A}_i = (S^i, s_0^i, \mathcal{I}^i, T^i, L^i)$, $1 \leq i \leq n$, — временной автомат с синхронизацией над теми же часами $\mathcal{C}$ и каналами $\mathfrak{C}$ и над множеством атомарных высказываний AP_i, где $\mathrm{AP}_i \subseteq \mathrm{AP}$
- $\mathrm{AP}_i \cap \mathrm{AP}_j = \emptyset$ для любых номеров i, j, таких что $1 \leq i < j \leq n$
- $\mathrm{AP}_1 \cup \cdots \cup \mathrm{AP}_n = \mathrm{AP}$
- $S_i \cap S_j = \emptyset$ для любых номеров i, j, таких что $1 \leq i < j \leq n$

Сеть временны́х автоматов: синтаксис

Пример

Смоделируем в виде сети автоматов такую систему:

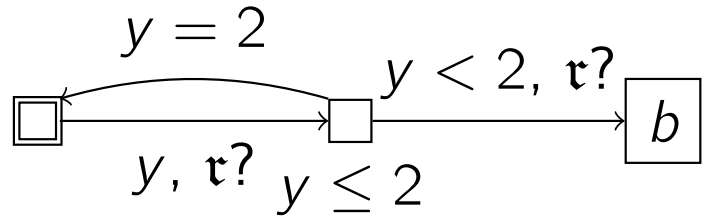
- Пользователь может посылать запросы на сервер
- *Из физических соображений* полагаем, что запросы посылаются не чаще чем раз в секунду
- Если сервер получил два запроса с интервалом строго менее двух секунд, то он ломается

Заведём в сети два автомата: один для пользователя, и один для сервера

Запрос смоделируем как сигнал в канале $\mathfrak{r}$

То, что сервер сломан, смоделируем как атомарное высказывание b

Тогда сеть может быть устроена так:



Сеть временны́х автоматов: семантика

Рассмотрим сеть $\mathcal{N} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ над множеством часов $\mathcal{C}$, где $\mathcal{A}_i = (S^i, s_0^i, \mathcal{I}^i, T^i, L^i)$

Конфигурация сети $\mathcal{N}$ — это пара $(\vec{s}, \nu)$, где $\vec{s} \in S^1 \times \cdots \times S^n$ и ν — оценка часов множества $\mathcal{C}$

Начальная конфигурация сети $\mathcal{N}$ — это $((s_0^1, \ldots, s_0^n), (0, 0, \ldots, 0))$

Шаг вычисления $\to_{\mathcal{N}}$ сети $\mathcal{N}$ — это двуместное отношение на множестве конфигураций, являющееся объединением трёх:

- Продвижение времени: $\sigma \mapsto_{\mathcal{N}} \sigma'$
- Выполнение перехода: $\sigma \hookrightarrow_{\mathcal{N}} \sigma'$
- Шаг рандеву: $\sigma \Rightarrow_{\mathcal{N}} \sigma'$

Сеть временны́х автоматов: семантика

Рассмотрим сеть $\mathcal{N} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ над множеством часов $\mathcal{C}$, где $\mathcal{A}_i = (S^i, s_0^i, \mathcal{I}^i, T^i, L^i)$

Для оценки часов ν, константы d, $d \in \mathbb{R}_{\geq 0}$, множества часов X, конфигурации σ и перехода t ряд обозначений дословно переносится с модели автомата на модель сети:

- $\nu + d$
- $\sigma + d$
- $\sigma \overset{d}{\mapsto} \sigma'$
- $\nu[X]$
- $\sigma[X]$

$\sigma[s/s']$ — так будем обозначать конфигурацию, получающуюся из σ заменой состояния s одного из автоматов сети на s'

$\sigma \xhookrightarrow{s_i \xrightarrow{g,X,\alpha} s_i'} \sigma'$ означает, что $\sigma' = \sigma[X][s_i/s_i']$

Сеть временны́х автоматов: семантика

Рассмотрим сеть $\mathcal{N} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ над множеством часов $\mathcal{C}$, где $\mathcal{A}_i = (S^i, s_0^i, \mathcal{I}^i, T^i, L^i)$, и конфигурации $\sigma = ((s_1, \ldots, s_n), \nu)$ и σ'

Продвижение времени

$\sigma \mapsto_{\mathcal{N}} \sigma'$, если существует константа $d \in \mathbb{R}_{>0}$, такая что:

1. $\sigma \overset{d}{\mapsto} \sigma'$
2. $\nu + d \models \mathcal{I}^1(s_1) \,\&\, \ldots \,\&\, \mathcal{I}^n(s_n)$

Выполнение перехода

$\sigma \hookrightarrow_{\mathcal{N}} \sigma'$, если в сети $\mathcal{N}$ есть переход $t = (s_k \xrightarrow{g,X,\lambda} s_k')$, такой что:

1. $\nu \models g$
2. $\sigma \overset{t}{\hookrightarrow} \sigma'$
3. $\nu[X] \models \mathcal{I}^k(s_k')$

Сеть временны́х автоматов: семантика

Рассмотрим сеть $\mathcal{N} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ над множеством часов $\mathcal{C}$, где $\mathcal{A}_i = (S^i, s_0^i, \mathcal{I}^i, T^i, L^i)$, и конфигурации $\sigma = ((s_1, \ldots, s_n), \nu)$ и σ'

Шаг рандеву

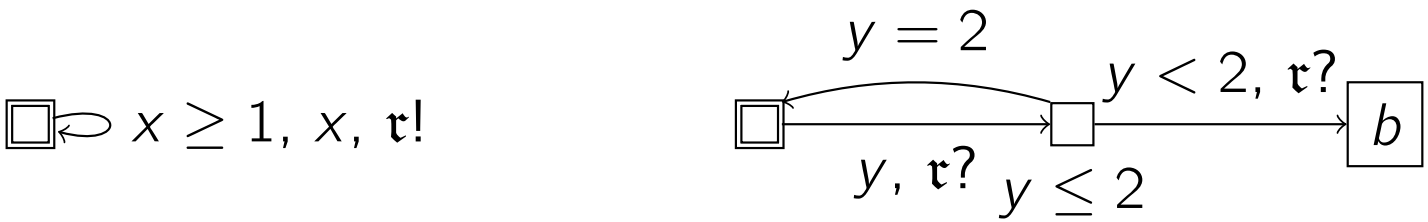
$\sigma \Rightarrow_{\mathcal{N}} \sigma'$, если в сети $\mathcal{N}$ есть канал $\mathfrak{c}$ и переходы $t_1 = (s_k \xrightarrow{g_k, X_k, \mathfrak{c}!} s'_k)$ и $t_2 = (s_m \xrightarrow{g_m, X_m, \mathfrak{c}?} s'_m)$, такие что:

- $k \neq m$
- $\nu \models g_k \,\&\, g_m$
- Для некоторой конфигурации σ'' верно $\sigma \xhookrightarrow{t_1} \sigma'' \xhookrightarrow{t_2} \sigma'$
- $\nu[X_k][X_m] \models \mathcal{I}^k(s'_k) \,\&\, \mathcal{I}^m(s'_m)$

Понятия, основанные на отношении $\rightarrow_{\mathcal{N}}$, дословно переносятся с модели временно́го автомато на модель сети: трасса (в том числе начальная, тупиковая, конвергентная, дивергентная, зеноновская), вычисление, корректность

Сеть временны́х автоматов: семантика

Пример



Это корректная сеть над часами $\{x, y\}$, и вот пример её вычисления для порядка часов (x, y) и для имён состояний (слева направо) n, ℓ, c, r:

$$
\begin{aligned}
&((n, \ell), (0, 0)) \\
\mapsto\ &((n, \ell), (\sqrt{2}, \sqrt{2})) \\
\Rightarrow\ &((n, c), (0, 0)) \\
\mapsto\ &((n, c), (2, 2)) \\
\hookrightarrow\ &((n, \ell), (2, 2)) \\
\Rightarrow\ &((n, c), (0, 0)) \\
\mapsto\ &((n, c), (1, 1)) \\
\Rightarrow\ &((n, r), (0, 1)) \\
\mapsto\ &((n, r), (1, 2)) \\
\mapsto\ &((n, r), (2, 3)) \\
\mapsto\ &\ldots
\end{aligned}
$$

Трансляция сетей в автомат

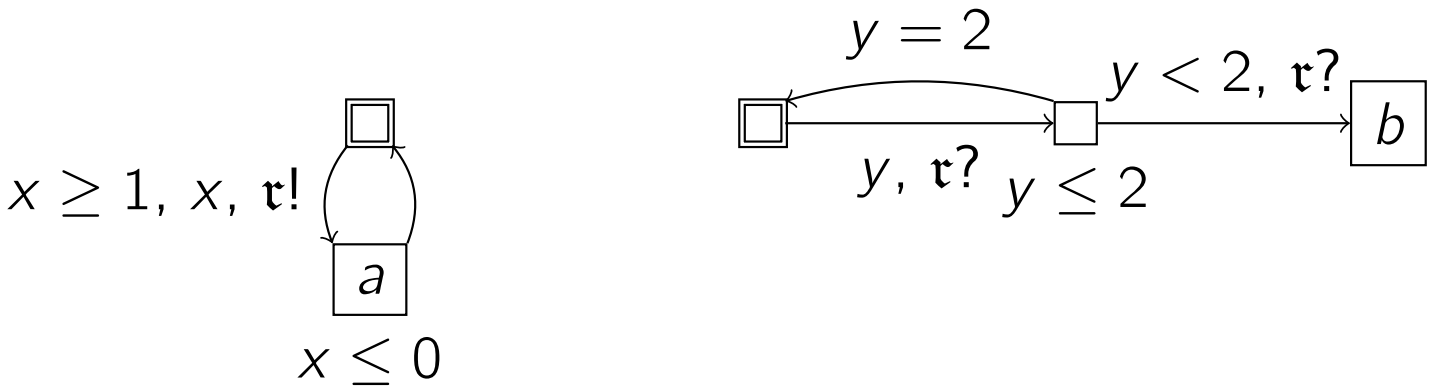
Рассмотрим произвольную сеть $\mathcal{N} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$, где $\mathcal{A}_i = (S^i, s_0^i, \mathcal{I}^i, T^i, L^i)$

Записью $\otimes\mathcal{N}$ обозначим временной автомат $(S, s_0, \mathcal{I}, T, L)$ над теми же атомарными высказываниями и часами, устроенный так:

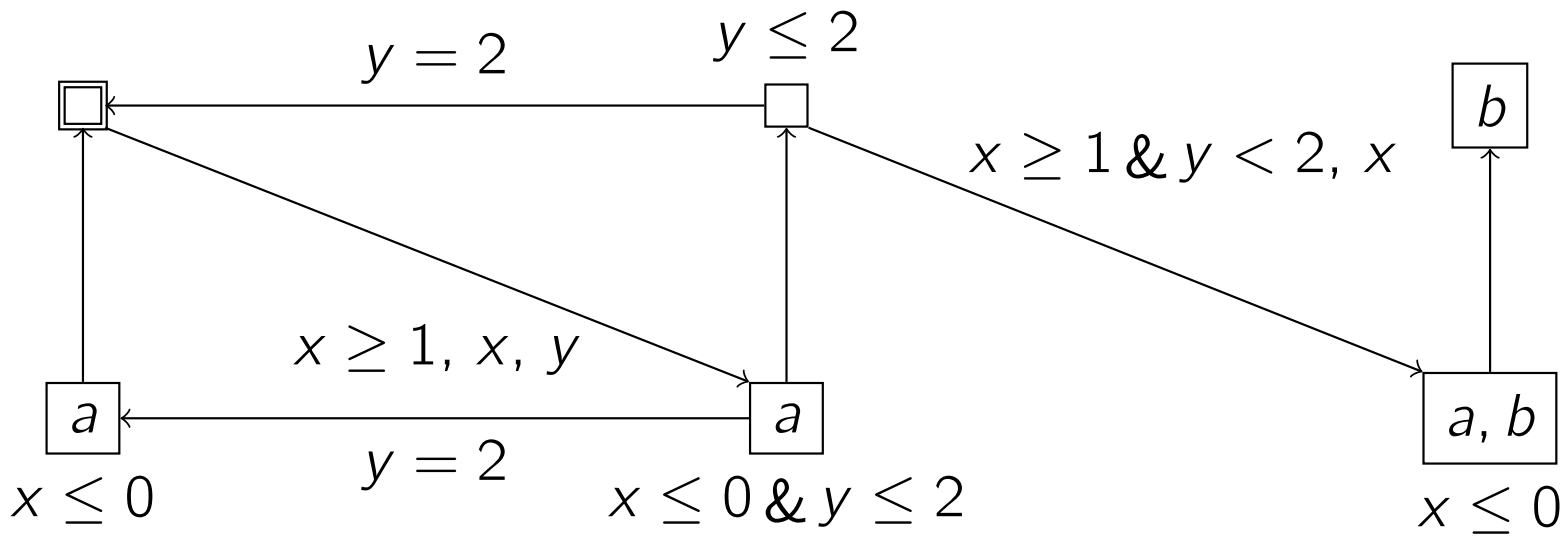
- $S = S^1 \times \cdots \times S^n$
- $s_0 = (s_0^1, \ldots, s_0^n)$
- $\mathcal{I}(s^1, \ldots, s^n) = \mathcal{I}^1(s^1) \,\&\, \ldots \,\&\, \mathcal{I}^n(s^n)$
- $L(s^1, \ldots, s^n) = L(s^1) \cup \cdots \cup L(s^n)$
- В T входят следующие переходы и только они:
  - $(s^1, \ldots, s^n) \xrightarrow{g, X} (s^1, \ldots, s^{i-1}, q^i, s^{i+1}, \ldots, s^n)$, если $(s^i \xrightarrow{g, X, \lambda} q^i) \in T^i$
  - $(s^1, \ldots, s^n) \xrightarrow{g_1 \,\&\, g_2, X_1 \cup X_2} (s^1, \ldots, s^{i-1}, q^i, s^{i+1}, \ldots, s^{j-1}, q^j, s^{j+1}, \ldots, s^n)$, если $i < j$ и существует канал $\mathfrak{c}$, такой что $(s^i \xrightarrow{g_1, X_1, \alpha_1} q^i) \in T^i$ и $(s^j \xrightarrow{g_2, X_2, \alpha_2} q^j) \in T^j$ и $\{\alpha_1, \alpha_2\} = \{\mathfrak{c}!, \mathfrak{c}?\}$

Трансляция сетей в автомат

Пример



Автомат $\otimes\mathcal{N}$ для сети $\mathcal{N}$ из двух автоматов, изображённой выше:



Трансляция сетей в автомат

Сеть $\mathcal{N}$ и временной автомат $\mathcal{A}$ будем называть эквивалентными ($\mathcal{N} \sim \mathcal{A}$), если

- они определены над одинаковыми множествами атомарных высказываний и часов и
- отношения $\to_{\mathcal{N}}$ и $\to_{\mathcal{A}}$ совпадают

Теорема. **Для любой сети $\mathcal{N}$ верно: $\mathcal{N} \sim \otimes\mathcal{N}$**

Доказательство этой теоремы можно считать нетрудным упражнением

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 34

Алгоритм model checking для TCTL
Временны́е регионы
Системы регионов

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Задача MC-TCTL: для заданного корректного временно́го автомата $\mathcal{A}$ и заданной tctl-формулы φ проверить соотношение $\mathcal{A} \models \varphi$

Задача MC-CTL: для заданной модели Крипке M и заданной ctl-формулы φ проверить соотношение $M \models \varphi$

Синтаксис CTL строго шире синтаксиса TCTL

Семантика CTL похожа на семантику TCTL, но существенно различается из-за особенностей отсчёта времени

Эти две логики настолько похожи друг на друга, что можно попробовать *свести* решение MC-TCTL к решению MC-CTL

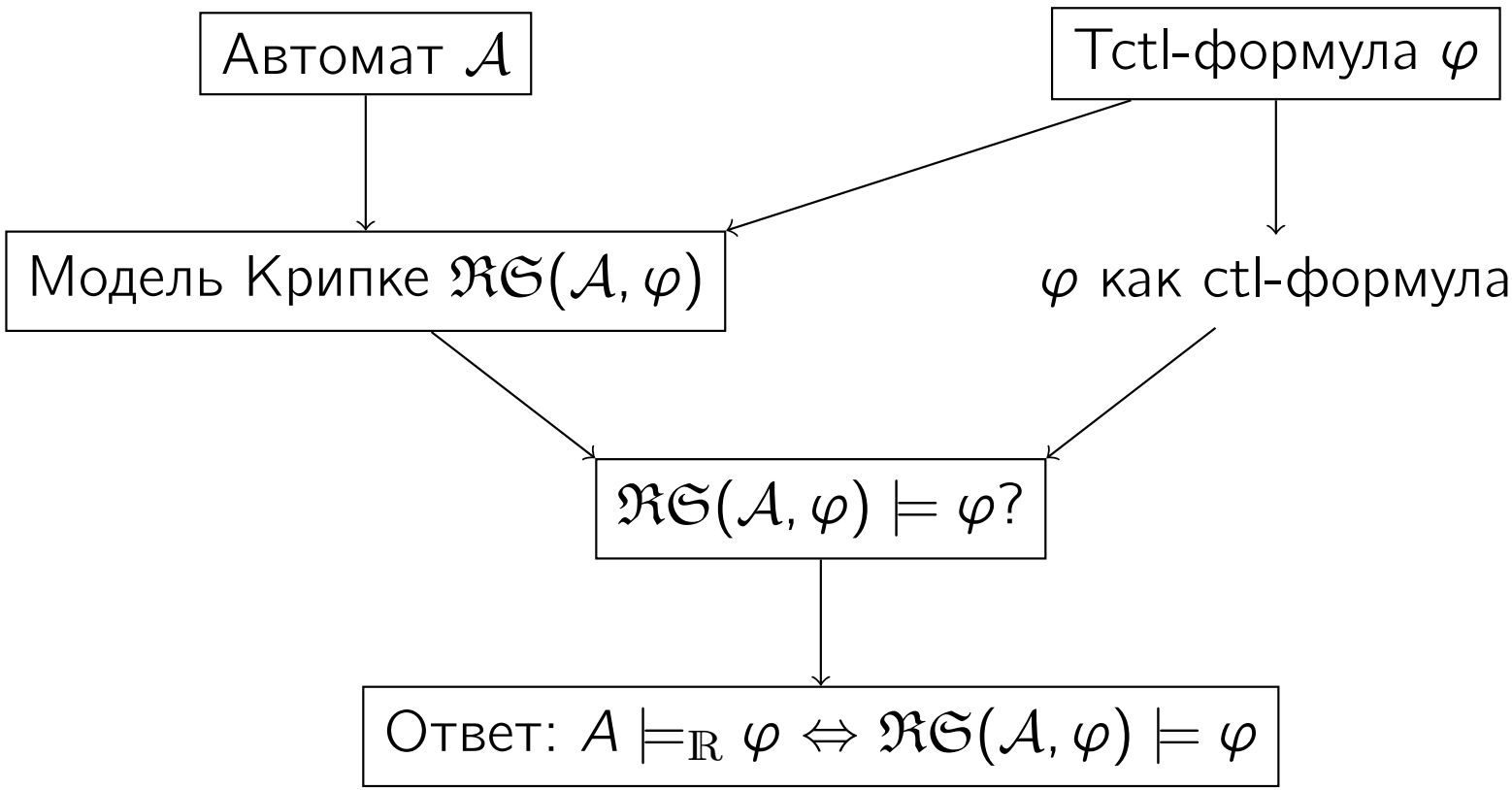
Чтобы нагляднее различать отношения выполнимости в смысле CTL и в смысле TCTL, будем отношение выполнимости для TCTL записывать так: $\models_{\mathbb{R}}$

