Формальные методы верификации. Темпоральная логика LTL

Данилов И. Г.
к.т.н., ассистент каф. МОП ЭВМ

Институт компьютерных технологий и информационной безопасности ЮФУ

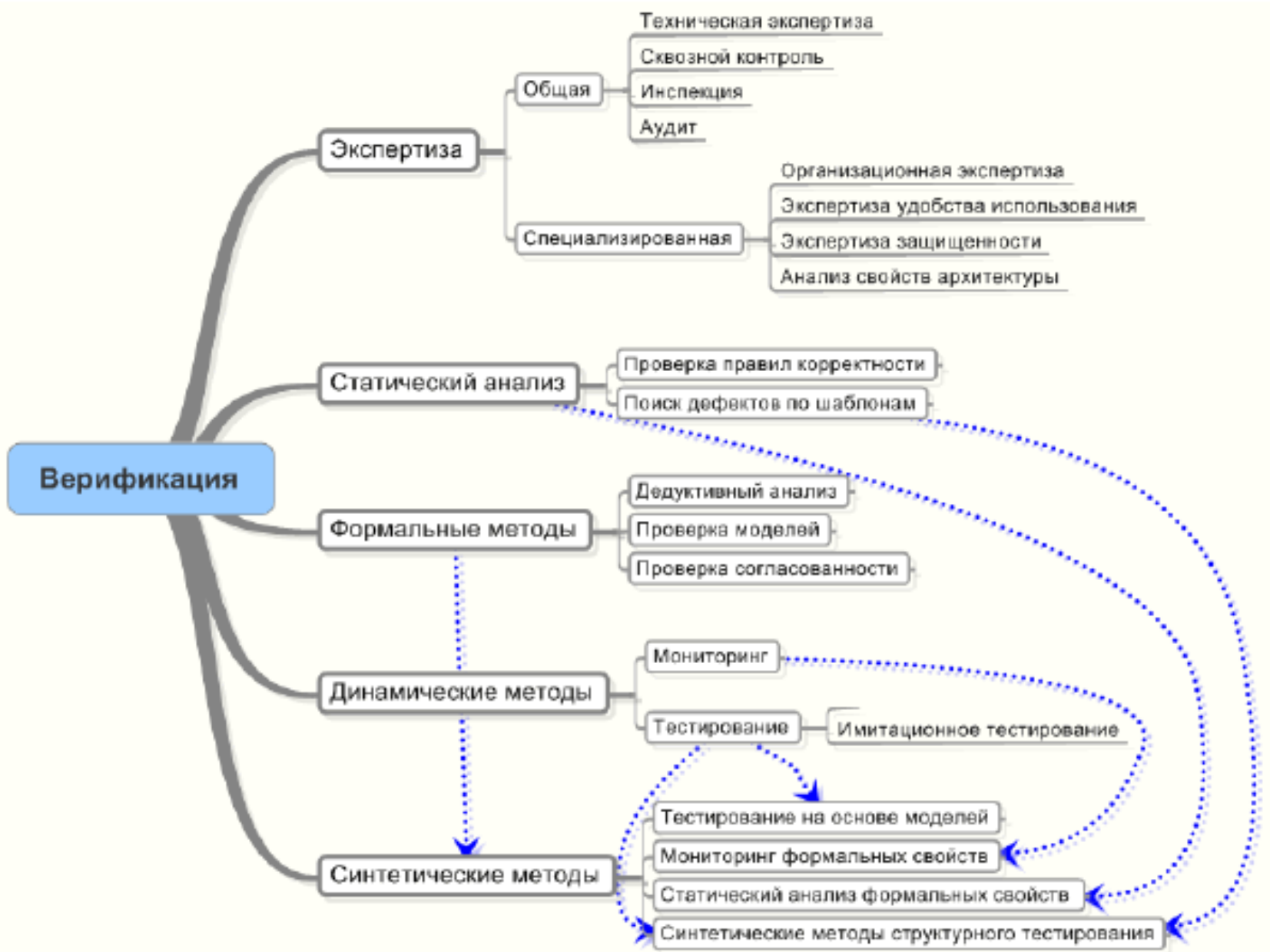
24 сентября 2016 г.