Общая схема MC-TCTL

Дано: временной автомат $\mathcal{A}$, tctl-формула φ над множествами атомарных высказываний AP и часов $\mathcal{C}$

Требуется проверить справедливость соотношения $\mathcal{A} \models_{\mathbb{R}} \varphi$

Схема проверки:



Модель $\mathfrak{RG}(\mathcal{A}, \varphi)$ будет называться системой регионов для автомата $\mathcal{A}$ и формулы φ

Общая схема MC-TCTL

$AC(\mathcal{A})$ и $AC(\varphi)$ — так будем обозначать все атомарные временны́е ограничения, содержащиеся соответственно в $\mathcal{A}$ и в φ

Система $\mathfrak{RG}(\mathcal{A}, \varphi)$ будет строиться над множеством $\mathrm{AP} \cup AC(\varphi)$

(*и тогда можно считать φ ctl-формулой*)

Каждая конфигурация автомата будет отвечать некоторому состоянию системы $\mathfrak{RG}(\mathcal{A}, \varphi)$:

- Множесто всех оценок часов будет разбито на классы эквивалентности (регионы)
- Оценка ν будет отвечать её региону $[\nu]$
- Конфигурация (s, ν) будет отвечать состоянию $(s, [\nu])$
- $[(0, 0, \ldots, 0)] = \{(0, 0, \ldots, 0)\}$

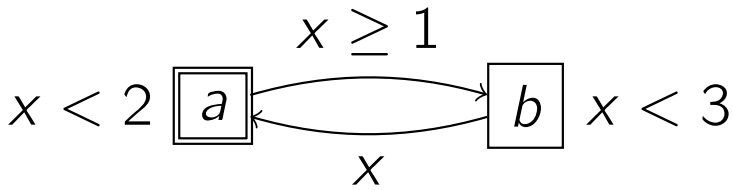
Шаг вычисления $(s_1, \nu_1) \to (s_2, \nu_2)$ автомата будет отвечать пути $(s_1, [\nu_1]) \to \cdots \to (s_2, [\nu_2])$ в системе регионов

(*и, в частности, все покрывающиеся конфигурации станут явными*)

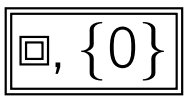
Состояние $(s, [\nu])$ будет размечаться высказываниями из AP согласно s и ограничениями из $AC(\varphi)$ согласно ν

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



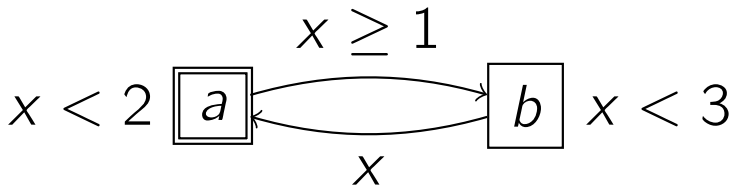
AGAF$(x = 1)$



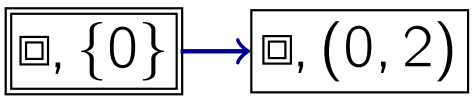
Единственное начальное состояние модели — это ▣ со значением 0 часов x

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$

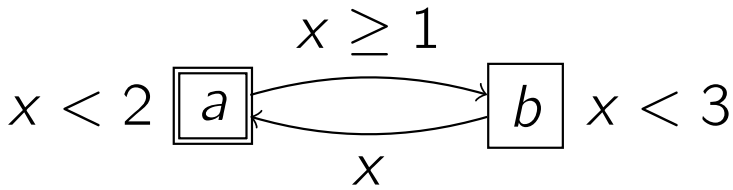


Начав вычисление в $(\boxdot, 0)$, $\mathcal{A}$ обязан продвинуть время, и может продвинуть часы до любого значения интервала $(0, 2)$

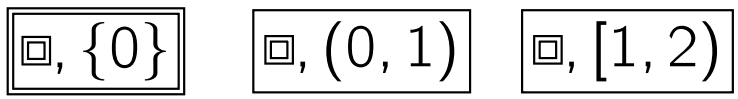
Для начала запишем все такие продвижения времени как один переход в модели

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$

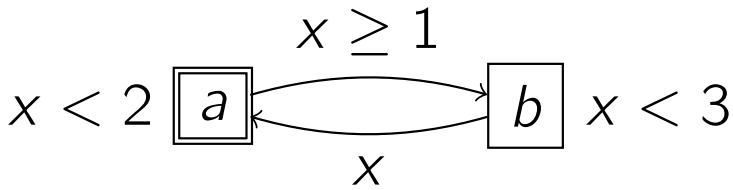


Для значений часов из $[1, 2)$ открыт верхний переход автомата, а для значений из $(0, 1)$ этот переход закрыт

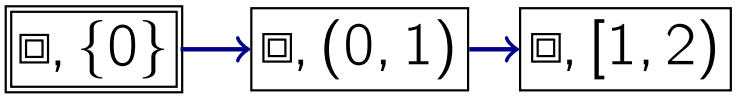
Чтобы **детерминированно** воспроизвести шаги вычисления автомата, следует разбить состояние $(\boxdot, (0, 2))$ на два: $(\boxdot, (0, 1))$ и $(\boxdot, [1, 2))$

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$

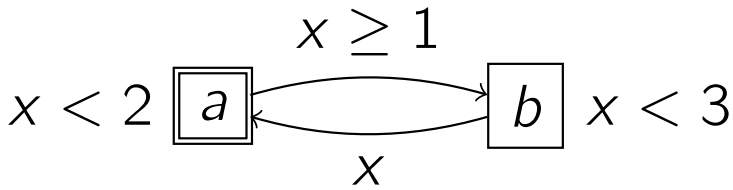


Когда автомат **непрерывно** ожидает (продвигает время), начав в $(▣, 0)$, значение часов последовательно проходит через интервалы $\{0\}$, $(0, 1)$ и $[1, 2)$

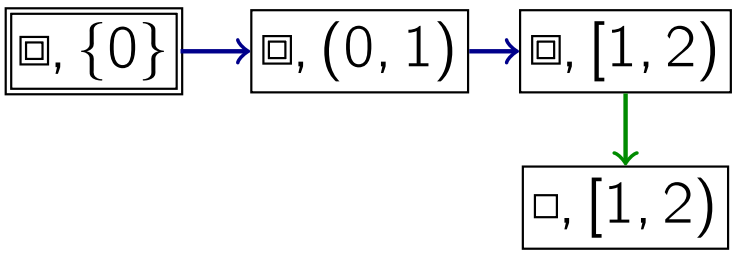
Чтобы воспроизвести все варианты такого ожидания, соединим соответствующие состояния по порядку

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$



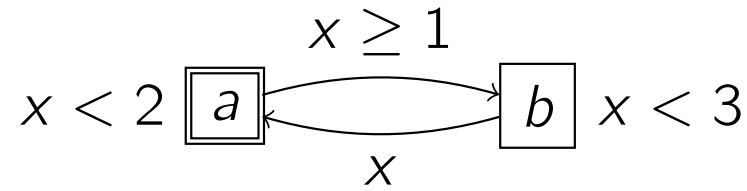
Для каждой конфигурации вида $(\boxdot, d)$, где $1 \leq d < 2$,

верно соотношение $(\boxdot, d) \xrightarrow{\boxdot \xrightarrow{x \geq 1} \square} (\square, d)$

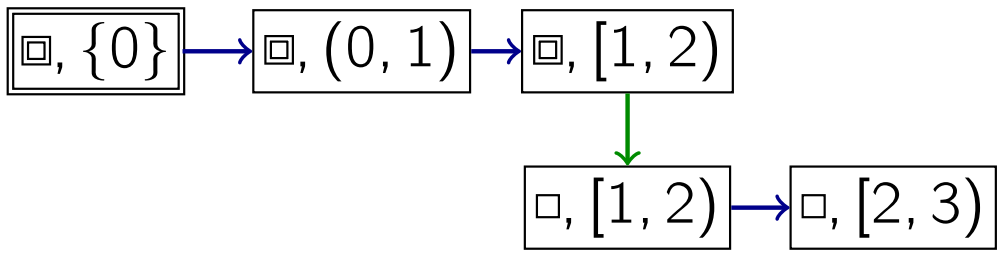
Добавим в модель переход, воспроизводящий все такие шаги вычисления автомата

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$

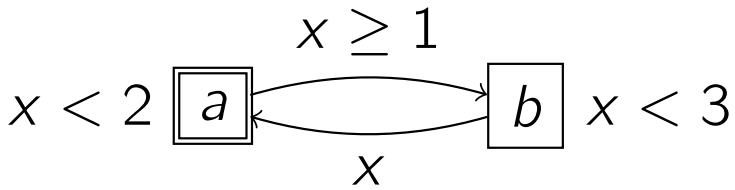


Когда автомат ожидает, начав в $(\square, d)$, где $1 \leq d < 2$, значение часов может пройти через остаток интервала $[1, 2)$ и через часть интервала $[2, 3)$

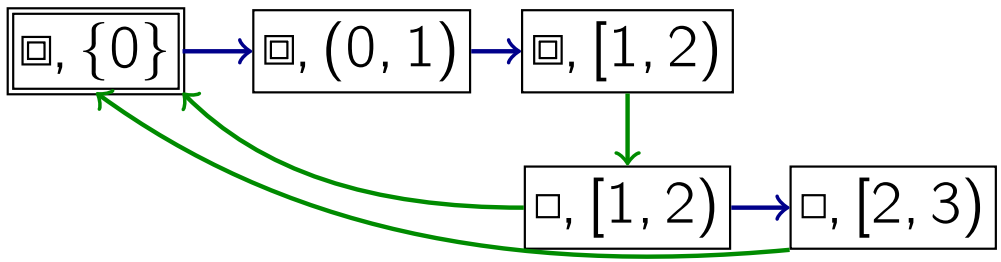
Добавим в модель переход, воспроизводящий такое ожидание

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$



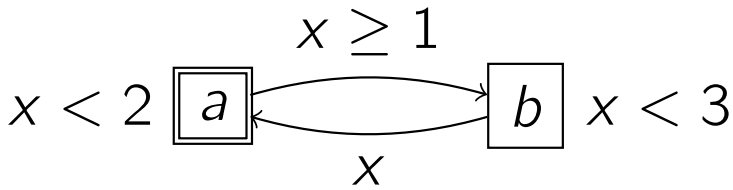
Для каждой конфигурации $(\square, d)$, где $1 \leq d < 3$,

верно соотношение $(\square, d) \xhookrightarrow{\square \xrightarrow{x} \boxdot} (\boxdot, 0)$

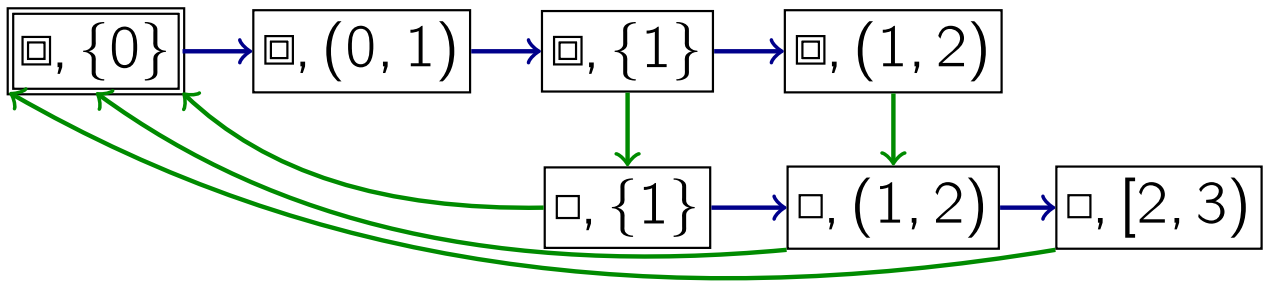
Добавим в модель переходы, отвечающие всем таким шагам вычисления автомата

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$

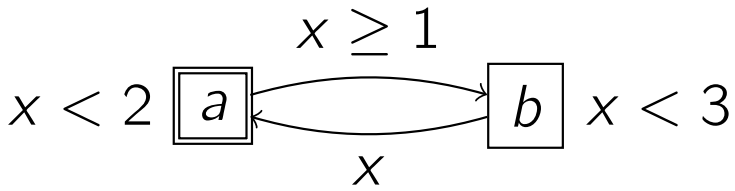


$(x = 1) \equiv (x \leq 1 \,\&\, \neg(x < 1))$:
в формуле φ содержатся ограничения $x \leq 1$ и $x < 1$

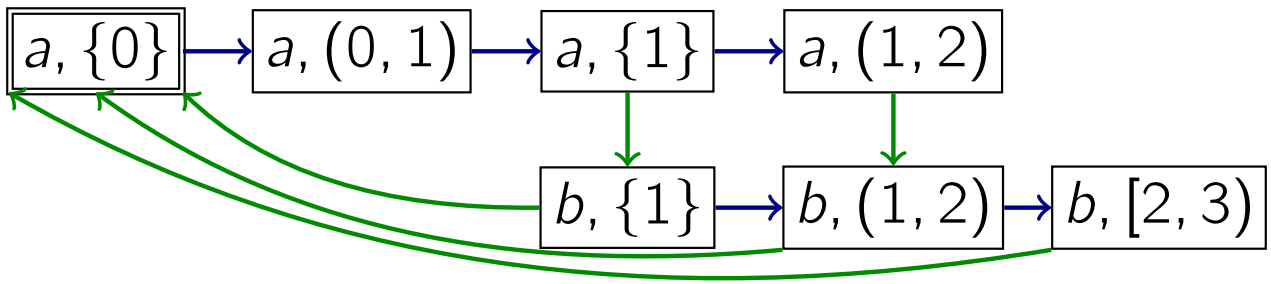
Чтобы **детерминированно** разметить состояния модели этими ограничениями, следует разбить в каждом состоянии модели интервал $[1, 2)$ на два: $\{1\}$ и $(1, 2)$

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$

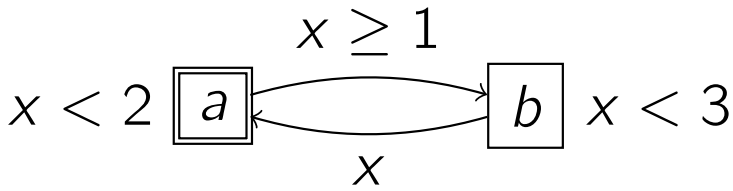


Получилась модель Крипке M, содержащая
в точности все шаги вычисления автомата $\mathcal{A}$
и все неявно покрытые конфигурации таких шагов

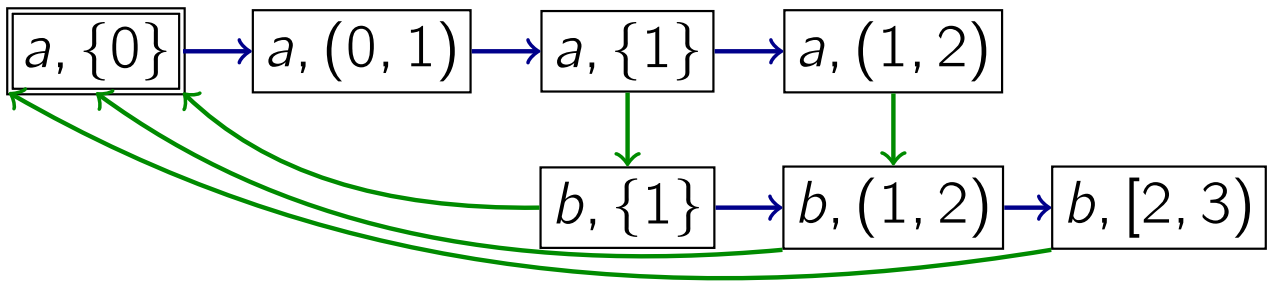
Нетрудно видеть, что $A \models_{\mathbb{R}} \varphi$ и $M \models \varphi$

Общая схема MC-TCTL

Пример: попробуем при помощи «пристального взгляда» построить подходящую модель Крипке (хотя и не в точности $\mathfrak{RG}(\mathcal{A}, \varphi)$) для таких автомата $\mathcal{A}$ и tctl-формулы φ:



AGAF$(x = 1)$



Теперь перейдём к трудной части: в общем случае ...

- как устроить множества оценок (регионы), чтобы обеспечить требуемое соответствие, детерминированность и конечность?
- как совместить состояния автомата, регионы и переходы, чтобы чудесным образом превратить «$\models_{\mathbb{R}}$» в «$\models$»?

Временны́е регионы

Разбиение оценок часов на классы основывается на региональном отношении эквивалентности оценок часов ($\approx$)

Полное подробное определение этого отношения будет приведено далее, и оно будет описываться постепенно (поэтапно)

Временной регион — это класс эквивалентности отношения $\approx$

Временны́е регионы — это множества оценок часов, использующиеся в качестве второго компонента состояния системы регионов

$\mathfrak{R}$ — так будем обозначать семейство всех временны́х регионов

Временны́е регионы

Особенности устройства отношения $\approx$, которые необходимы для соответствия содержательному краткому описанию системы регионов и для всех технических тонкостей, возникших в примере, строго можно определить так:

- Конечность: общее число классов эквивалентности отношения $\approx$ конечно
  - $\Rightarrow$ множество состояний системы регионов конечно, и можно к ней применять известные алгоритмы анализа конечных моделей Крипке
- Неразличимость временны́ми ограничениями: если $\nu_1 \approx \nu_2$ и $ag \in AC(\mathcal{A}) \cup AC(\varphi)$, то $\nu_1 \models ag \Leftrightarrow \nu_2 \models ag$
  - $\Rightarrow$ детерминированность относительно предусловий и разметки состояний
- Корректный сброс: если ρ — регион и X — множество часов, то $\rho[X] = \{\nu[X] \mid \nu \in \rho\}$ — тоже регион
  - $\Rightarrow$ детерминированность относительно сброса часов при выполнении переходов автомата
- Корректное ожидание: для любого региона ρ существует единственный регион ρ^+, следующий за ρ при непрерывном ожидании автомата
  - $\Rightarrow$ детерминированность относительно продвижения времени автоматом

Временны́е регионы

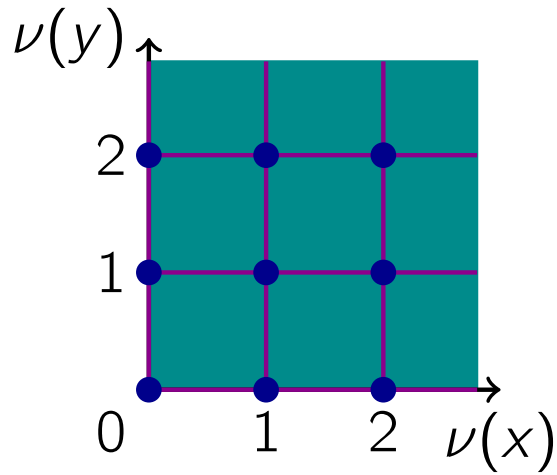
Первая (неудачная) попытка определить $\approx$

$\lfloor t \rfloor$ и $frac(t)$ — так будем обозначать соответственно целую и дробную часть числа действительного числá t

$\nu_1 \approx_1 \nu_2 \quad \Leftrightarrow \quad$ для любых часов x верно следующее:

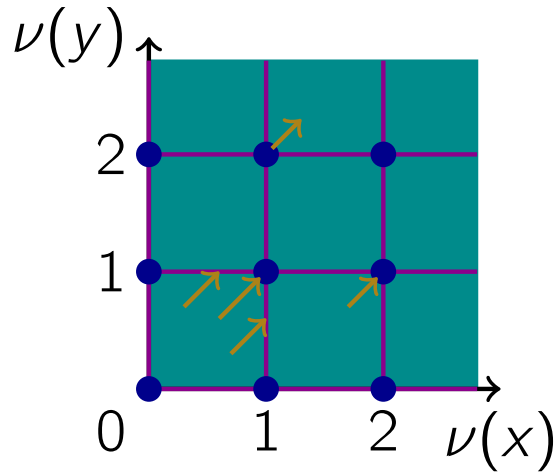
1. $\lfloor \nu_1(x) \rfloor = \lfloor \nu_2(x) \rfloor$
2. $frac(\nu_1(x)) = 0 \Leftrightarrow frac(\nu_2(x)) = 0$

Пример: регионы отношения $\approx_1$ для пары часов x, y изображены ниже как связные одноцветные области числовой плоскости



Временны́е регионы

Первая (неудачная) попытка определить $\approx$



Хорошие свойства $\approx_1$:

- Неразличимость временны́ми ограничениями
- Корректный сброс

Плохие свойства $\approx_1$:

- $|\mathfrak{R}| = \infty$
- Невозможно однозначно определить ρ^+
  - примеры пар (ρ, ρ^+) изображены выше стрелками

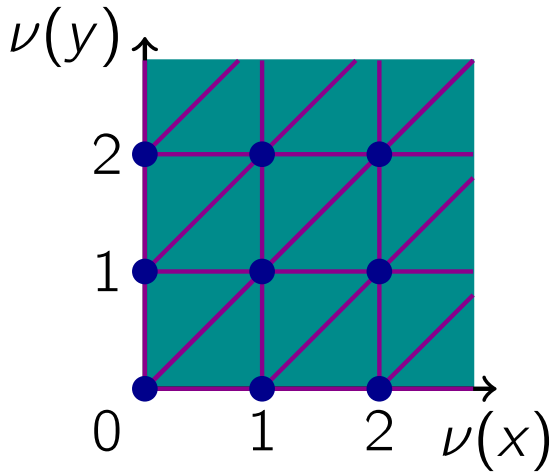
Временны́е регионы

Вторая (неудачная) попытка определить $\approx$

$\nu_1 \approx_2 \nu_2 \quad \Leftrightarrow \quad$ для любых часов x, y верно следующее:

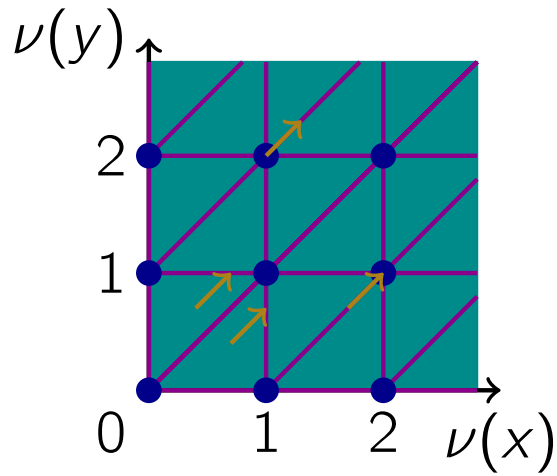
1. $\lfloor \nu_1(x) \rfloor = \lfloor \nu_2(x) \rfloor$
2. $frac(\nu_1(x)) = 0 \Leftrightarrow frac(\nu_2(x)) = 0$
3. $frac(\nu_1(x)) \leq frac(\nu_1(y)) \Leftrightarrow frac(\nu_2(x)) \leq frac(\nu_2(y))$

Пример: регионы отношения $\approx_2$ для пары часов x, y изображены ниже как связные одноцветные области числовой плоскости



Временны́е регионы

Вторая (неудачная) попытка определить $\approx$



Хорошие свойства $\approx_2$:

- Неразличимость временны́ми ограничениями
- Корректный сброс
- Корректное ожидание

Плохие свойства $\approx_2$:

- $|\mathfrak{R}| = \infty$

Временны́е регионы

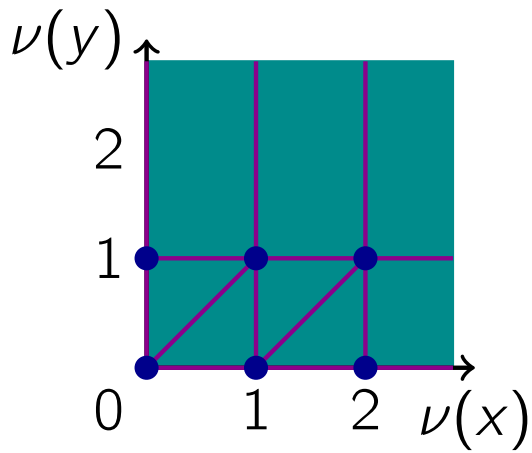
Определение $\approx$ (*третья попытка, удачная*)

k_x — так будем обозначать максимальное целое число, встречающееся в правых частях ограничений из $AC(\mathcal{A}) \cup AC(\varphi)$

$\nu_1 \approx \nu_2 \quad \Leftrightarrow \quad$ для любых часов x, y верно следующее:

1. $\nu_1(x) > k_x \Leftrightarrow \nu_2(x) > k_x$
2. если $\nu_1(x) \le k_x$ и $\nu_1(y) \le k_y$, то
   - $\lfloor \nu_1(x) \rfloor = \lfloor \nu_2(x) \rfloor$,
   - $frac(\nu_1(x)) = 0 \Leftrightarrow frac(\nu_2(x)) = 0$ и
   - $frac(\nu_1(x)) \le frac(\nu_1(y)) \Leftrightarrow frac(\nu_2(x)) \le frac(\nu_2(y))$

Пример: регионы для пары часов x, y и констант $k_x = 2$, $k_y = 1$ изображены ниже как связные одноцветные области числовой плоскости



Оценка числá регионов

Утверждение. $|\mathcal{C}|! \cdot \prod\limits_{x \in \mathcal{C}} k_x \leq |\mathfrak{R}| \leq |\mathcal{C}|! \cdot 2^{|\mathcal{C}|-1} \cdot \prod\limits_{x \in \mathcal{C}} (2k_x + 2)$

Доказательство. Ограничимся пояснениями всех множителей в оценках:

- $\prod\limits_{x \in C} k_x$ — это количество единичных $|\mathcal{C}|$-мерных кубов, которыми можно покрыть декартово произведение всех интервалов $[0, k_x]$
- $|C|!$ — столькими способами можно упорядочить дробные части значений часов
  - В оценке снизу: по крайней мере столько регионов содержится во внутренности одного единичного куба
- $2k_x + 2$ — это общее число попарно различных интервалов значений часов x в регионах: $\{0\}; (0, 1); \{1\}; (1, 2); \{2\}; \ldots$
- $2^{|C|-1}$ — столькими способами можно для заданного порядка дробных частей объявить, какие из этих дробных частей равны ▼

Следствие (конечность отношения $\approx$). $|\mathfrak{R}| < \infty$

Другие свойства регионов

Утверждение (неразличимость временны́ми ограничениями)
Для любых часов x, оценок ν_1 и ν_2, таких что $\nu_1 \approx \nu_2$, и числа́ $k \in \mathbb{N}_0$ верно:

$$\nu_1 \models x < k \quad \Leftrightarrow \quad \nu_2 \models x < k \quad \textbf{и}$$
$$\nu_1 \models x \leq k \quad \Leftrightarrow \quad \nu_2 \models x \leq k$$

Временно́е ограничение g над атомарными ограничениями $ACC_A \cup ACC_\varphi$ выполняется в регионе ρ ($\rho \models g$), если
для любой оценки часов ν из ρ верно $\nu \models g$

Утверждение (корректный сброс). **Для любого региона ρ и любого множества часов X множество $\rho[X]$ является регионом**

Другие свойства регионов

Регион называется открытым для часов x, если он содержит оценку ν, такую что $\nu(x) > k_x$

Регион называется открытым, если он открыт для всех часов, а иначе — закрытым

ρ^+ — это регион, следующий за регионом ρ:

- Если ρ открыт, то $\rho^+ = \rho$
- Иначе ρ^+ — регион, для которого верно следующее:
  - $\rho^+ \neq \rho$
  - Если $\nu \in \rho$ и $(\nu + d) \in \rho^+$, где $d \in \mathbb{R}_{>0}$, то для любого d' из $\mathbb{R}_{>0}$, такого что $d' < d$, верно $(\nu + d') \in \rho \cup \rho^+$

Утверждение (корректное ожидание)

За любым регионом ρ следует ровно один регион

Для лучшего понимания регионов можете попробовать строго доказать последние три утверждения

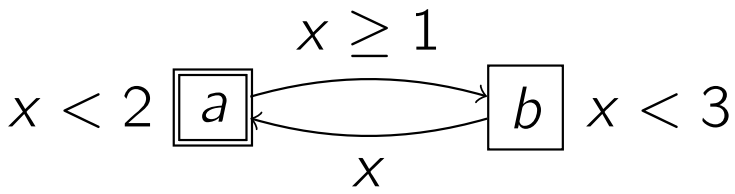
Системы регионов

Для временно́го автомата $\mathcal{A} = (S, s_0, \mathcal{I}, T, L)$ и tctl-формулы φ ...

- региональным состоянием будем называть пару $(s, \rho) \in S \times \mathfrak{R}$
- системой регионов будем называть модель Крипке $\mathfrak{RS}(\mathcal{A}, \varphi)$, задающуюся как подграф следующего графа Γ, порождённый множеством всех вершин, достижимых из начальной:
  - Вершины Γ — это всевозможные региональные состояния
  - Вершина $(s_0, \{(0, 0, \ldots, 0)\})$ — начальная
  - Каждая вершина (s, ρ) помечена множеством
    $$L(s) \cup \{ag | ag \in AC(\varphi),\ \rho \models ag\}$$
  - Дуга $(s, \rho) \to (s', \rho')$ входит в Γ $\Leftrightarrow$ верно хотя бы одно из двух:
    1. $\rho' = \rho^+$, $s' = s$ и $\rho^+ \models \mathcal{I}(s)$
    2. В $\mathcal{A}$ существует переход $s \xrightarrow{g,X} s'$, такой что $\rho \models g$, $\rho' = \rho[X]$, and $\rho' \models \mathcal{I}(s')$

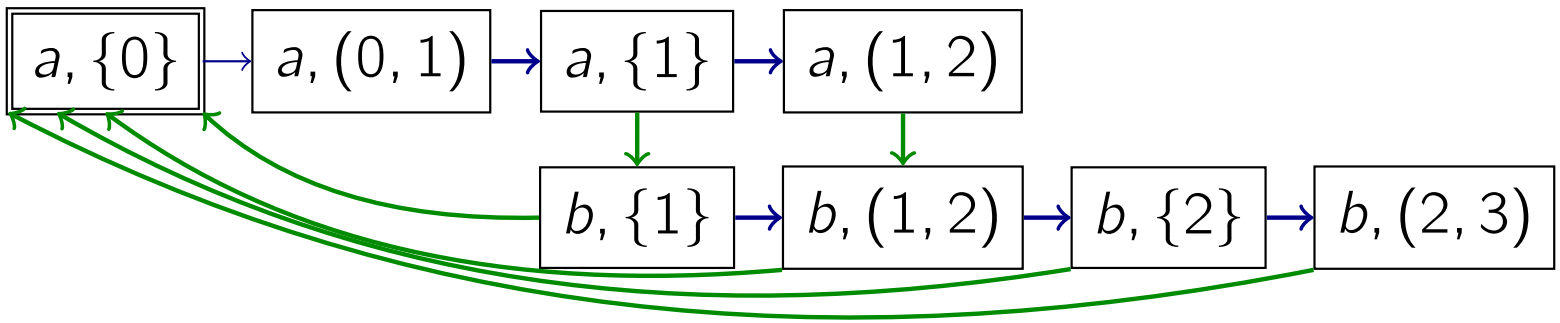
Системы регионов

Пример



AGAF$(x = 1)$

Система регионов для изображённых автомата и формулы устроена так (*атомарные временны́е ограничения, помечающие состояние, изображены как подходящие интервалы значений часов* x):



Теорема. **Для любого корректного временно́го автомата $\mathcal{A}$ и любой tctl-формулы φ верно:**

$$\mathcal{A} \models_{\mathbb{R}} \varphi \quad \Leftrightarrow \quad \mathfrak{RG}(\mathcal{A}, \varphi) \models \varphi$$

Доказательство опустим: его объём и трудность намного превосходят пользу включения его в рассказ

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 35

Отношения симуляции и бисимуляции
Симуляционная и бисимуляционная эквивалентности

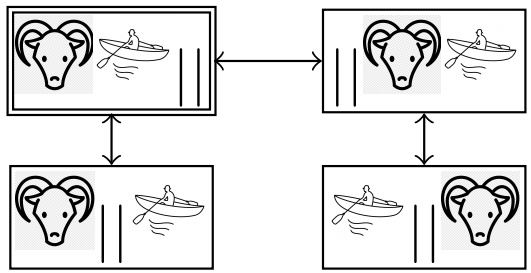
Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

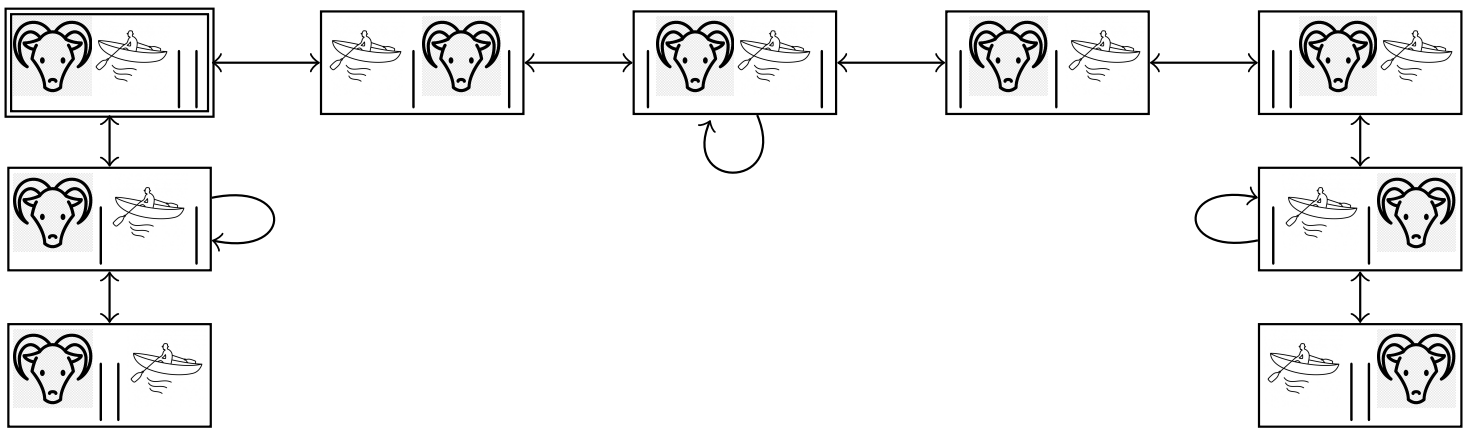
Вступление

Пример/наблюдение 1

Модели Крипке бывают простые



А бывают и сложные



Вступление

Пример/наблюдение 1

Сложные модели придумываются для

- точного соответствия поведению исходной системы и
- детального воспроизведения шагов выполнения системы

Но такие модели трудно строить (вычислять) и трудно анализировать

С другой стороны, простые модели

- легко строятся и
- легко анализируются

Но простой моделью лишь приблизительно (неточно) описывается поведение исходной системы и предоставляется лишь поверхностная информация о шагах выполнения системы

Вступление

Пример/наблюдение 1

Хотелось бы иметь возможность «переключать» степень простоты модели, чтобы использовать преимущества, отвечающие текущим целям исследования системы

Например, было бы неплохо уметь разрабатывать точную детальную модель так, чтобы она затем не слишком трудно перестраивалась в простую для достаточно эффективной верификации

Но для этого требуются гарантии того, что две заданные модели настолько схожи, что обладают одинаковыми спектрами свойств относительно поставленной задачи верификации

Вступление

Пример/наблюдение 2

Представим себе, что

- для системы $\mathfrak{S}$ была построена модель M,
- для этой модели было показано соответствие набору спецификаций $\varphi_1, \ldots, \varphi_k$,
- но после этого в модель были внесены изменения (исправления, улучшения, оптимизации), и в результате получена модель M'

Хотелось бы иметь возможность переносить результаты о правильности модели с M на M', избегая трудоёмкого процесса проверки соответствия модели спецификациям и ограничившись только анализом различий моделей M и M'

Но тогда требуется придумать отношение схожести между двумя моделями, гарантирующее одинаковое соответствие или несоответствие спецификациям $\varphi_1, \ldots, \varphi_k$ для схожих моделей

Трассовая эквивалентность моделей

Если в качестве языка спецификаций выбрать язык LTL, то нетрудно заметить (вспомнить), что:

- Каждая ltl-формула φ является формой записи множества «хороших» трасс $\mathrm{Tr}(\varphi)$
- Верификация модели M состоит в том, чтобы проверить, что все трассы модели являются «хорошими»: $\mathrm{Tr}(M) \subseteq \mathrm{Tr}(\varphi)$

Тогда можно схожесть моделей определить так

Модели M_1 и M_2 трассово эквивалентны, если $\mathrm{Tr}(M_1) = \mathrm{Tr}(M_2)$

Трассовая эквивалентность моделей

Утверждение. **Для любых трассово эквивалентных моделей M_1, M_2 и любой ltl-формулы φ верно:**

$$M_1 \models \varphi \quad \Leftrightarrow \quad M_2 \models \varphi$$

Но иметь одну только трассовую эквивалентность в качестве меры схожести моделей недостаточно:

- Аналогичное утверждение для многих других языков спецификаций оказывается неверным
  - Например, для CTL
- Проверка трассовой эквивалентности конечных моделей Крипке — это очень трудная задача
  - Более точно — PSPACE-полная, но не хочется тратить время на разъяснение этого факта

Необходимо иметь и такие определения схожести моделей, которые были бы более «простыми» и подходили для других языков спецификаций