Материалы для занятия

Используемые материалы:

- Методы верификации программного обеспечения, В.В. Кулямин;
- Практикум по дедуктивной верификации программ, Д.В. Буздалов, Е.В. Корныхин, А.А, Панфёров, А.К. Петренко, А.В. Хорошилов;
- Верификация параллельных и распределенных программных систем, лекция в Comp. Sci. Club 18.03.2012, Ю.Г. Карпов, И.В. Шошмина, А. Б. Беляев;
- Model Checking. Верификация параллельных и распределенных программных систем, Ю.Г. Карпов.

Схема классфикации методов верификации



Формальные методы верификации

Формальные методы верификации ПО используют формальные модели **требований, поведения и окружения ПО** для анализа его свойств.

Такие модели являются либо **логико-алгебраическими**, либо **исполнимыми**, либо **промежуточными**, имеющими черты и логико-алгебраических, и исполнимых моделей.

Логико-алгебраические модели

Логико-алгебраические модели (property-based models), они же — логические или алгебраические исчисления.

При моделировании ПО модель такого типа описывает некоторый набор его свойств, быть может, изменяющийся со временем, но не дает точного представления о том, за счет чего изменяются эти свойства.

Примеры логических исчислений

- исчисление высказываний (пропозициональное исчисление, propositional calculus): атомарные высказывания + логические связки ($\wedge, \vee ...$);
- исчисление предикатов (predicate calculus): + кванторы по объектным переменным ($\forall, \exists$);
- исчисления предикатов более высоких порядков (higher-order calculi): кванторы не только по объектным переменным, но и по функциональным или предикатным;
- λ-исчисление (lambda calculus): с помощью ламбда-оператора позволяет строить функции из выражений;
- λ-исчисления более высоких порядков: позволяют применять ламбда-оператор не только к объектам, но и к типам;

Примеры логических исчислений

- модальные логики: допускают построение утверждений с помощью операторов с дополнительной смысловой нагрузкой, давая возможность строго анализировать связи между, например, утверждениями «$x = 3$», «доказано, что $x = 3$», «может быть, что $x = 3$», «всегда будет $x = 3$» или «хотелось бы, чтобы было $x = 3$»;
- специальным случаем модальных логик являются **временные логики** (temporal logics): дополнительные операторы используются для описания временной последовательности событий — «как только x станет равно 3, y должно стать равно 0», «после того, как x станет больше 0, спустя некоторое время y обязательно станет равно 5».
- ...

Исполнимые модели

Исполнимые модели (или операционные, executable models) характеризуются тем, что их можно каким-то образом выполнить, чтобы проследить изменение свойств моделируемого ПО.

Каждая исполнимая модель является, по сути, **программой для некоторой достаточно строго определенной виртуальной машины**.

Зачем это нужно?

Зачем моделировать свойства одних программ через свойства других?

- модели оказываются значительно проще моделируемых систем, их гораздо удобнее анализировать (учитываются не все свойства моделируемого ПО, а только **важные для рассматриваемой в данный момент задачи**);
- виртуальные машины используемых на практике языков программирования очень сложны и определены нечетко, а виртуальные машины моделей значительно более просты и обозримы (исчерпывающий анализ возможного поведения модели, выявление всех классов возможных при ее работе ситуаций);
- за счет **иного взгляда** на систему часто можно увидеть такие характеристик программы и и особенности, на которые ранее просто не обращали внимания.

Зачем это нужно?

Зачем моделировать свойства одних программ через свойства других?

- модели оказываются значительно проще моделируемых систем, их гораздо удобнее анализировать (учитываются не все свойства моделируемого ПО, а только **важные для рассматриваемой в данный момент задачи**);
- виртуальные машины используемых на практике языков программирования очень сложны и определены нечетко, а виртуальные машины моделей значительно более просты и обозримы (исчерпывающий анализ возможного поведения модели, выявление всех классов возможных при ее работе ситуаций);
- за счет **иного взгляда** на систему часто можно увидеть такие характеристик программы и и особенности, на которые ранее просто не обращали внимания.