Отношение симуляции

Рассмотрим две модели Крипке над общим множеством атомарных высказываний AP (в этом блоке множество AP считается всегда заданным по умолчанию):

$$M' = (S', S'_0, \to, L'), \quad M'' = (S'', S''_0, \mapsto, L'')$$

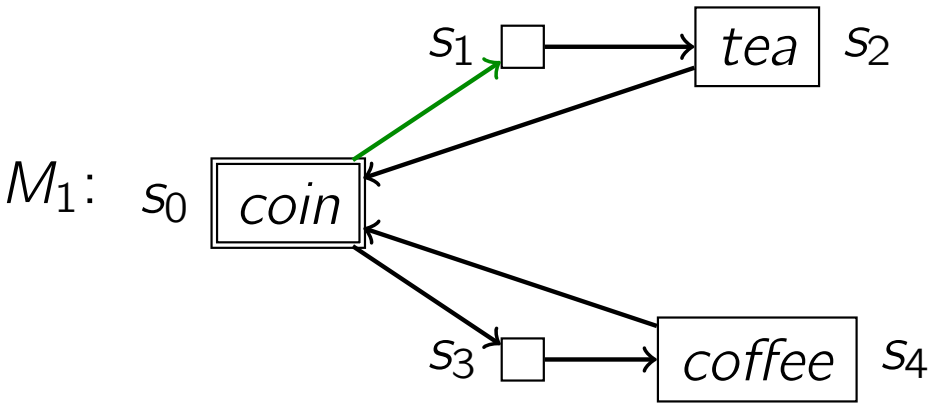
Отношение $\mathcal{R} \subseteq S' \times S''$ называется отношением симуляции между M' и M'', если для него выполнены два условия:

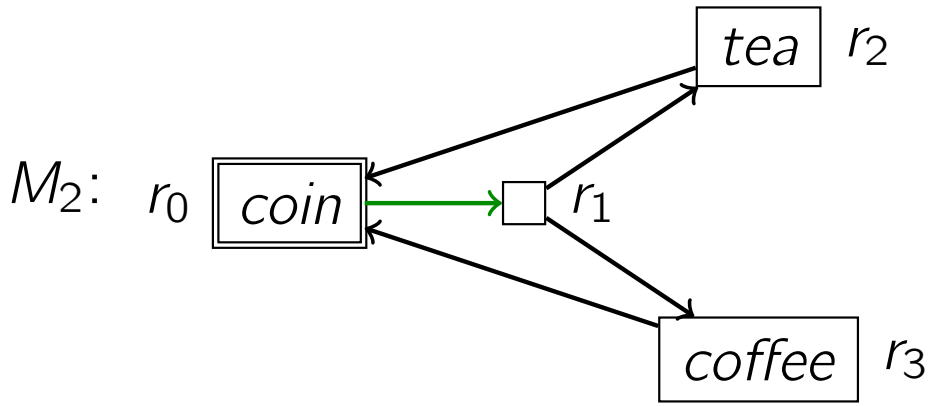
1. Для любого состояния s'_0 из S'_0 существует состояние s''_0 из S''_0, такое что $(s'_0, s''_0) \in \mathcal{R}$
2. Для любой пары s', s'', такой что $R(s', s'')$, верно следующее:
   - 2.1 $L(s') = L(s'')$
   - 2.2 Для любого состояния r', такого что $s' \to r'$, существует состояние r'', такое что $s'' \mapsto r''$ и $R(r', r'')$

Будем говорить, что модель M' симулируется моделью M'' ($M' \preceq M''$), если существует отношение симуляции между M' и M''

Отношение симуляции

Пример





Пример отношения симуляции между M_1 и M_2:

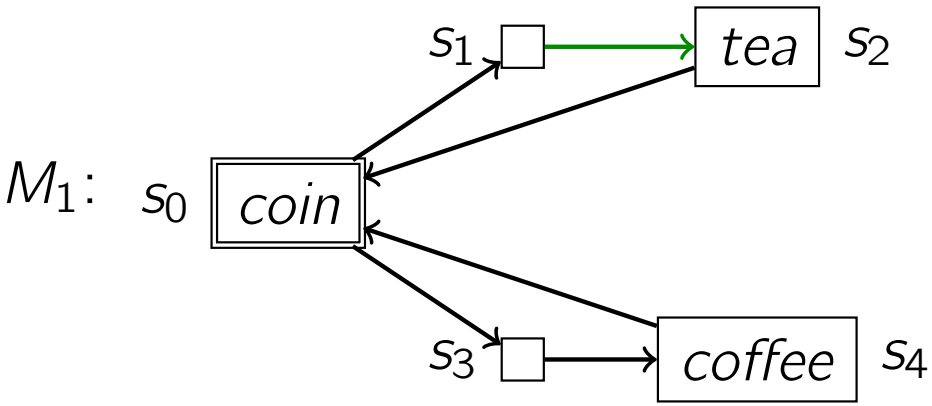
$$\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_1), (s_4, r_3)\}$$

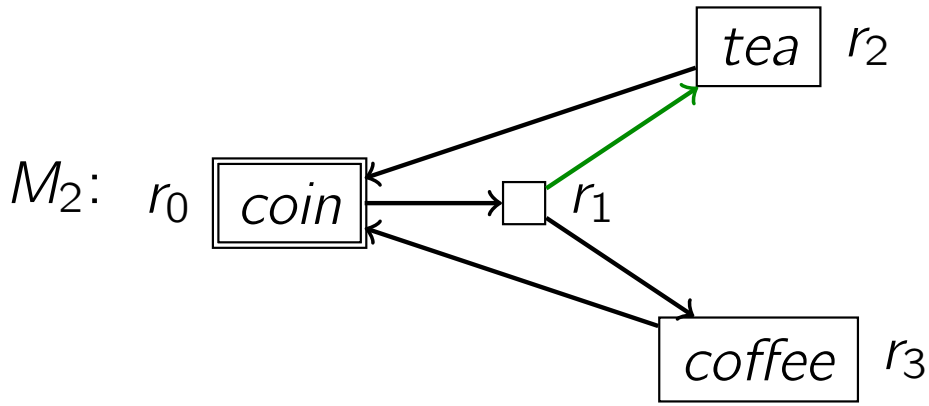
Это несложно проверить:

- Для s_0 в отношении существует пара
- Состояния каждой пары одинаково размечены атомарными высказываниями
- Каждой паре и каждому переходу в первой модели отвечает подходящий переход из парного состояния во второй модели (такой что пара концов переходов входит в отношение)

Отношение симуляции

Пример





Пример отношения симуляции между M_1 и M_2:

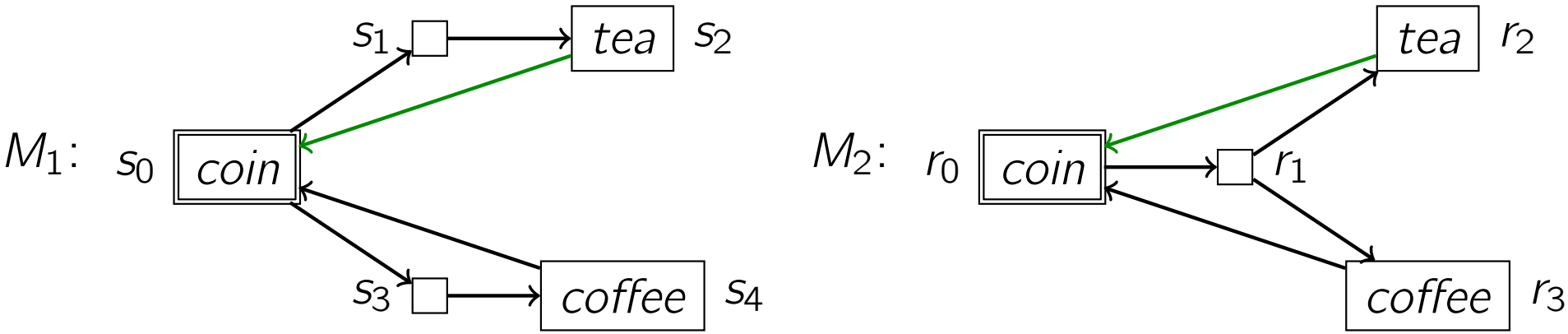
$$\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_1), (s_4, r_3)\}$$

Это несложно проверить:

- Для s_0 в отношении существует пара
- Состояния каждой пары одинаково размечены атомарными высказываниями
- Каждой паре и каждому переходу в первой модели отвечает подходящий переход из парного состояния во второй модели (такой что пара концов переходов входит в отношение)

Отношение симуляции

Пример



Пример отношения симуляции между M_1 и M_2:

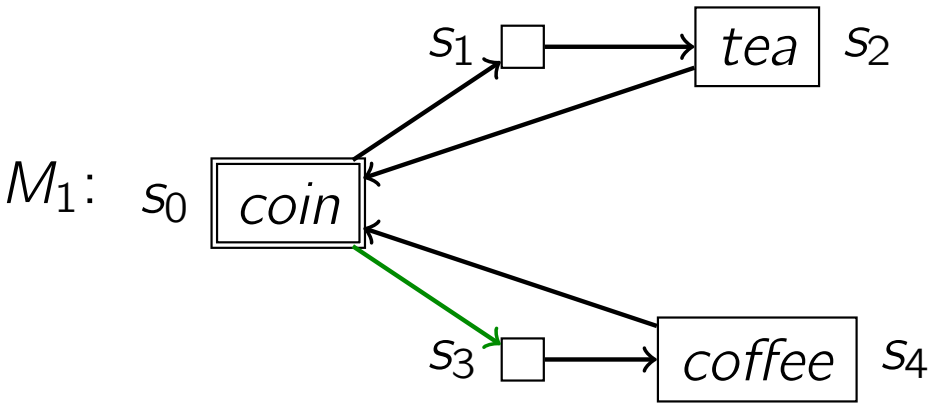
$$\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_1), (s_4, r_3)\}$$

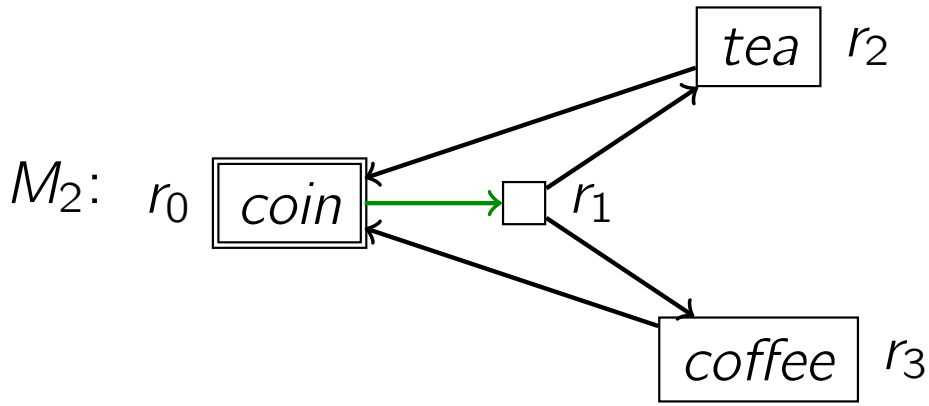
Это несложно проверить:

- Для s_0 в отношении существует пара
- Состояния каждой пары одинаково размечены атомарными высказываниями
- Каждой паре и каждому переходу в первой модели отвечает подходящий переход из парного состояния во второй модели (такой что пара концов переходов входит в отношение)

Отношение симуляции

Пример





Пример отношения симуляции между M_1 и M_2:

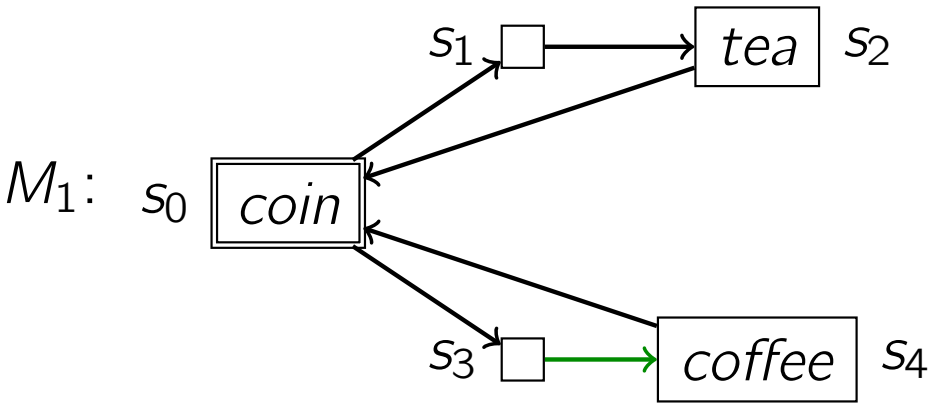
$$\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_1), (s_4, r_3)\}$$

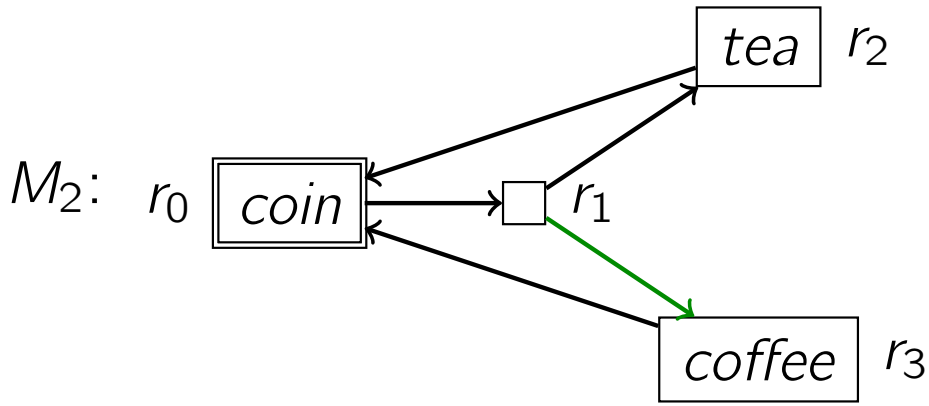
Это несложно проверить:

- Для s_0 в отношении существует пара
- Состояния каждой пары одинаково размечены атомарными высказываниями
- Каждой паре и каждому переходу в первой модели отвечает подходящий переход из парного состояния во второй модели (такой что пара концов переходов входит в отношение)

Отношение симуляции

Пример





Пример отношения симуляции между M_1 и M_2:

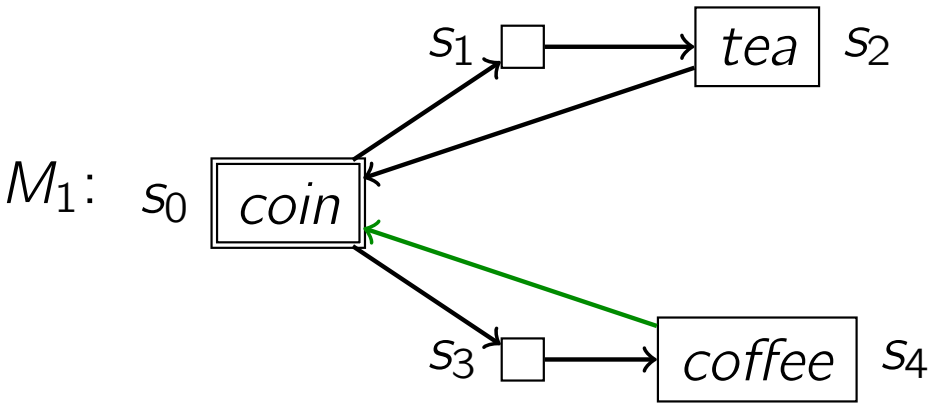
$$\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_1), (s_4, r_3)\}$$

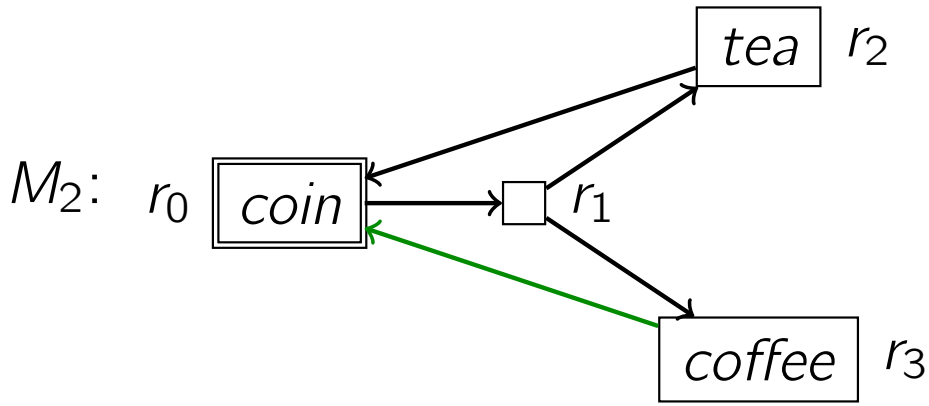
Это несложно проверить:

- Для s_0 в отношении существует пара
- Состояния каждой пары одинаково размечены атомарными высказываниями
- Каждой паре и каждому переходу в первой модели отвечает подходящий переход из парного состояния во второй модели (такой что пара концов переходов входит в отношение)

Отношение симуляции

Пример





Пример отношения симуляции между M_1 и M_2:

$$\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_1), (s_4, r_3)\}$$

Это несложно проверить:

- Для s_0 в отношении существует пара
- Состояния каждой пары одинаково размечены атомарными высказываниями
- Каждой паре и каждому переходу в первой модели отвечает подходящий переход из парного состояния во второй модели (такой что пара концов переходов входит в отношение)

Отношение симуляции

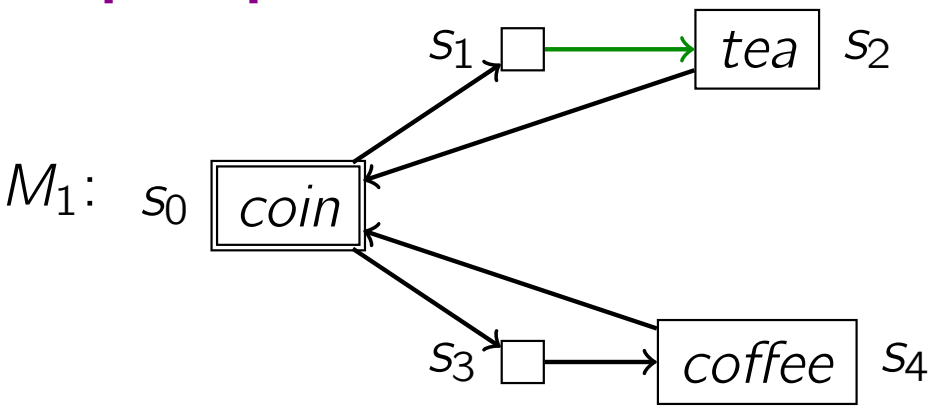
Симуляцию можно представить себе как *игру* между моделями M_1, M_2:

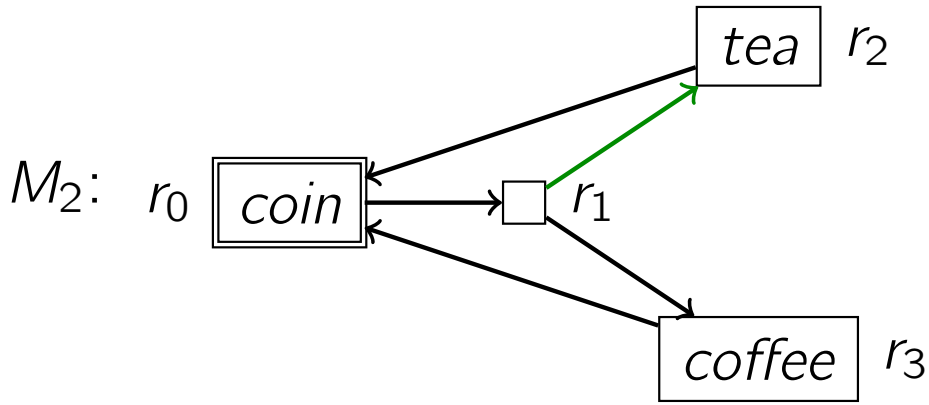
- M_1 выбирает своё начальное состояние
- M_2 выбирает начальное состояние с такой же разметкой
- Затем пошагово происходит следующее:
  - Каждая модель M_i находится в некотором состоянии s_i (на первом шаге — в выбранных начальных состояниях)
  - M_1 выбирает переход из s_1 и выполняет его, изменяя своё текущее состояние
  - M_2 выбирает переход из s_2 так, чтобы перейти в состояние с такой же разметкой
- M_2 проигрывает партию игры, если не может выбрать подходящее начальное состояние или подходящий очередной переход, и выигрывает, если партия бесконечна

Тогда $M_1 \preceq M_2$ означает, что существует *выигрышная стратегия* M_2, то есть способ выбора начального состояния и переходов в ответ на любые действия M_1 так, чтобы выиграть

Отношение симуляции

Например:





Тогда

- пары переходов, которые ранее подсвечивались зелёным для обоснования того, что отношение
$$\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_1), (s_4, r_3)\},$$
является отношением симуляции между M_1 и M_2, и
- пары состояний, входящие в это отношение,

представляют собой соответственно

- выигрышный способ ответного выбора переходов для M_2 и
- всевозможные расклады партии, возникающие при таком способе игры

Отношение симуляции

Утверждение. **Отношение $\preceq$ рефлексивно и транзитивно**

Доказательство.