Исполнимые модели и конечные автоматы

Все виды исполнимых моделей можно считать расширением и обобщением конечных автоматов (finite state machine, FSM):

конечные системы помеченных переходов (или просто системы переходов, labeled transition systems, LTS), расширенные конечные автоматы (extended finite state machines, EFSM), взаимодействующие автоматы (communicating finite state machines, CFSM), иерархические автоматы (hierarchical state machines), временные автоматы (timed automata), сети Петри (Petri nets), ω-автоматы...

Промежуточные модели

- логики Хоара (Hoare logics) являются специфическим видом логик, утверждения которых состоят из формул логики некоторого вида и программных инструкций. В простейшем виде это тройки $\Phi P \Psi$, где P – часть программы на определенном языке, а Φ и Ψ – формулы исчисления высказываний, зависящие от переменных, входящих в P.

Промежуточные модели

- обобщением логик Хоара являются динамические или программные логики (dynamic logics, program logics). Они являются специальным типом модальных логик, в которых операторы модальности связаны с инструкциями программ. Обычно используются операторы [P] и <P>, где P – некоторая программа. Утверждение [P]Φ означает, что всегда после выполнения программы P формула Φ истинна, а <P>Φ – что после выполнения P Φ может оказаться истинной. Тройка Хоара ΦPΨ может быть представлена в динамической логике как $\Phi \Rightarrow [P]\Psi$.

Промежуточные модели

- программные контракты (software contracts), наоборот, являются частным случаем логики Хоара, сужающим возможности использования логических формул. Программный контракт представляет собой описание поведения набора программных компонентов представленное в виде описания сигнатур операций каждого из этих компонентов, структур их состояний, а также предусловий и постусловий для каждой операции и наборов инвариантов для каждого компонента в отдельности.

Классификация формальных методов

Для того, чтобы проверить выполнение тех или иных свойств с помощью формальных методов, необходимо **формализовать свойства и проверяемый артефакт**, т.е. построить формальные модели для того и другого.

Модель проверяемых свойств принято называть **спецификацией**, а модель проверяемого артефакта — **реализацией**.

Заметим, что здесь спецификация и реализация — **термины, обозначающие формальные модели**, а не описание требований и реализующий их набор программ, как обычно.

Классификация формальных методов

- **и спецификация S, и реализация I представлены как логико-алгебраические модели**. В этом случае выполнение специфицированных свойств в реализации моделируется **отношением выводимости**, что обычно записывается как $I \vdash S$. Чаще всего для его проверки используется **метод дедуктивного анализа (theorem proving)**, т.е. проверки того, что набор утверждений, представляющий спецификацию, формально выводится из реализации и, быть может, каких-то гипотез о поведении окружения системы, сформулированных в том же формализме, что и реализация.

Классификация формальных методов

- **спецификация S является логико-алгебраической моделью, а реализация I — исполнимой**. Выполнение специфицированных свойств в реализации в этой ситуации называется **отношением выполнимости** и записывается как $I \models S$. Для его проверки используется **метод проверки моделей (model checking)**, в рамках которого чаще всего выполнимость проверяется непосредственным исследованием всей реализации, или такой ее части, свойства которой полностью определяют свойства всей реализации в целом. Обычно эту работу выполняет не человек, а специализированный инструмент.

Классификация формальных методов

- **и спецификация S, и реализация I представлены как исполнимые модели**. В этом случае общепринятого названия или обозначения для выполнения специфицированных свойств в реализации нет — используются термины “симуляция” или “моделирование” (simulation), “сводимость” (reduction), “соответствие” или “согласованность” (conformance). Для **методов проверки согласованности** пока нет общепринятого названия.

Классификация формальных методов и промежуточные модели

В тех случаях, когда используются модели промежуточного типа, применяемый метод определяется **теми составляющими модели, которые для него наиболее существенны**.

Так, логики Хоара и динамические логики чаще всего используют для дедуктивного анализа, а программные контракты могут применяться в различных методах.

Методы и инструменты дедуктивного анализа

Первые методы дедуктивного анализа программ были предложены **Флойдом и Хоаром** в конце 1960-х годов.

Корни этих методов уходят к Алану Тьюрингу, который в своей лекции Лондонскому математическому обществу в 1947 впервые озвучил идею индуктивных утверждений.