Рефлексивность ($M \preceq M$) подтверждается отношением $\{(s, s) \mid s \in S\}$, где S — множество состояний модели M

Транзитивность: если $M_1 \preceq M_2$ и $M_2 \preceq M_3$, то $M_1 \preceq M_3$

Пусть $M_i = (S^i, S^i_0, \rightarrow^i, L^i)$

Рассмотрим отношения $\mathcal{R}_1$ и $\mathcal{R}_2$ симуляции для (M_1, M_2) и для (M_2, M_3) соответственно

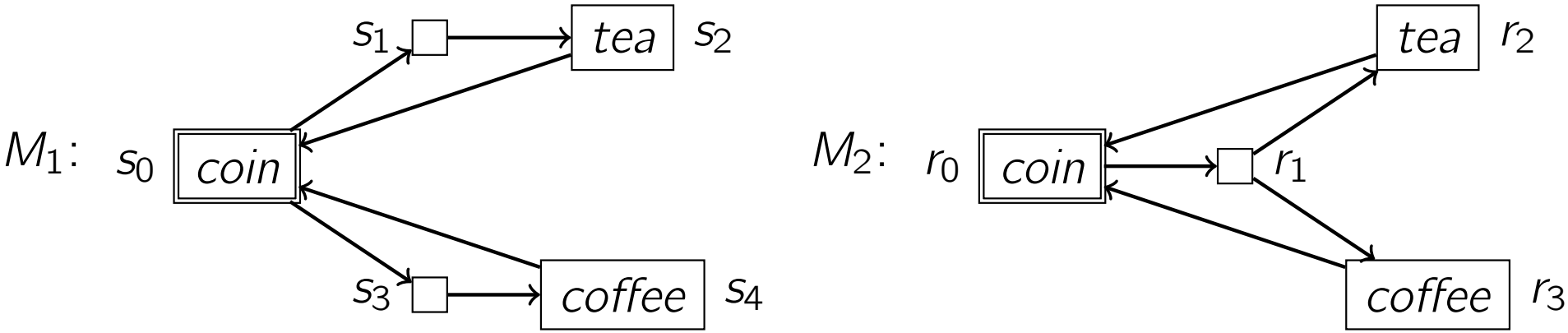
Тогда отношение симуляции $\mathcal{R}$ для M_1 и M_3 может быть устроено так: $\mathcal{R} = \{(s_1, s_3) \mid \exists s_2 : (s_1, s_2) \in \mathcal{R}_1,\ (s_2, s_3) \in \mathcal{R}_2\}$

Несложно убедиться, что для этого отношения выполнены все пункты определения симуляции ▼

Симуляционная эквивалентность

Модели M_1 и M_2 симуляционно эквивалентны ($M_1 \sim_s M_2$), если $M_1 \preceq M_2$ и $M_2 \preceq M_1$

Пример

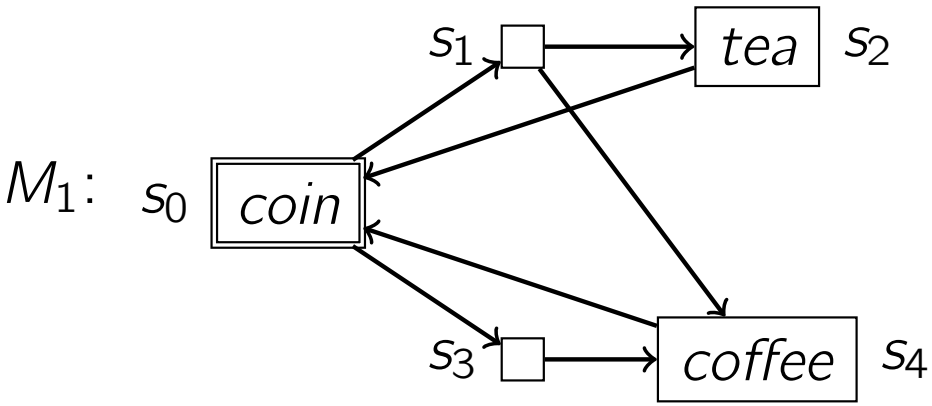


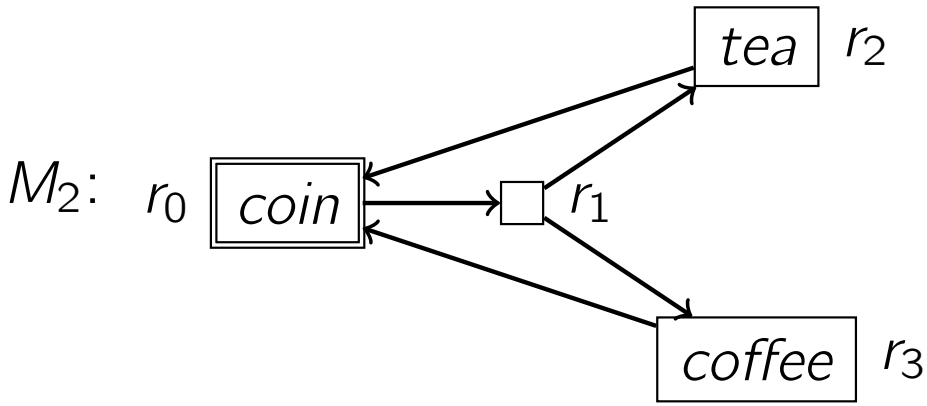
Эти модели не являются симуляционно эквивалентными, т.к. $M_2 \not\preceq M_1$:

- В отношение симуляции $\mathcal{R}$, если оно существует, должна входить хотя бы одна пара вида $(r_1, \_)$
- Согласно функции разметки, для $\mathcal{R}$ могут подойти только пары (r_1, s_1) и (r_2, s_2)
- Ни одна из этих пар не подходит:
  - (r_1, s_1) — т.к. из s_1 не исходит переход, отвечающий $r_1 \to r_3$
  - (r_1, s_3) — т.к. из s_1 не исходит переход, отвечающий $r_1 \to r_2$

Симуляционная эквивалентность

Другой пример





Эти модели симуляционно эквивалентны, что подтверждается такими двумя отношениями симуляции:

- $M_1 \preceq M_2$: $\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_1), (s_4, r_3)\}$
- $M_2 \preceq M_1$: $\{(r_0, s_0), (r_1, s_1), (r_2, s_2), (r_4, s_4)\}$

Утверждение. $\sim_s$ **— отношение эквивалентности**

Доказательство.

Рефлексивность и транзитивность следуют из рефлексивности и транзитивности отношения $\preceq$

Симметричность очевидна: $M_1 \sim_s M_2 \Leftrightarrow M_1 \preceq M_2$ и $M_2 \preceq M_1 \Leftrightarrow M_2 \preceq M_1$ и $M_1 \preceq M_2 \Leftrightarrow M_2 \sim M_1$ ▼

Свойства симуляционной эквивалентности

Будем говорить, что пути π_1, π_2 в моделях Крипке $M_1 = (S^1, S_0^1, \to^1, L^1)$, $M_2 = (S^2, S_0^2, \to^2, L^2)$ $\mathcal{R}$-соответствуют друг другу для отношения $\mathcal{R} \subseteq S^1 \times S^2$, если для любого момента времени k верно $(\pi_1[k], \pi_2[k]) \in \mathcal{R}$

Утверждение. **Для любых**

- **моделей Крипке** $M_1 = (S^1, S_0^1, \to^1, L^1)$, $M_2 = (S^2, S_0^2, \to^2, L^2)$,
- **отношения симуляции** $\mathcal{R}$ **для** M_1 **и** M_2,
- **состояний** $s_1 \in S^1$ **и** $s_2 \in S^2$, **таких что** $(s_1, s_2) \in \mathcal{R}$, **и**
- **пути** π_1 **в** M_1, **исходящего из** s_1

существует $\mathcal{R}$**-соответствующий путь** π_2 **в** M_2, **исходящий из** s_2

Это утверждение несложно доказывается индукцией по длине пути

Следствие. **Если** $M_1 \preceq M_2$, **то** $\mathrm{Tr}(M_1) \subseteq \mathrm{Tr}(M_2)$

Следствие. **Если** $M_1 \sim_s M_2$, **то модели** M_1 **и** M_2 **трассово эквивалентны**

Свойства симуляционной эквивалентности

Actl∗-формулой будем называть ctl*-формулу частного вида, подходящего под БНФ

$$\Phi \quad ::= \quad \mathbb{t} \mid p \mid \neg p \mid (\Phi \,\&\, \Phi) \mid (\Phi \vee \Phi) \mid (\mathbf{A}\varphi),$$
$$\varphi \quad ::= \quad \Phi \mid \varphi \,\&\, \varphi \mid \varphi \vee \varphi \mid (\mathbf{X}\varphi) \mid (\varphi \mathbf{U} \varphi) \mid (\varphi \mathbf{R} \psi),$$

где

- $p \in \mathrm{AP}$ и
- $\varphi \mathbf{R} \psi = \neg(\neg\varphi \mathbf{U} \neg\psi)$

Теорема. **Для любых моделей M_1, M_2, таких что $M_1 \preceq M_2$, и любой actl*-формулы φ, верно:** **если $M_1 \models \varphi$, то $M_2 \models \varphi$**

Можете попробовать сами доказать это индукцией по структуре формулы с использованием последнего утверждения о соответствующих путях (это не очень трудно)

Следствие. **Для любых моделей M_1, M_2, таких что $M_1 \sim_s M_2$, и любой actl*-формулы φ верно:** $M_1 \models \varphi \quad \Leftrightarrow \quad M_2 \models \varphi$

Хотя ACTL* — это достаточно широкий фрагмент CTL* (*шире LTL*), но хотелось бы иметь аналогичное утверждение и про весь язык CTL*

Бисимуляция (отношение, эквивалентность)

Отношение $\mathcal{R}^- \subseteq Y \times X$ называется обратным к $\mathcal{R} \subseteq X \times Y$, если $R^- = \{(y, x) \mid (x, y) \in R\}$

Рассмотрим модели Крипке $M' = (S', S'_0, \to, L')$ и $M'' = (S'', S''_0, \mapsto, L'')$

Отношение $\mathcal{R} \subseteq S' \times S''$ называется отношением бисимуляции между M' и M'', если $\mathcal{R}$ и $\mathcal{R}^-$ — отношения симуляции между M' и M'' и между M'' и M' соответственно

Модели M_1 и M_2 бисимуляционно эквивалентны ($M_1 \sim_b M_2$), если существует отношение бисимуляции между этими моделями

Пример

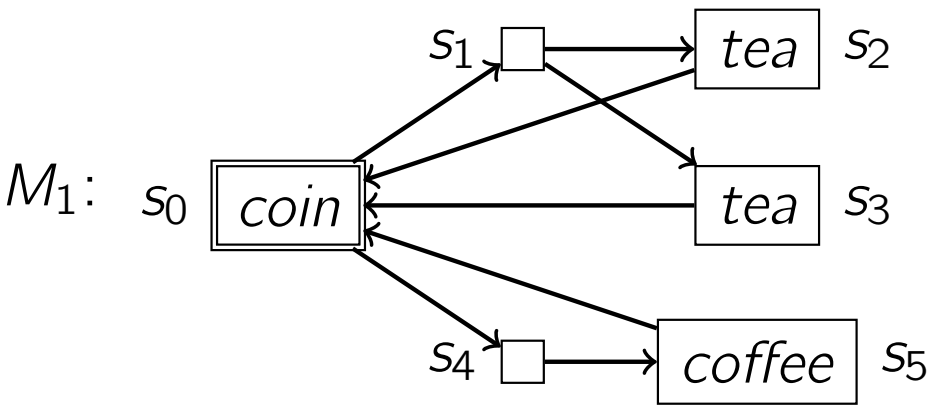
Эти две модели бисимуляционно эквивалентны, что подтверждается таким отношением бисимуляции:

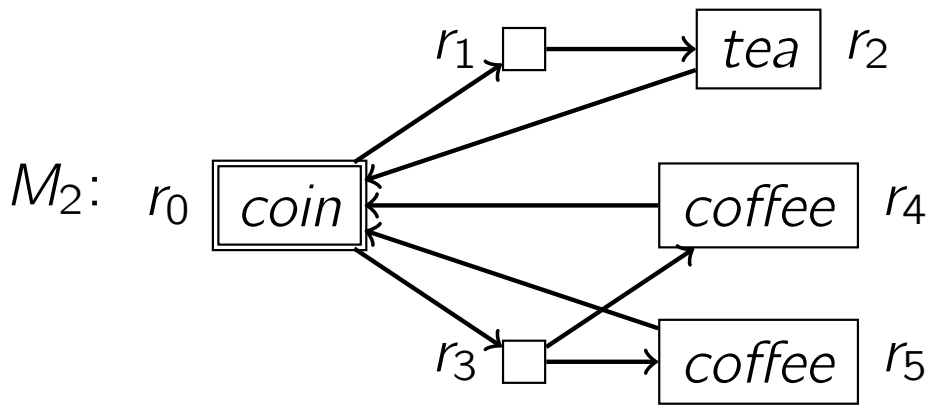
$$\{(s_0, r_0), (s_0, r_2), (s_1, r_1), (s_1, r_3)\}$$

Бисимуляционная эквивалентность похожа на изоморфизм графов, но допускает, в числе прочего, «частичную развёртку» модели

Бисимуляция (отношение, эквивалентность)

Другой пример





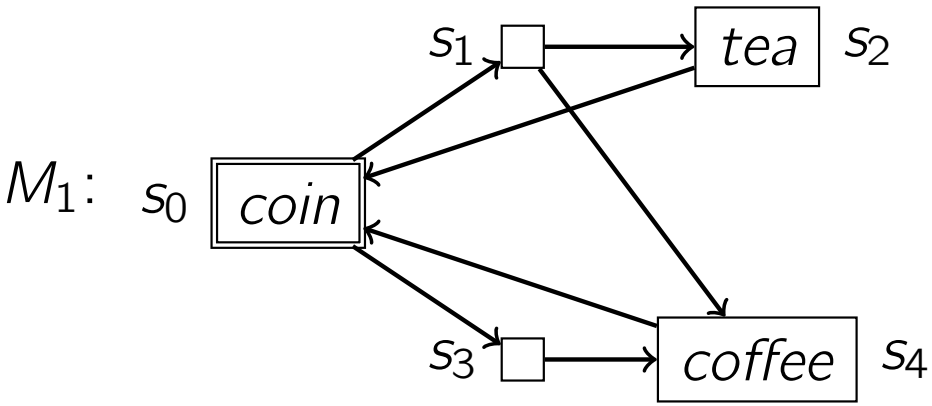
Эти модели бисимуляционно эквивалентны, что подтверждается отношением бисимуляции

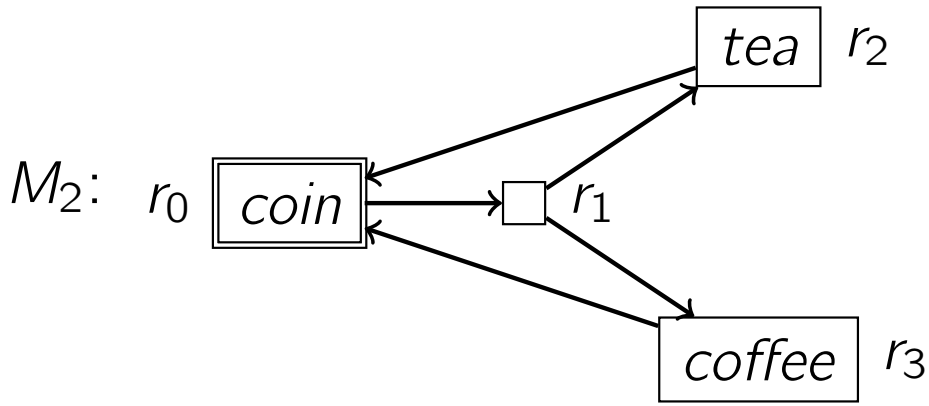
$$\{(s_0, r_0), (s_1, r_1), (s_2, r_2), (s_3, r_2), (s_4, r_3), (s_5, r_4), (s_5, r_5)\}$$

Бисимуляция допускает и «дублирование» состояний в модели

Бисимуляция (отношение, эквивалентность)

И ещё один пример





Ранее было показано, что эти модели симуляционно эквивалентны

Но они не являются бисимуляционно эквивалентными, так как если существует отношение бисимуляции $\mathcal{R}$ между M_1 и M_2, то:

- Так как $\mathcal{R}$ — отношение симуляции между M_1 и M_2, то в нём должна содержаться хотя бы одна пара вида $(s_3, \_)$
- Согласно функции разметки, такой парой может быть только (s_3, r_1)
- Так как $\mathcal{R}^-$ — отношение симуляции между M_2 и M_1 и переходу $r_1 \to r_2$ не соответствует ни один переход из s_3, то пара (s_3, r_1) не может содержаться в $\mathcal{R}$ (*противоречие*)

Бисимуляционная эквивалентность и её свойства

Утверждение. $\sim_b$ **— отношение эквивалентности**

Доказательство. Рефлексивность и транзитивность следуют из того, что

- отношение бисимуляции между двумя моделями является и отношением симуляции между этими моделями
- отношение $\preceq$ рефлексивно и транзитивно

Симметричность очевидна: если $\mathcal{R}$ — отношение бисимуляции между M_1 и M_2, то $\mathcal{R}^-$ — отношение бисимуляции между M_2 и M_1 ▼

Утверждение. Любые две бисимуляционно эквивалентные модели симуляционно эквивалентны

Доказательство. Достаточно заметить, что (по определению) если $\mathcal{R}$ — отношение бисимуляции между M_1 и M_2, то $\mathcal{R}$ и $\mathcal{R}^-$ — отношения симуляции между M_1 и M_2 и между M_2 и M_1 соответственно ▼

Бисимуляционная эквивалентность и её свойства

Утверждение. **Для любых**

- **моделей Крипке** $M_1 = (S^1, S_0^1, \to^1, L^1)$, $M_2 = (S^2, S_0^2, \to^2, L^2)$,
- **отношения бисимуляции** $\mathcal{R}$ **для** M_1 **и** M_2,
- **состояний** $s_1 \in S^1$ **и** $s_2 \in S^2$, **таких что** $(s_1, s_2) \in \mathcal{R}$,
- **номера** i, $i \in \{1, 2\}$ **и**
- **пути** π_i **из** s_i **в** M_i

существует путь π_{3-i} **из** s_{3-i} **в** M_{3-i}, $\mathcal{R}$**-соответствующий** π_i

Это утверждение несложно доказывается индукцией по длине пути

Бисимуляционная эквивалентность и её свойства

Утверждение. **Для любых**

- **бисимуляционно эквивалентных моделей M_1, M_2,**
- **отношения бисимуляции $\mathcal{R}$ между этими моделями,**
- **состояний s_1, s_2 этих моделей, таких что $(s_1, s_2) \in \mathcal{R}$,**
- **$\mathcal{R}$-соответсвующих путей π_1, π_2 в этих моделях,**
- **формулы состояния Φ CTL* и**
- **формулы пути φ CTL***

верны равносильности

$$M_1, s_1 \models \Phi \quad \Leftrightarrow \quad M_2, s_2 \models \Phi$$
$$M_1, \pi_1 \models \varphi \quad \Leftrightarrow \quad M_2, \pi_2 \models \varphi$$

Можете попробовать сами доказать это индукцией по структуре формулы с привлечением предыдущего утверждения

Бисимуляционная эквивалентность и её свойства

Теорема. **Для любых бисимуляционно эквивалентных моделей M_1, M_2 и любой ctl*-формулы φ верно:**

$$M_1 \models \varphi \quad \Leftrightarrow \quad M_2 \models \varphi$$

Итого определено три отношения эквивалентности $\sim$ моделей Крипке с соответствующими фрагментами $\mathcal{L}$ языка CTL*:

1. Трассовая эквивалентность и LTL
2. Симуляционная эквивалентность и ACTL*
3. Бисимуляционная эквивалентность и CTL*

Если для спецификации модели M_1 используется фрагмент $\mathcal{L}$, то можно быть уверенным в том, что на любой модель M_2, такая что $M_1 \sim M_2$ для соответствующего $\sim$, настолько же удовлетворяет всем спецификациям, насколько и M_1

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 36

Бисимуляция состояний модели
Алгоритм проверки бисимуляционной эквивалентности
Фактор-модель

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Три отношения эквивалентности $\sim$ моделей Крипке с соответствующими фрагментами $\mathcal{L}$ языка CTL*:

1. Трассовая эквивалентность и LTL
2. Симуляционная эквивалентность и ACTL*
3. Бисимуляционная эквивалентность и CTL*

Если модель M_1 специфицирована в терминах фрагмента $\mathcal{L}$, то можно быть уверенным в том, что на любой модели M_2, такой что $M_1 \sim M_2$, выполняются в точности те же формулы, что и на M_1

А как проверить такую эквивалентность?

Про трассовую эквивалентность упоминалось, что её проверка — это трудная задача

Поэтому трассовую эквивалентность оставим в стороне

Симуляционная и бисимуляционная эквивалентности похожи, и алгоритмы проверки для них тоже похожи

Поэтому подробно рассмотрим только бисимуляционную эквивалентность

Бисимуляция состояний модели

$M(s)$ — так будем обозначать модель Крипке, получающуюся из модели M заменой множества начальных состояний на $\{s\}$

Состояния s_1 и s_2 модели Крипке M бисимуляционно эквивалентны ($s_1 \sim_b^M s_2$), если $M(s_1) \sim_b M(s_2)$

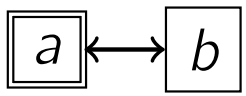
Проверку бисимуляционной эквивалентности конечных моделей M_1, M_2 можно свести к проверке бисимуляционной эквивалентности двух состояний одной конечной модели:

1. Добавим в каждую из моделей M_i одно новое состояние s_i с меткой $\emptyset$
2. Проведём из s_i дуги во все начальные состояния
3. Переименуем состояния M_1 и M_2 так, чтобы их множества состояний не пересекались
4. Объединим все состояния и переходы моделей в модель M с пустым множеством начальных состояний
5. $M_1 \sim_b M_2 \quad \Leftrightarrow \quad s_1' \sim_b^M s_2'$, где s_1' и s_2' — состояния, получающиеся из s_1 и s_2 после переименования

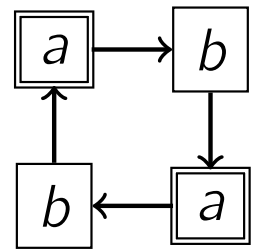
Бисимуляция состояний модели

Пример

Чтобы проверить бисимуляционную эквивалентность моделей

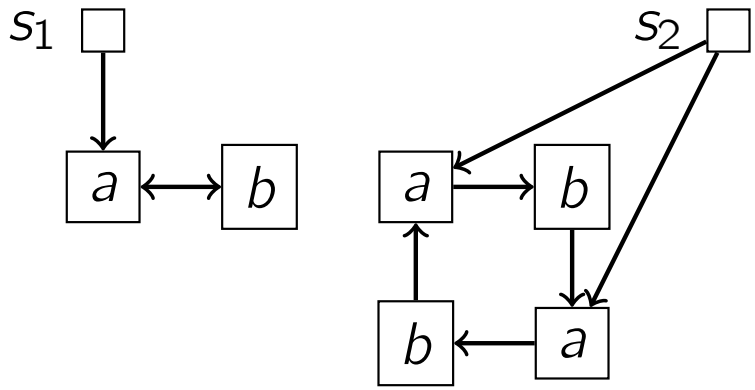


и



,

достаточно проверить бисимуляционную эквивалентность состояний s_1 и s_2 в модели



Бисимуляция состояний модели

Рассмотрим модель $M = (S, S_0, \to, L)$

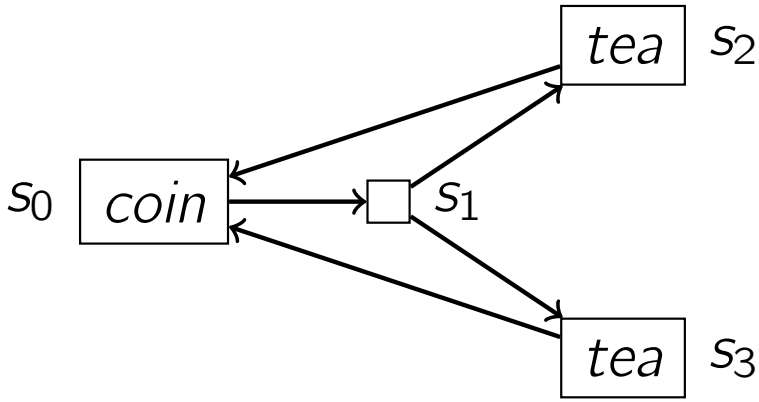
Отношение $\mathcal{R} \subseteq S \times S$ называется отношением бисимуляции на модели M, если для любой пары $(s, r) \in \mathcal{R}$ отношение $\mathcal{R}$ является отношением бисимуляции между $M(s)$ и $M(r)$

Это определение отличается от определения отношения бисимуляции для пары моделей только тем, что

- вместо двух моделей рассматривается одна, взятая два раза, и
- не требуется соответствие начальных состояний

Бисимуляция состояний модели

Пример



Примеры отношений бисимуляции на этой модели:

1. $\{(s_0, s_0), (s_1, s_1), (s_2, s_2), (s_3, s_3)\}$
2. $\{(s_0, s_0), (s_1, s_1), (s_2, s_2), (s_2, s_3), (s_3, s_2), (s_3, s_3)\}$
3. $\{(s_0, s_0), (s_1, s_1), (s_2, s_3), (s_3, s_2)\}$
4. $\emptyset$

Утверждение. **Если $\mathcal{R}_1$ и $\mathcal{R}_2$ — отношения бисимуляции на конечной модели Крипке M, то отношение $\mathcal{R}_1 \cup \mathcal{R}_2$ также является отношением бисимуляции на M**

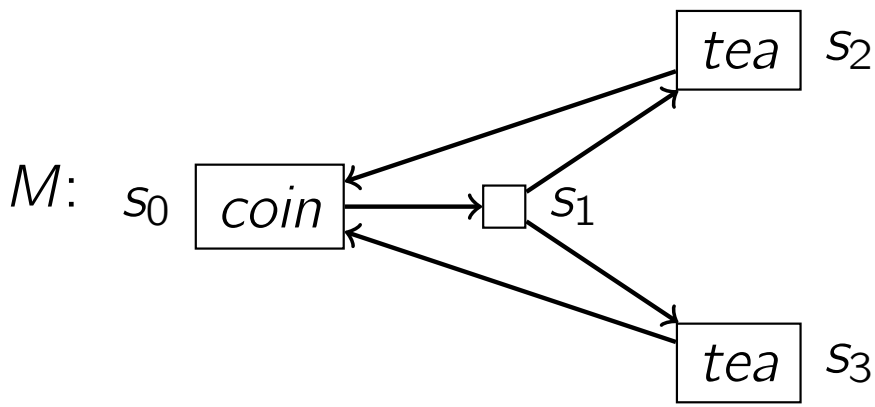
Можете попробовать доказать это самостоятельно (это не сверхсложно)

Бисимуляция состояний модели

Следствие. **Если $\mathcal{R}_1, \ldots, \mathcal{R}_n$ — все отношения бисимуляции на конечной модели Крипке M, то $\bigcup_{i=1}^{n} \mathcal{R}_i$ — отношение бисимуляции на M, наибольшее по теоретико-множественному включению**

$\approx_M$ — так будем обозначать отношение бисимуляции на конечной модели Крипке M, наибольшее по теоретико-множественному включению (*существующее согласно следствию выше, и очевидным образом единственное*)

Пример



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

Бисимуляция состояний модели

Утверждение. **Для любой конечной модели Крипке M отношение $\approx_M$ является отношением эквивалентности**

Утверждение. **Для любой конечной модели Крипке M отношения $\sim_b^M$ и $\approx_M$ совпадают**

И это можете попробовать доказать самостоятельно (и это не очень сложно)

Следовательно, проверку соотношения $s_1 \sim_b^M s_2$ можно устроить так:

1. Вычислить все классы эквивалентности отношения $\approx_M$
2. Проверить, лежат ли s_1 и s_2 в одном классе эквивалентности

Осталось показать, как можно вычислить все классы эквивалентности отношения $\approx_M$

$S/\approx$ — так для отношения эквивалентности $\approx$ на множестве S будем обозначать семейство всех классов эквивалентности отношения $\approx$

Вычисление классов эквивалентности $\approx_M$

Дано: конечная модель Крипке $M = (S, S_0, \to, L)$

Требуется: вычислить семейство $S/{\approx_M}$

Общая идея алгоритма похожа на идею алгоритма минимизации детерминированного конечного автомата и на вычисление наибольшей неподвижной точки преобразователя предикатов:

- На каждом шаге имеем некоторое разбиение множества S на предполагаемые классы эквивалентности
- Если можно «тривиально» заключить, что некоторые два состояния, предполагающиеся эквивалентными, на самом деле неэквивалентны, то соответственно разобъём предполагаемый класс эквивалентности на два
- Иначе полученное семейство множеств состояний выдаётся в ответ
- Начнём со «слабого» предположения: разбиения, из которого можно получить ответ такими подразбиениями классов

Вычисление классов эквивалентности $\approx_M$

Разбиением множества S будем называть любое конечное семейство $\{B_1, \ldots, B_n\}$ попарно непересекающихся предикатов, такое что $S = \bigcup_{i=1}^{n} B_i$

$Post_M(s)$ — так будем обозначать множество всех состояний s', таких что в M содержится переход $s \to s'$

Предикат C назовём разветвителем предиката B, если существует пара состояний $s_1, s_2 \in B$, такая что

- $Post_M(s_1) \cap C \neq \emptyset$ (*из s_1 можно перейти в C*) и
- $Post_M(s_2) \cap C = \emptyset$ (*из s_2 нельзя перейти в C*)

Содержательно, существование разветвителя — это «тривиальный» индикатор того, что состояния неэквивалентны

Вычисление классов эквивалентности $\approx_M$

Уточнением предиката B относительно предиката C будем называть семейство предикатов $Ref(B|C)$, устроенное так:

- Если C — разветвитель предиката B, то $Ref(B|C) = \{D, E\}$, где
  - $D = \{s \mid s \in B,\ Post_M(s) \cap C \neq \emptyset\}$
  - $E = \{s \mid s \in B,\ Post_M(s) \cap C = \emptyset\}$
- Иначе $Ref(B|C) = \{B\}$

Уточнением семейства предикатов $\mathfrak{B} = \{B_1, \ldots, B_n\}$ относительно предиката C будем называть семейство предикатов

$$Ref(\mathfrak{B}|C) = \bigcup_{i=1}^{n} Ref(B_i|C)$$

Уточнением разбиения $\mathfrak{B} = \{B_1, \ldots, B_n\}$ будем называть разбиение

$$Ref(\mathfrak{B}) = Ref(\ldots Ref(Ref(\mathfrak{B}|B_1)|B_2)\ldots|B_n)$$

Вычисление классов эквивалентности $\approx_M$

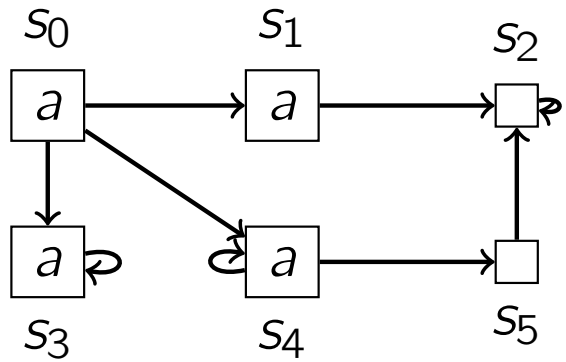
Алгоритм $\mathfrak{A}$ вычисления классов эквивалентности $\approx_M$ для $M = (S, S_0, \to, L)$ над AP:

1. Вычислить разбиение
   $\mathfrak{B}_0 = \{B_X | B_X = \{s \mid s \in S,\ L(s) = X\},\ B_X \neq \emptyset, X \subseteq \mathrm{AP}\}$
   - *То есть эквивалентными полагаются состояния с одинаковой разметкой атомарными высказываниями*

2. Последовательно для каждого $i \in \{1, 2, \dots\}$:
   2.1 Вычислить $\mathfrak{B}_i = \mathit{Ref}(\mathfrak{B}_{i-1})$
   2.2 Если $\mathfrak{B}_i = \mathfrak{B}_{i-1}$, то завершить алгоритм и выдать ответ $\mathfrak{B}_i$

Вычисление классов эквивалентности $\approx_M$

Пример



$$\mathfrak{B}_0 = [\{s_0, s_1, s_3, s_4\}, \{s_2, s_5\}]$$

$$\begin{aligned}
\mathfrak{B}_1 &= Ref([\{s_0, s_1, s_3, s_4\}, \{s_2, s_5\}]) \\
&= Ref(Ref([\{s_0, s_1, s_3, s_4\}, \{s_2, s_5\}]|\{s_0, s_1, s_3, s_4\})|\{s_2, s_5\}) \\
&= Ref([\{s_0, s_3, s_4\}, \{s_1\}, \{s_2, s_5\}]|\{s_2, s_5\}) \\
&= [\{s_0, s_3\}, \{s_4\}, \{s_1\}, \{s_2, s_5\}] \\
\mathfrak{B}_2 &= Ref([\{s_0, s_3\}, \{s_4\}, \{s_1\}, \{s_2, s_5\}]) \\
&= \dots \\
&= [\{s_0\}, \{s_3\}, \{s_4\}, \{s_1\}, \{s_2, s_5\}]
\end{aligned}$$

$$\mathfrak{B}_3 = Ref(\mathfrak{B}_2) = \dots = \mathfrak{B}_2$$

Ответ: $[\{s_0\}, \{s_3\}, \{s_4\}, \{s_1\}, \{s_2, s_5\}]$

Вычисление классов эквивалентности $\approx_M$

Уточнением отношения эквивалентности $\mathcal{R}$ на множестве S назовём отношение эквивалентности $Ref(\mathcal{R})$, такое что $S/Ref(\mathcal{R}) = Ref(S/\mathcal{R})$

Утверждение. **Для любой конечной модели Крипке M и любого отношения эквивалентности $\mathcal{R}$ на состояниях этой модели верно:**

1. $\mathcal{R}$ **— отношение бисимуляции** $\quad\Leftrightarrow\quad Ref(\mathcal{R}) = \mathcal{R}$
2. $Ref(\mathcal{R}) \subseteq \mathcal{R}$
3. **Если** $\approx_M \subseteq \mathcal{R}$**, то** $\approx_M \subseteq Ref(\mathcal{R})$

Теорема. **Для любой конечной модели Крипке $M = (S, S_0, \to, L)$ алгоритм $\mathfrak{A}$ завершается и выдаёт в ответ семейство $S/\approx_M$**

Можете доказать утверждение и теорему самостоятельно

Для самостоятельного размышления:

1. Какова сложность предложенного алгоритма по времени работы?
2. Можно ли предложить алгоритм с меньшим порядком сложности?
3. Попробуйте предложить (с обоснованием) аналогичный алгоритм для симуляционной эквивалентности

Фактор-модель

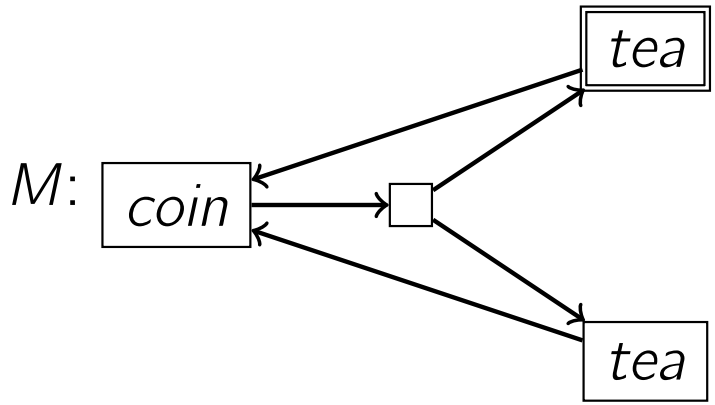
$[s]_{\approx}$ — так будем обозначать класс эквивалентности элемента s по отношению $\approx$

Фактор-моделью модели Крипке $M = (S, S_0, \to, L)$ называется модель $M_{\approx} = (S', S'_0, \mapsto, L')$, устроенная так:

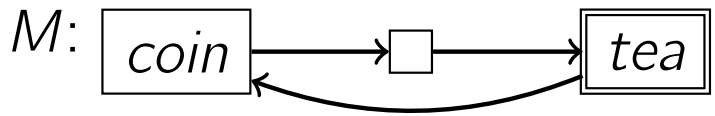
- $S' = S/{\approx_M}$
- $S'_0 = \{[s]_{\approx_M} \mid s \in S_0\}$
- $\mapsto = \{([s]_{\approx_M}, [r]_{\approx_M}) \mid s \to r\}$
- $L'([s]_{\approx_M}) = L(s)$