В основе этих методов лежит логика Хоара и предложенная Флойдом техника доказательства завершения циклов, основанная на инвариантах цикла и монотонно изменяющихся в ходе его выполнения оценочных функциях.

Методы Флойда-Хоара

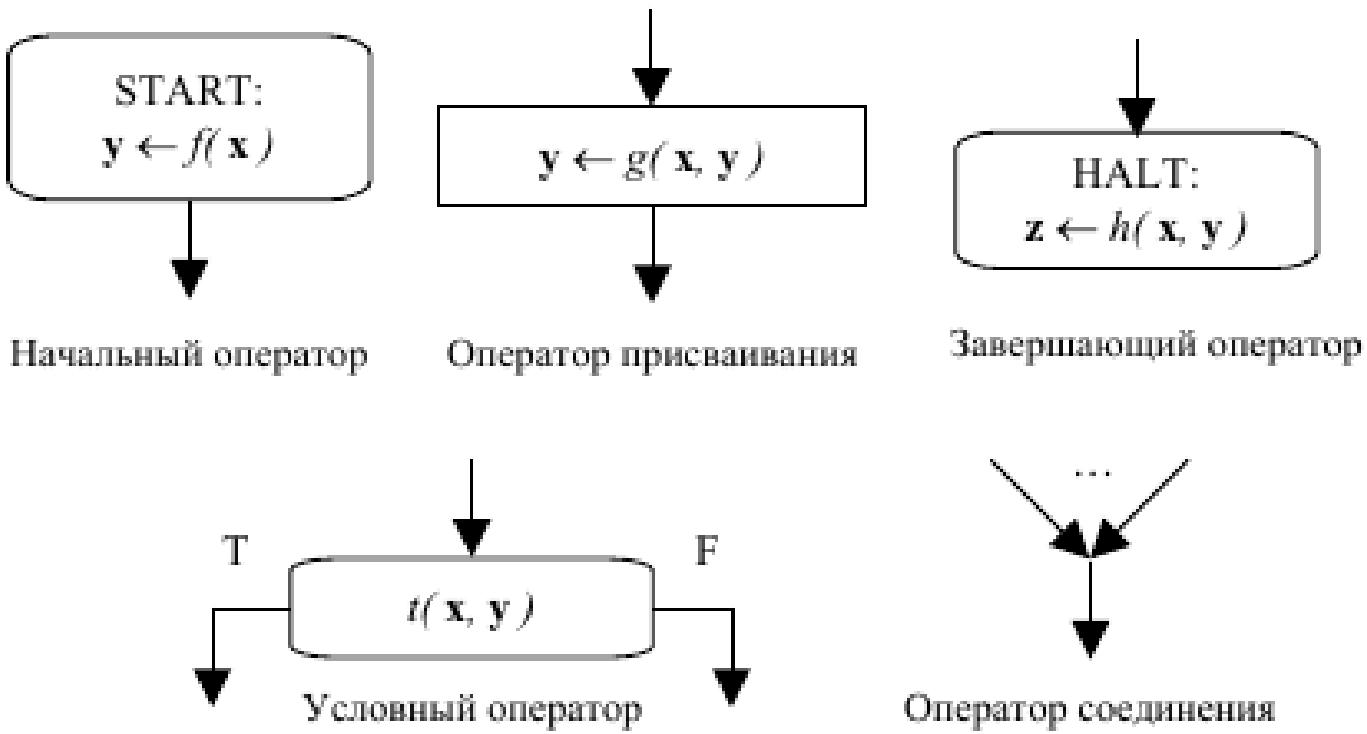
- Спецификация программы в виде ее предусловия и постусловия определяется формально, например, в рамках исчисления высказываний.
- В коде программы или на ее блок-схеме выбираются **точки сечения**, так чтобы любой цикл содержал по крайней мере одну такую точку. Начало и конец программы (все возможные точки выхода из программы можно свести к одной) тоже считаются точками сечения.
- Для каждой точки сечения i находится предикат Φ_i, характеризующий отношения между переменными программы в этой точке. В начале программы в качестве такого предиката выбирается предусловие, в конце — постусловие. Кроме того, выбирается оценочная функция φ, отображающая значения переменных программы в некоторое упорядоченное множество без бесконечных убывающих цепей (например, натуральные числа).

Методы Флойда-Хоара

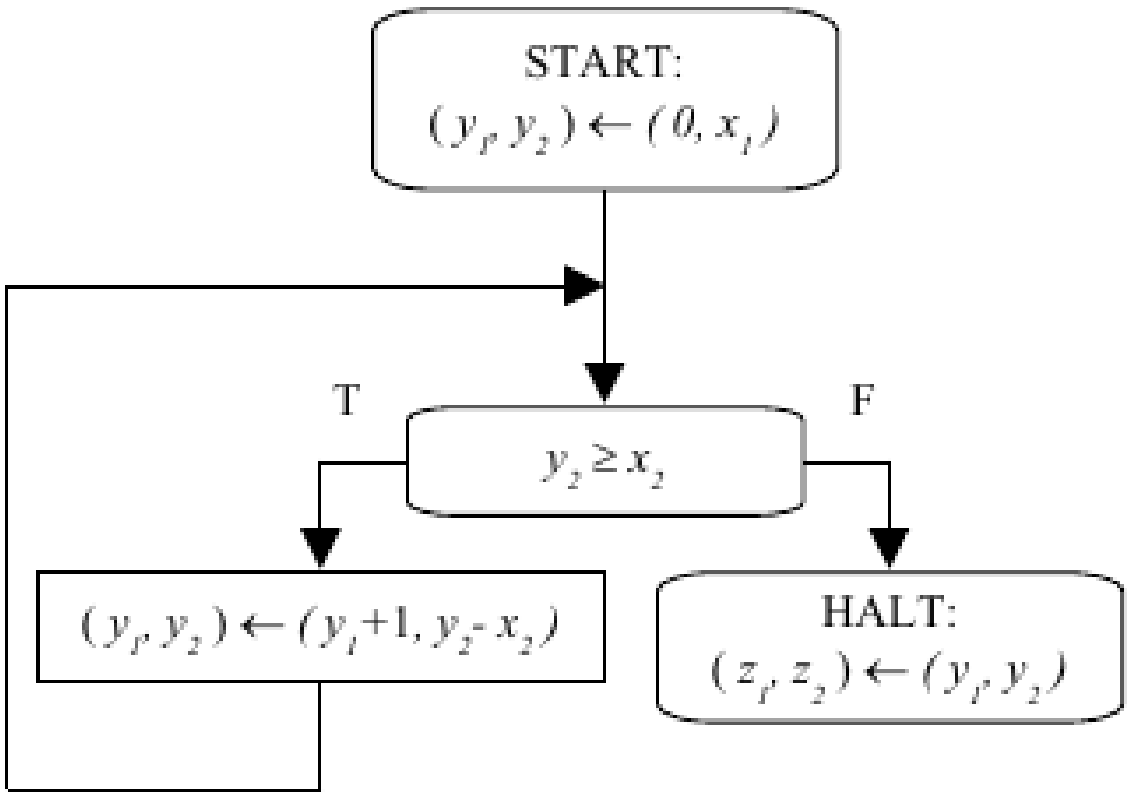
- В результате программа разбивается на набор возможных линейных путей между парами точек сечения. Для каждого такого пути P_{ij} между точками i и j нужно проверить истинность тройки $\Phi_i P_{ij} \Phi_j$. Если это удается, программа частично корректна, т.е. работает правильно, если завершается.
- Для каждого простого цикла (начинающегося и заканчивающегося в одной и той же точке), нужно найти на нем такой путь P_{ij}, для которого можно доказать $\{\Phi_i \ \& \ \varphi = \mathrm{a}\} P_{ij} \{\Phi_j \ \& \ \varphi < \mathrm{a}\}$ для некоторой дополнительной переменной a. Это позволяет утверждать, что цикл завершится, поскольку значения оценочной функции не могут уменьшаться неограниченно.