Фактор-модель

Пример



Фактор-модель $M_{\approx}$ устроена так:



Утверждение. **Для любой конечной модели Крипке** M **верно** $M \sim_b M_{\approx}$

И это тоже можете попробовать доказать самостоятельно

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 37

Абстракция моделей
Редукция по конусу влияния
Абстракция данных

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Абстракция моделей

Абстракция модели — это устранение из неё деталей, которые по тем или иным причинам считаются излишними (избыточными, несущественными) и не влияют на результаты анализа интересующих свойств

Абстракция — это самый действенный способ решения проблемы комбинаторного взрыва числа состояний

Для примера рассмотрим два метода абстракции:

1. Редукция по конусу влияния
2. Абстракция данных

Эти методы применяются к описанию модели, более «высокоуровневому» по сравнению с моделью Крипке, и потому позволяют избежать построения чрезмерно большой модели Крипке, упрощая модель до начала её построения

Абстракция моделей

Редукция по конусу влияния применяется к системам, состояния которых задаются как наборы значений заданных переменных

Основная идея метода состоит в том, чтобы устранить из системы те переменные, которые не содержатся в спецификации и не оказывают на спецификацию никакого *влияния*

То есть удаляются переменные, которые с точки зрения спецификации являются *фиктивными*

Абстракция данных состоит в отображении «реальных» значений переменных (которых может быть очень много) в небольшое число *абстрактных* значений данных с сохранением интересующих свойств исходной системы

Например, если абстрактная система симулирует исходную, то можно гарантировать сохранение выполнимости спецификаций из достаточно широкого фрагмента языка ACTL*

Редукция по конусу влияния

Рассмотрим систему, состояния которой определяются как всевозможные наборы значений переменных множества $V = \{v_1, \ldots, v_n\}$, а переходы определяются согласно рассказанному для символьных представлений относительно двух комплектов переменных и задаются уравнениями вида

$$v_i' = f_i(v_1, \ldots, v_n)$$

Пусть в спецификации системы используются только переменные множества W, $W \subseteq V$

Для упрощения системы можно было бы попробовать удалить из неё все переменные, которые не входят в W

Но увы, так сделать «в лоб» нельзя: значения переменных из W могут прямо или косвенно зависеть от значений переменных из $W \setminus V$

Редукция по конусу влияния

Системой при обсуждении конусов влияния будем называть четвёрку $\mathfrak{S} = (V, X, \vec{f}, W)$ такого вида:

- $V = (v_1, \ldots, v_n)$ — конечный набор булевых переменных, $n \geq 1$
- $X \subseteq \{0,1\}^n$ — множество начальных оценок переменных
- $\vec{f} = (f_1, \ldots, f_n)$ — набор n-местных булевых функций, задающих уравнения системы
- W — множество наблюдаемых переменных, $W \subseteq V$

Для такой системы $\mathfrak{S}$ модель Крипке $M_{\mathfrak{S}} = (S, S_0, \to, L)$ устроим так:

- $S = \{0,1\}^n$
- $S_0 = X$
- $\to$ — множество переходов, символьное представление которого имеет вид $\&_{i=1}^{n} (v_i' \leftrightarrow f_i(v_1, \ldots, v_n))$
- $L(s) = \{v_i \mid s[i] = 1, v_i \in W\}$

Редукция по конусу влияния

Конусом влияния $\mathcal{C}_{\mathfrak{S}}(W)$ множества переменных W в системе $\mathfrak{S}$ будем называть наименьшее множество переменных, для которого верно следующее:

- $W \subseteq \mathcal{C}_{\mathfrak{S}}(W)$
- Если для переменной v_i из $\mathcal{C}_{\mathfrak{S}}(W)$ функция f_i системы $\mathfrak{S}$ существенно зависит от переменной v_m, то $v_m \in \mathcal{C}_{\mathfrak{S}}(W)$

Метод редукции по конусу влияния состоит в удалении всех переменных системы, не входящих в конус влияния переменных, значения которых используются в спецификации системы

Редукция по конусу влияния

Пример (счётчик по модулю 8)

Рассмотрим систему над переменными $V = (v_0, v_1, v_2)$, принимающими значения $\{0, 1\}$ и задаваемую системой

$$\begin{cases} v_0' &= \neg v_0 \\ v_1' &= v_0 \oplus v_1 \\ v_2' &= (v_0 \,\&\, v_1) \oplus v_2 \end{cases}$$

Для множества $\{v_0\}$ конусом влияния является $\{v_0\}$

Для множеств $\{v_1\}$ и $\{v_0, v_1\}$ конусом влияния является $\{v_0, v_1\}$

Для множеств переменных, содержащих v_2, конус влияния — это множество всех переменных

Таким образом, если в спецификации существенным является только значение переменной v_0, то систему можно упростить, оставив только первое уравнение, и если в спецификации не используется значение v_2, то из системы можно удалить третье уравнение

Редукция по конусу влияния

Записью $M|_W$ для модели Крипке $M = (S, S_0, \to, L)$ и подмножества W атомарных высказываний обозначим модель, получающуюся из M заменой каждой метки $L(s)$ на $L(s) \cap W$

Утверждение. Для любой модели Крипке M и любой ctl*-формулы φ над атомарными высказываниями W верна равносильность

$$M \models \varphi \quad \Leftrightarrow \quad M|_W \models \varphi$$

Систему $\mathcal{R}(\mathfrak{S}, W)$, получающуюся из $\mathfrak{S} = (V, X, \vec{f}, U)$ редукцией по множеству переменных W, зададим как четвёрку $(C, Y, \vec{g}, U \cap W)$, где:

- C — множество $C_{\mathfrak{S}}(W)$ с таким же порядком переменных, как в V
- Y получается из X удалением разрядов всех наборов, отвечающих расположению переменных из $V \setminus C_{\mathfrak{S}}(W)$ в V
- $\vec{g}$ — функции из $\vec{f}$, отвечающие переменным из C

Редукция по конусу влияния

Утверждение. **Для любой системы $\mathfrak{S}$ и любого подмножества переменных W верно соотношение**

$$M_{\mathfrak{S}}|_W \sim_b M_{\mathcal{R}(\mathfrak{S},W)}$$

Можете доказать это утверждение самостоятельно, основываясь на отношении бисимуляции, состоящем из всех пар $(x, y) \in \{0,1\}^n \times \{0,1\}^{|C_{\mathfrak{S}}(W)|}$, в которых y получается из x вычёркиванием разрядов, отвечающих переменным не из конуса влияния

Следствие. **Для любых системы $\mathfrak{S}$, подмножества её переменных W и ctl*-формулы φ над атомарными высказываниями W верно**

$$M_{\mathfrak{S}}|_W \models \varphi \quad \Leftrightarrow \quad M_{\mathcal{R}(\mathfrak{S},W)} \models \varphi$$

Абстракция данных

Редукция системы основывалась на том, что редуцированная система бисимуляционно эквивалентна исходной: системы до и после имеют одинаковую степень детализации и отличаются только удалением несущественных переменных

Абстракция данных обычно приводит к существенному снижению детальности модели, и бисимуляционную эквивалентность для таких целей использовать нецелесообразно

Вместо этого разумно использовать более «слабое» отношение эквивалентности, предполагающее существенно различающиеся эквивалентные модели и гарантирующее при этом сохранение свойств достаточно многих свойств — например, отношение симуляции

Абстракция данных

Рассмотрим модель Крипке $M = (S, S_0, \to, L)$ и разбиение $\mathfrak{B} = [B_1, \ldots, B_n]$ множества S, согласованное с функцией разметки модели M: для любой пары состояний s, r из одного предиката разбиения верно $L(s) = L(r)$

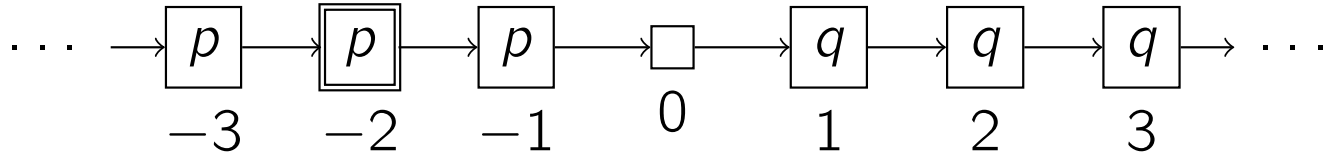
Абстракцией модели M, порождённой согласованным разбиением $\mathfrak{B}$ называется модель $\mathcal{A}(M, \mathfrak{B}) = (S', S'_0, \mapsto, L')$, где:

- $S' = \mathfrak{B}$
- $S'_0 = \{B_i \mid B_i \in \mathfrak{B},\ B_i \cap S_0 \neq \emptyset\}$
- $B_i \mapsto B_j \quad \Leftrightarrow \quad$ существуют состояния $s_i \in B_i$ и $s_j \in B_j$, такие что $s_i \to s_j$
- $L'(B_i) = L(s)$, где $s \in B_i$

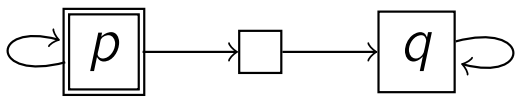
Абстракция данных

Пример

Абстракцией (бесконечной) модели M



для разбиения $\mathfrak{B} = [\{i \mid i \in \mathbb{Z},\ i \leq -1\}, \{0\}, \{j \mid j \in \mathbb{Z}, j \geq 1\}]$ $(\mathcal{A}(M, \mathfrak{B}))$ является модель



Утверждение. **Для любой модели Крипке M и любого согласованного разбиения $\mathfrak{B}$ её множества состояний верно:** $M \preceq abstr(M, \mathfrak{B})$

И это тоже можете попробовать доказать самостоятельно

Следствие. **Для любых модели Крипке M, разбиения $\mathfrak{B}$ её множества состояний, согласованного с функцией разметки, и actl*-формулы φ верно:** **если** $abstr(M, \mathfrak{B}) \models \varphi$**, то** $M \models \varphi$

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 38

Bounded model checking (BMC)
(постановка)

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Вспомним задачу model checking для LTL (MC-LTL)

Дано: конечное множество атомарных высказываний AP, конечная модель Крипке $M = (S, S_0, \mapsto, L)$, ltl-формула φ

Требуется: проверить соотношение $M \models \varphi$

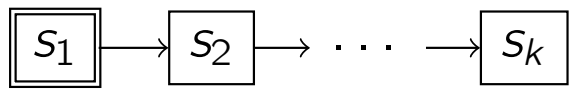
Известно, что это трудная задача (PSPACE-полная)

Но всё же хочется уметь её решать настолько эффективно и в теоретическом, и в практическом смысле, насколько это возможно

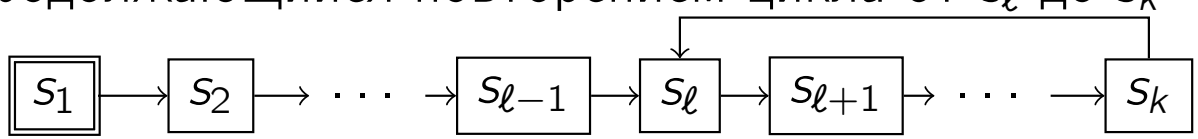
Вступление

Свойство трасс, которое хочется проверить относительно заданной модели на практике, зачастую устроено относительно просто:

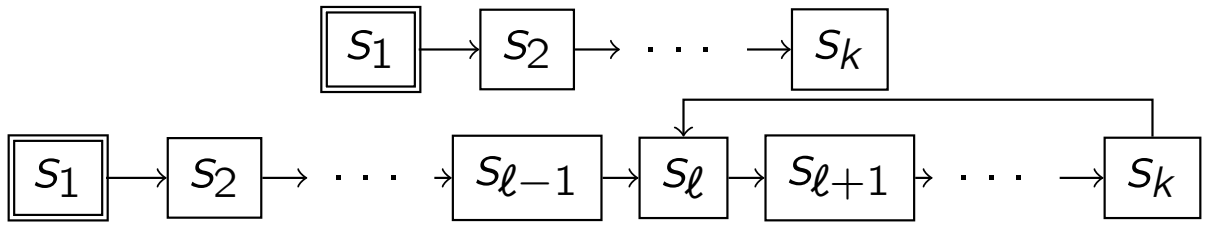
- Это либо свойство безопасности, либо свойство живости
- Если свойство не выполнено, то существует достаточно маленький конечный начальный путь в модели (порядка 10-100 состояний), являющийся «ядром» невыполнимости:
  - Для свойства безопасности это путь, для которого трасса любого продолжения небезопасна



  - Для свойства живости это путь (до состояния s_k) с выделенным состоянием (s_ℓ), обозначающий бесконечный путь с неживой трассой, продолжающийся повторением цикла от s_ℓ до s_k



Отношение ограниченной выполнимости



В предположении о том, что длина пути, представляющеего «ядро» невыполнимости свойства, не превосходит k, можно организовать его поиск как перебор всех путей модели длины не более k

k-путём будем называть путь, содержащий k вершин

(k, ℓ)-циклом, где $\ell \leq k$, а также k-циклом будем называть пару (π, π'), где π — ℓ-путь, $\pi\pi'$ — k-путь и существует переход $\pi\pi'[k] \mapsto \pi\pi'[\ell]$

(k, ℓ)-циклу $\Pi = (\pi, \pi')$ отвечает бесконечный путь $\hat{\Pi} = \pi\pi'\pi' \ldots \pi' \ldots$

Отношение ограниченной выполнимости

Далее будем рассматривать ltl-формулы с поднятыми отрицаниями над множеством AP (negation normal form; NNF), то есть задающиеся БНФ

$$\varphi \quad ::= \quad \mathbb{t} \mid p \mid \neg p \mid \varphi \,\&\, \varphi \mid \varphi \vee \varphi$$
$$\mid \quad \mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \varphi\mathbf{U}\varphi \mid \varphi\mathbf{R}\varphi,$$

где φ — nnf-формула и $p \in \mathrm{AP}$

Напоминание: для «обычного» LTL верно $\psi_1\mathbf{R}\psi_2 = \neg(\neg\psi_1\mathbf{U}\neg\psi_2)$

Это ограничение синтаксиса ltl-формул *аналогично* ограничению, задаваемому для actl*-формул в языке CTL*

Некоторые аналогии можно проследить и для свойств этих двух фрагментов

Отношение ограниченной выполнимости

Рассмотрим nnf-формулу φ и модель Крипке $M = (S, S_0, \mapsto, L)$

Отношение k-выполнимости φ на k-пути или k-цикле π модели M ($M, \pi \models_k \varphi$) зададим так:[1]

- Если π — k-цикл, то
$$M, \pi \models_k \varphi \quad \Leftrightarrow \quad M, \hat{\pi} \models \varphi$$
- Если π — k-путь, то
$$M, \pi \models_k \varphi \quad \Leftrightarrow \quad M, \pi \models_k^1 \varphi$$
- Соотношение $M, \pi \models_k^i \mathrm{tt}$ всегда верно
- $M, \pi \models_k^i p$ для $p \in \mathrm{AP} \quad \Leftrightarrow \quad p \in L(\pi[i])$
- $M, \pi \models_k^i \neg p$ для $p \in \mathrm{AP} \quad \Leftrightarrow \quad p \notin L(\pi[i])$
- $M, \pi \models_k^i \psi_1 \,\&\, \psi_2 \quad \Leftrightarrow \quad M, \pi \models_k^i \psi_1$ и $M, \pi \models_k^i \psi_2$
- $M, \pi \models_k^i \psi_1 \vee \psi_2 \quad \Leftrightarrow \quad M, \pi \models_k^i \psi_1$ или $M, \pi \models_k^i \psi_2$

1 Biere et al. Bounded Model Checking. 2003

Отношение ограниченной выполнимости

Рассмотрим nnf-формулу φ и модель Крипке $M = (S, S_0, \mapsto, L)$

Отношение k-выполнимости φ на k-пути или k-цикле π модели M ($M, \pi \models_k \varphi$) зададим так:[1]

- Соотношение $M, \pi \models_k^i \mathbf{G}\psi$ всегда неверно
- $M, \pi \models_k^i \mathbf{F}\psi \quad \Leftrightarrow \quad$ существует момент времени j, такой что $i \le j \le k$ и $M, \pi \models_k^j \psi$
- $M, \pi \models_k^i \mathbf{X}\psi \quad \Leftrightarrow \quad i < k$ и $M, \pi \models_k^{i+1} \psi$
- $M, \pi \models_k^i \psi_1 \mathbf{U} \psi_2 \quad \Leftrightarrow \quad$ существует момент времени j, такой что
  - $i \le j \le k$,
  - $M, \pi \models_k^j \psi_2$ и
  - для любого момента времени m, такого что $i \le m < j$, верно $M, \pi \models_k^m \psi_1$
- $M, \pi \models_k^i \psi_1 \mathbf{R} \psi_2 \quad \Leftrightarrow \quad M, \pi \models_k^i \psi_2 \mathbf{U} \psi_1$

1 Biere et al. Bounded Model Checking. 2003

Отношение ограниченной выполнимости

Для ограниченной выполнимости очевидным образом неверны некоторые законы, справедливые для «неограниченной» выполнимости

Например, $M, \pi \models_k \mathbf{G}\neg p \quad \not\Leftrightarrow \quad M, \pi \not\models_k \mathbf{F}p$

Для самостоятельного размышления:

1. Какие из законов (равносильностей), приводившихся в лекциях для отношения $\models$, справедливы для $\models_k$, а какие нет?
2. В частности, справедливы ли равносильности, приводившиеся в блоке 28 в доказательстве лемм о том, что предикаты $Sat_M(\mathbf{EG}\varphi)$ и $Sat_M(\mathbf{E}(\varphi\mathbf{U}\psi))$ являются неподвижными точками преобразователей, приведённых в соответствующих леммах?