Методы Флойда-Хоара. Переменные, домены, операторы

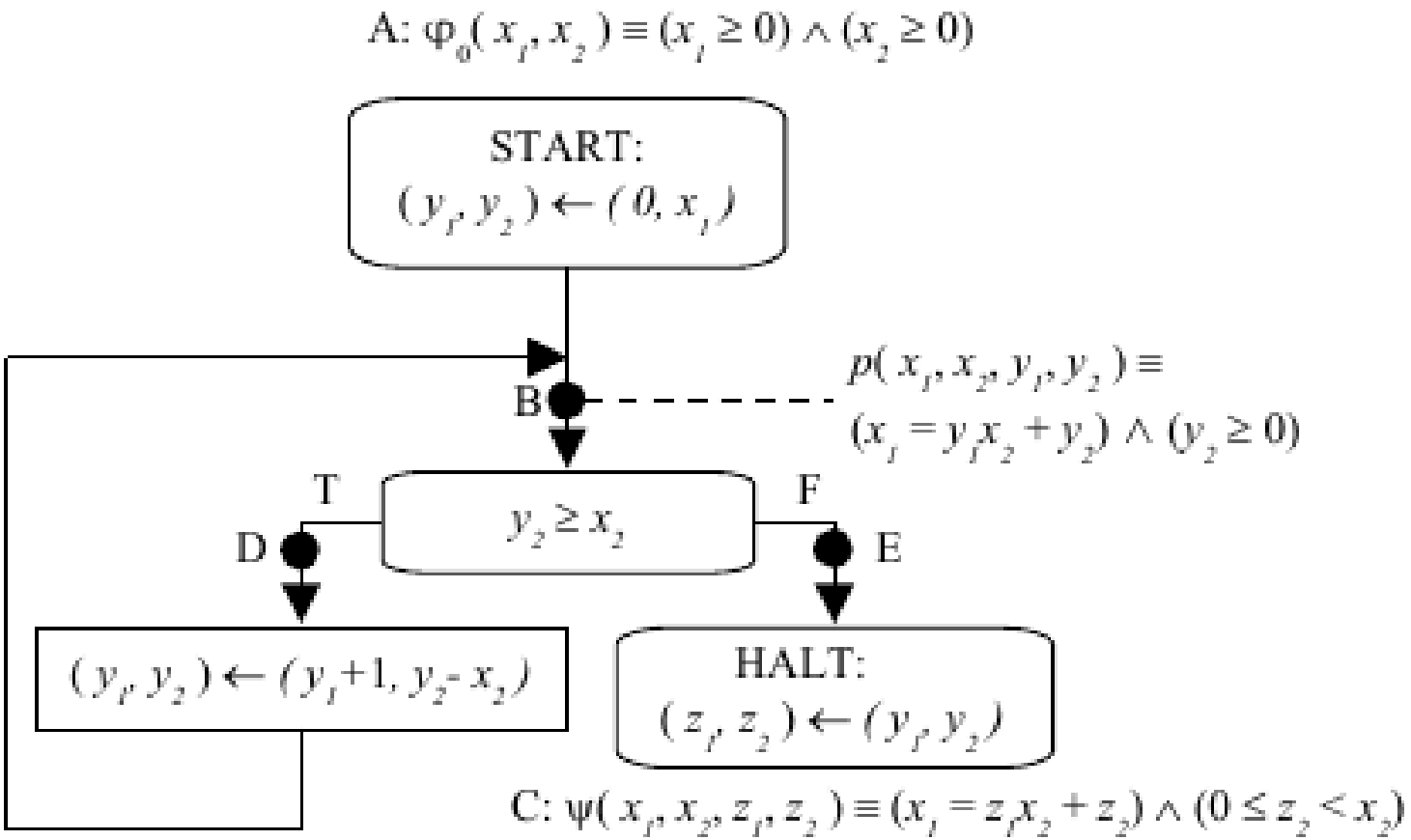
Каждая программа работает с конечным числом переменных. Переменные разделяются на три типа: **входные, промежуточные и выходные**.Каждая переменная v может принимать значения из некоторого множества Dv, которое называется **доменом переменной**. Основные операторы программы:



Методы Флойда-Хоара. Пример: блок-схема целочисленного деления



Методы Флойда-Хоара. Пример: блок-схема целочисленного деления с предикатами



Методы Флойда-Хоара. Ограничения

В таком виде метод Флойда применим лишь к довольно ограниченному классу программ — в них не должно быть массивов или указателей, вызова подпрограмм, параллелизма, взаимодействия с окружением по ходу работы.

В дальнейшем были предложены аналогичные методы для дедуктивной верификации более сложных программ, в том числе параллельных, с использованием более сложных моделей, например, процессных алгебр, вместо исчисления высказываний.

Дедуктивный метод верификации трудно использовать на практике.

Дедуктивный анализ, инструменты

Дедуктивный анализ в принципе может быть выполнен человеком, но для практически значимых систем сам размер спецификации и реализации таков, что необходимо использование специализированных инструментов для **автоматическогопостроения доказательств (provers)** или **предоставляющих существенную помощь в их осуществлении (proof assistants)**.

- инструменты, основанные на расширениях пропозициональной логики или логик первого порядка — Vampire, Waldmeister, Darwin...;
- инструменты, основанные на логиках высших порядков — PVS, Coq...

Метод проверки моделей (model-checking)

Проверка моделей (model checking) используется для проверки выполнения набора свойств, записанных в виде утверждений какого-либо логико-алгебраического исчисления на исполнимой модели, моделирующей определенные проектные решения или код ПО.

Премия ACM

21 июня 2008 г премия Тьюринга была вручена трем создателям техники MODEL CHECKING, внесшим наиболее существенный вклад: Edmund M. Clarke (CMU), E. Allen Emerson (U Texas, Austin), Joseph Sifakis (Verimag, France).

“за их роль в превращении метода Model checking в высокоэффективную технологию верификации, широко используемую в индустрии разработки ПО и аппаратных средств”

Ограниченность классической логики для выражения свойств динамики (процессов, развивающихся во времени)

- Классическая логика
  - Высказывания статичны, неизменны во времени
- Пример - некоммутативность конъюнкции, A&B ≠ B&A:
  - *"Джону стало страшно и он убил"* ⇐?⇒ *"Джон убил и ему стало страшно"*
  - *"Смит умер и его похоронили"* ⇐?⇒ *"Смита похоронили и он умер"*
  - *"Джейн вышла замуж и родила ребенка"* ⇐?⇒ *"Джейн родила и вышла замуж"*
- В классической логике высказываний не формализуются:
  - *Путин - президент России* (истинно только в какой-то период)
  - *Мы не друзья, пока ты не извинишься*
  - *Если m поступило на вход в канал, то m обязательно появится на выходе*
  - *Запрос к лифту с произвольного этажа будет обязательно удовлетворен*

Мы хотим верифицировать системы, развивающиеся во времени, а обычная классическая логика неадекватна для выражения их свойств!

Как ввести понятие времени в логические утверждения?

Использование модальностей (Tense Logic, A.Prior)

Fq – q случится когда-нибудь в будущем

Pq – q случилось когда-то в прошлом

Gq – q всегда будет в будущем

Hq – q всегда было в прошлом

pUq – q случится в будущем, а до него непрерывно будет выполняться p

Xp - p выполнится в следующий момент

Примеры формализации высказываний

- *Джон умер и его похоронили*

  P(Джон_умирает $\wedge$ XF *Джона_хоронят*)

- *Если я видел ее раньше, то я ее узнаю при встрече*

  G(P Увидел $\Rightarrow$ G(Встретил $\Rightarrow$ X Узнал))

- *Ленин – жил, Ленин – жив, Ленин – будет жить* (В.В.Маяковский)

  PG Ленин_жив

- *Мы придем к победе коммунистического труда!*

  F Коммунистический_труд_победил!

- *Сегодня он играет джаз, а завтра Родину продаст*

  G(Он_играет_джаз $\Rightarrow$ FX Он_продает_Родину)

- *Раз Персил – всегда Персил (раз попробовав, будешь всегда)*

  G(Персил $\Rightarrow$ G Персил)

- *Любой запрос к ресурсу будет висеть до подтверждения либо отклонения*

  G(Request $\Rightarrow$ Request U (Reject $\oplus$ Confirm))

Набор упражнений 1

Запишите следующие утверждения:

1. “Джейн вышла замуж и родила ребенка”.
2. “Пока ключ зажигания не вставлен, машина не поедет”.
3. “Джон убил и ему стало страшно”.
4. “Лифт никогда не пройдет мимо этажа, вызов от которого поступил, но еще не обслужен”.