Отношение ограниченной выполнимости

Утверждение. **Для любых nnf-формулы φ, модели Крипке M, момента времени k, k-пути π и бесконечного пути $\pi\pi'$ в M верно:**

если $M, \pi \models_k \varphi$**, то** $M, \pi\pi' \models \varphi$

Утверждение

Для любых nnf-формулы φ и модели Крипке M верно:
если $M \not\models \neg\varphi$**,**
то существует k-путь или k-цикл, такой что $M, \pi \models_k \varphi$

Nnf-формула φ k-выполняется на модели Крипке M ($M \models_k \varphi$), если **существует** начальный k-путь или k-цикл π, такой что он отвечает начальному пути в M или является таким путём и верно $M, \pi \models_k \varphi$

Теорема. **Для любых nnf-формулы φ и модели Крипке M верно:**

$M \not\models \neg\varphi$ $\Leftrightarrow$ **существует момент времени k, такой что** $M \models_k \varphi$

Проверка невыполнимости LTL-формулы φ на модели Крипке M $M \not\models \psi$ может быть записана относительно языка CTL* как $M \not\models \mathbf{A}\psi$
и переписана в виде $M \models \mathbf{E}\neg\psi$,
то есть для ltl-формулы $\varphi = \neg\psi$ — как $M \models \mathbf{E}\varphi$

Постановка задачи bounded model checking

Задача bounded model checking (BMC) формулируется так: для заданных момента времени k, конечной модели Крипке M и nnf-формулы φ проверить соотношение $M \models_k \varphi$

Иными словами, BMC — это задача MC-LTL, переформулированная как поиск пути заданной длины, являющегося «ядром» бесконечного пути, опровергающего выполнимость ltl-формулы с поднятыми отрицаниями

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 39

Проблема выполнимости булевых формул (SAT)

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Задача MC-LTL трудна (PSPACE-полна), а значит, не имеет достаточно эффективного решения

BMC (проверку соотношения $M \models_k \varphi$) можно расценивать как попытку «упростить» задачу MC-LTL, сохранив общность и открыв возможность изменять сложность за счёт выбора параметра k

Разумно предположить, что для нетривиальных значений k эта задача всё равно останется трудной как минимум в теоретическом смысле

Однако в современном мире задачи, трудные в теоретическом смысле, иногда считаются достаточно простыми и эффективно решаемыми на практике

Проблема выполнимости булевых формул (SAT)

КНФ (конъюнктивная нормальная форма) над n переменными — это булева формула вида

$$D_1 \,\&\, \ldots \,\&\, D_k,$$

где $k \geq 0$ (для $k = 0$ полагаем КНФ равной 1) и для каждого i, $1 \leq i \leq k$, множитель D_i представляет собой дизъюнкт

$$L_1 \vee \cdots \vee L_m,$$

где $m \geq 0$ (для $m = 0$ полагаем дизъюнкт равным 0) и для каждого j, $1 \leq j \leq m$, L_i представляет собой литеру: одну из булевых переменных $x_1, \ldots, x_n$ или её отрицание

Булева формула выполнима, если реализуемая ей функция истинна (принимает значение 1) хотя бы на одном наборе значений переменных

Проблема выполнимости булевых формул (SAT) формулируется так: для заданного натурального числа n и заданной КНФ K над n переменными проверить выполнимость K

Проблема выполнимости булевых формул (SAT)

Иногда рассматривается более широкая постановка задачи SAT, в которой вместо КНФ можно использовать произвольную формулу над задарнее заданной системой булевых функций (например, над $\{\&, \vee, \neg\}$)
На практике такое расширение оказывается несущественным:

- Подформула вида $f(\ldots, \varphi, \ldots)$, где φ не переменная, может быть заменена на $f(\ldots, x, \ldots) \&(x \leftrightarrow \varphi)$, где x — переменная, не встречавшаяся ранее в КНФ
- Подформулы вида $f(x_{i_1}, \ldots, x_{i_m})$ и $(x \leftrightarrow f(x_{i_1}, \ldots, x_{i_m}))$, где f — предзаданная функция и $x_{i_1}, \ldots, x_{i_m}$ — переменные, могут быть заменены на (настолько же предзаданные, как и функция) равносильные КНФ
- Последовательно применяя эти преобразования, можно получить из произвольной формулы настолько же (не)выполнимую КНФ, добавив линейное (относительно размера исходной формулы) число переменных и литер

Имея в виду эту взаимосвязь, будем в примерах и алгоритмах выбирать наиболее удобную из постановок задачи SAT

Проблема выполнимости булевых формул (SAT)

Известно, что задача SAT вычислительно трудна (NP-полна)

То есть с теоретической точки зрения эта задача скорее всего не имеет эффективного (*полиномиального*) решения

А если обнаружится, что эффективное решение всё-таки есть, то и у теоретиков, и у практиков возникнет очень много проблем, проистекающих из предположения, что такого решения нет — например:

- Ряд методов защиты информации, основанных на трудных (NP-полных или близких к ним) задачах, перестанет работать в теории
- Методы эффективного приближенного или частного решения многих трудных задач окажутся неактуальными

Проблема выполнимости булевых формул (SAT)

Но на практике в современном мире принято считать, что существуют способы эффективного решения задачи SAT (хотя и не настолько эффективного, чтобы можно было столкнуться с упомянутыми ранее проблемами):

- Эффективные решающие алгоритмы
- Эффективные программные средства (SAT-решатели)
  - MiniSAT, zChaff, RSat, Lingeling, WinSAT, ...
- История успеха применения этих средств к КНФ с **миллионами** переменных и множителей, возникающим на практике
- Конференции, соревнования и коммерческие запросы, «подстёгивающие» дальнейшее повышение практической эффективности
- **Но никто не понимает, почему оно так хорошо работает**

Современные SAT-решатели, как правило, основываются на двух видах алгоритмов:

- DPLL/CDCL
- Локального поиска

SAT: локальный поиск

Алгоритмы локального поиска устроены так:

1. Выбирается начальный набор значений переменных
2. Если на текущем наборе КНФ истинна, то завершить работу
3. Иначе выбирается переменная, её значение изменяется на противоположеное, и повторяется (2)
4. По необходимости выбирается новый набор значений переменных либо алгоритм досрочно завершается

Выбор, как правило, выполняется с привлечением случайности

Цель «блужданий» по наборам значений, как правило, — получение как можно большего числа истинный множителей

Если алгоритм завершился досрочно, то констатируется, что выполнимость КНФ неизвестна

SAT: DPLL/CDCL

CDCL (Conflict-driven clause learning) — это способ организации алгоритмов решения задачи SAT, наиболее хорошо зарекомендовавший себя на практике на текущий момент

Этот способ является «умной» надстройкой над способом организации DPLL (по фамилиям создателей: Davis, Putnam, Logemann, Loveland)

Чтобы не перегружать рассказ деталями, остановимся только на кратком описании схемы работы DPLL

SAT: DPLL/CDCL

Общая схема работы DPLL устроена так:

1. Если текущая КНФ имеет вид 1, то работа завершается: «да» (КНФ выполнима)
2. Если текущая КНФ содержит множитель 0, то работа завершается: «нет» (КНФ невыполнима)
3. Иначе
   - 3.1 Выбираются переменная (x) и её значение 0 или 1 (b)
   - 3.2 Подставляется значение x/b, формула оптимизируется, и (рекурсивно) проверяется выполнимость оптимизированной формулы
   - 3.3 Если последняя формула выполнима, то работа завершается: «да»
   - 3.4 Иначе подставляется значение $x/\neg b$, формула оптимизируется, и возвращается результат (рекурсивной) проверки её выполнимости

Основные оптимизации:

- Удаление слагаемых 0 и множителей со слагаемым 1
- Свёртка констант: если есть множитель с одной литерой x или $\neg x$, то подставляется значение $x/1$ (для литеры x) или $x/0$ (для $\neg x$)
- Деполяризация: если переменная x входит в КНФ только без отрицания (только с отрицанием), то подставляется $x/1$ ($x/0$)

SAT: DPLL/CDCL

Пример

$$(x_1 \vee x_2 \vee x_3) \& (x_1 \vee \neg x_2 \vee x_3) \& (\neg x_1 \vee \neg x_2) \& (\neg x_2 \vee \neg x_3)$$

SAT: DPLL/CDCL

Пример

$$(x_1 \vee x_2 \vee x_3) \,\&\, (x_1 \vee \neg x_2 \vee x_3) \,\&\, (\neg x_1 \vee \neg x_2) \,\&\, (\neg x_2 \vee \neg x_3)$$

$x_2/1$, удаление множителей ↙

$$(x_1 \vee x_3) \,\&\, \neg x_1 \,\&\, \neg x_3$$

свёртка x_1 и x_3 ↓

$$0$$

SAT: DPLL/CDCL

Пример

$$(x_1 \vee x_2 \vee x_3) \,\&\, (x_1 \vee \neg x_2 \vee x_3) \,\&\, (\neg x_1 \vee \neg x_2) \,\&\, (\neg x_2 \vee \neg x_3)$$

$x_2/1$, удаление множителей

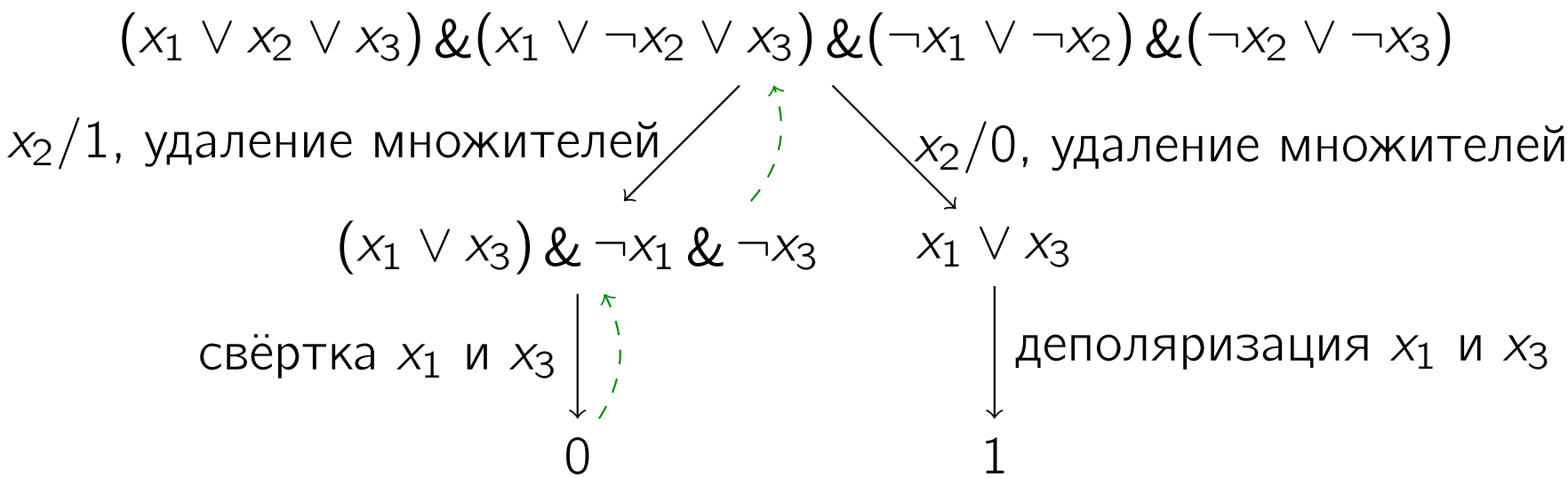
$$(x_1 \vee x_3) \,\&\, \neg x_1 \,\&\, \neg x_3$$

свёртка x_1 и x_3

$$0$$

SAT: DPLL/CDCL

Пример



Получили КНФ 1: конец выполнения, исходная КНФ выполнима

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 40

Решение BMC при помощи SAT

Лектор:
Подымов Владислав Васильевич
E-mail:
valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Вступление

Задача BMC *трудна*

Задача SAT *тоже трудна, но не очень сильно*: NP-полна

При этом в некотором смысле задача SAT *проста*: имеются до некоторой степени эффективные средства её решения

Обсудим то, как можно использовать практически эффективное решение задачи SAT для повышения эффективности решения задачи BMC

BMC→SAT: общая схема

Покажем, как по входным данным задачи BMC (число k, модель Крипке M, nnf-формула φ) построить формулу $\Phi^k_{M,\varphi}$, такую что

$$\Phi^k_{M,\varphi} \text{ выполнима} \quad \Leftrightarrow \quad M \models_k \varphi$$

Тогда решение задачи BMC можно устроить так:

1. Построить формулу $\Phi^k_{M,\varphi}$
2. Проверить выполнимость этой формулы (*или соответствующей КНФ*)
3. Вернуть результат этой проверки как ответ к задаче

Если формула $\Phi^k_{M,\varphi}$ будет иметь *достаточно небольшой* размер относительно (k, M, φ), то такое решение можно считать эффективным (*настолько же, насколько эффективны современные SAT-решатели*)

BMC→SAT: общая схема

Формулу $\Phi^k_{M,\varphi}$ устроим так:

$$\Phi^k_{M,\varphi} = \Phi^k_M \,\&\, (\Phi^k_\varphi \vee \Psi^k_\varphi), \text{ где}$$

- Φ^k_M представляет множество всех начальных k-путей модели M
- Φ^k_φ представляет множество всех начальных k-путей, на которых k-выполнена φ
- $\Psi^k_\varphi = \bigvee\limits_{\ell=1}^{k} (\Psi^k_\ell \,\&\, \Psi^k_{\ell,\varphi})$, где
  - Ψ^k_ℓ — условие, отвечающее возможности разбить k-путь на (k,ℓ)-цикл, и
  - $\Psi^k_{\ell,\varphi}$ представляет множество всех (k,ℓ)-циклов, на которых k-выполнена φ

BMC→SAT: пути модели

Рассмотрим символьное представление модели Крипке $M = (S, S_0, \mapsto, L)$:

- Для кодирования состояний выбрано m переменных: $u_1, \ldots, u_m$
- Каждому состоянию s сопоставлена элементарная конъюнкция χ_s от переменных $u_1, \ldots, u_m$
- Множествам S, S_0 и отношению $\mapsto$ сопоставлены стандартные представления χ_S, χ_{S_0} и $\chi_\mapsto$
- Для простоты положим, что $\mathrm{AP} = S$ и для каждого состояния s верно $L(s) = \{s\}$

$[\vec{u}/\vec{w}]$ — так сократим запись $[u_1/w_1, \ldots, u_m/w_m]$

Состоянию $\pi[i]$ начального k-пути π сопоста́вим переменные $x_1^i, \ldots, x_m^i$

Формула, представляющая множество всех начальных k-путей:

$$\Phi_M^k = \chi_{S_0}[\vec{u}/\vec{x}^1] \,\&\, \&_{i=2}^{k} \chi_\mapsto[\vec{u}/\vec{x}^{i-1}, \vec{u}'/\vec{x}^i]$$

BMC→SAT: k-пути формулы

Формула, представляющая множество всех начальных k-путей, на которых k-выполнена φ: $\Phi^k_\varphi = \Phi^k_{1,\varphi}$

Для $i \leq k$:

- $\Phi^k_{i,s} = \chi_s[\vec{u}/\vec{x}^i]$
- $\Phi^k_{i,\neg s} = \neg\chi_s[\vec{u}/\vec{x}^i]$
- $\Phi^k_{i,\psi_1 \,\&\, \psi_2} = \Phi^k_{i,\psi_1} \,\&\, \Phi^k_{i,\psi_2}$
- $\Phi^k_{i,\psi_1 \vee \psi_2} = \Phi^k_{i,\psi_1} \vee \Phi^k_{i,\psi_2}$
- $\Phi^k_{i,\mathbf{X}\psi} = \Phi^k_{i+1,\psi}$
- $\Phi^k_{i,\mathbf{F}\psi} = \Phi^k_{i,\psi} \vee \Phi^k_{i+1,\mathbf{F}\psi}$
- $\Phi^k_{i,\mathbf{G}\psi} = 0$
- $\Phi^k_{i,\psi_1\mathbf{U}\psi_2} = \Phi^k_{i,\psi_2} \vee (\Phi^k_{i,\psi_1} \,\&\, \Phi^k_{i+1,\psi_1\mathbf{U}\psi_2})$
- $\Phi^k_{i,\psi_1\mathbf{R}\psi_2} = \Phi^k_{i,\psi_2\mathbf{U}\psi_1}$

Кроме того, для любой nnf-формулы ψ и любого m, $m > k$, верно $\Phi^k_{m,\psi} = 0$

BMC→SAT: (k, ℓ)-циклы формулы

Легко видеть, что k-путь π можно разбить на (k, ℓ)-цикл $\Leftrightarrow$ в отношении $\mapsto$ содержится дуга $(\pi[k], \pi[\ell])$

Формула, обозначающая, что k-путь может быть разбит на (k, ℓ)-цикл для заданного ℓ:

$$\Psi_\ell^k = \psi_\mapsto[\vec{u}/\vec{x}^k, \vec{u}'/\vec{x}^\ell]$$

$next_\ell^k(i)$ — так обозначим номер состояния, следующего за i-м в (k, ℓ)-цикле:

- Если $i < k$, то $next_\ell^k(i) = i + 1$
- $next_\ell^k(k) = \ell$

BMC→SAT: (k, ℓ)-циклы формулы

Формула, представляющая множество всех (k, ℓ)-циклов, на которых k-выполнена φ: $\Psi^k_{\ell,\varphi} = \Psi^k_{1,\varphi}$

- $\Psi^k_{i,s} = \chi_s[\vec{u}/\vec{x}^i]$
- $\Psi^k_{i,\neg s} = \neg\chi_s[\vec{u}/\vec{x}^i]$
- $\Psi^k_{i,\psi_1 \,\&\, \psi_2} = \Psi^k_{i,\psi_1} \,\&\, \Psi^k_{i,\psi_2}$
- $\Psi^k_{i,\psi_1 \vee \psi_2} = \Psi^k_{i,\psi_1} \vee \Psi^k_{i,\psi_2}$
- $\Psi^k_{i,\mathbf{X}\psi} = \Psi^k_{next^k_\ell(i),\psi}$
- $\Psi^k_{i,\mathbf{F}\psi} = \Psi^{k,k}_{i,\mathbf{F}\psi}$, $\Psi^{k,m}_{i,\mathbf{F}\psi} = \Psi^k_{i,\psi} \vee \Psi^{k,m-1}_{next^k_\ell(i),\mathbf{F}\psi}$ для $m > 1$, $\Psi^{k,1}_{i,\mathbf{F}\psi} = \Psi^k_{i,\psi}$
- $\Psi^k_{i,\mathbf{G}\psi} = \Psi^{k,k}_{i,\mathbf{G}\psi}$, $\Psi^{k,m}_{i,\mathbf{G}\psi} = \Psi^k_{i,\psi} \,\&\, \Psi^{k,m-1}_{next^k_\ell(i),\mathbf{G}\psi}$ для $m > 1$, $\Psi^{k,1}_{i,\mathbf{G}\psi} = \Psi^k_{i,\psi}$
- $\Psi^k_{i,\psi_1\mathbf{U}\psi_2} = \Psi^{k,k}_{i,\psi_1\mathbf{U}\psi_2}$
  $\Psi^{k,m}_{i,\psi_1\mathbf{U}\psi_2} = \Psi^k_{i,\psi_2} \vee (\Psi^k_{i,\psi_1} \,\&\, \Psi^{k,m-1}_{next^k_\ell(i),\psi_1\mathbf{U}\psi_2})$ для $m > 1$
  $\Psi^{k,1}_{i,\psi_1\mathbf{U}\psi_2} = \Psi^k_{i,\psi_2}$
- $\Psi^k_{i,\psi_1\mathbf{R}\psi_2} = \Psi^k_{i,\psi_2\mathbf{U}\psi_1}$

BMC→SAT: заключение

Теорема. **Для любых модели Крипке M, натурального числа k и nnf-формулы φ верно:**

$$M \models_k \varphi \quad \Leftrightarrow \quad \textbf{булева формула } \Phi^k_{M,\varphi} \textbf{ выполнима}$$

Можете попробовать доказать это сами (и это технически непросто)

А какой размер имеет булева формула $\Phi^k_{M,\varphi}$ относительно числа k, количества состояний n и переходов m модели M и относительно числа операций q в формуле φ?

Описанная трансляция не очень эффективна, но можно её улучшить настолько, чтобы размер $\Phi^k_{M,\varphi}$ оказался линейным

Это в сочетании с использованием символьного представления, эффективностью SAT-решателей и тезисом о том, что на практике верхняя граница для числа k, за пределы которой увеличивать k нет смысла, невысока, обеспечивает эффективность решения задачи BMC, основанного на SAT