Слабый U

В темпоральной логике иногда используется оператор W, так называемый “слабый Until” (Unless), имеющий следующую семантику:

$$p\ W\ q = p\ U\ q \vee Gp$$

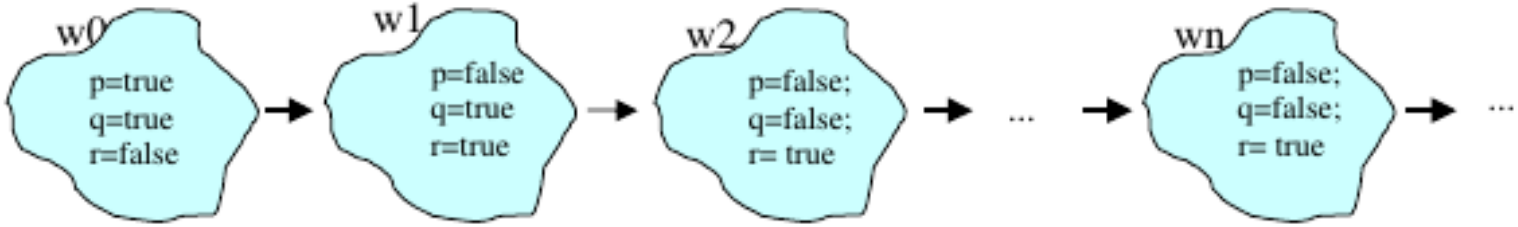
Пример формализации высказывания “Мы будем бороться, пока не победим”:

- Мы_боремся U Мы_победили — есть уверенность в победе;
- Мы_боремся W Мы_победили — мы будем бороться, но уверенности в победе нет.

LTL в дискретном времени – А. Пнуэли (1977)

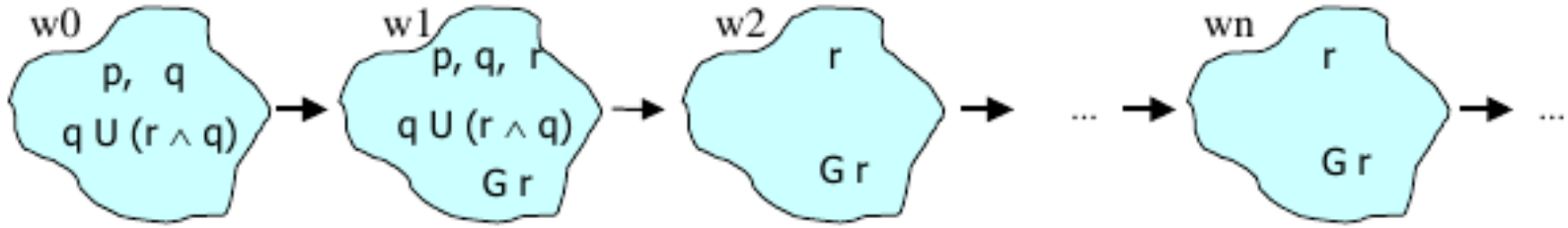
Модель времени – последовательность целых (вчера, сегодня, завтра, …)

Семантика “возможных миров” Лейбница, в каждом свое понимание истинности:



В каждом мире произвольная логическая формула истинна, либо нет:

Это же справедливо и для произвольной темпоральной формулы:



На всей цепочке выполняются формулы p, q, qU(r∧q), - *они истинны в w0*

Linear Temporal Logic (LTL)

- Формальное определение логики LTL
  - Формула ϕ PLTL это :
    - атомарное утверждение (атомарный предикат) p, q, ...,
    - или Формулы, связанные логическими операторами $\vee, \neg$
    - или Формулы, связанные темпоральными операторами U, X
  - Модальных операторов прошлого в LTL нет

Грамматика: $\phi ::= p \mid \phi \vee \phi \mid \neg\phi \mid X\phi \mid \phi\ U\ \phi \mid F\phi \mid G\ \phi$

Выводимые темпоральные операторы :

$Fp = true\ U\ p$

$Gp = \neg F\ \neg p$

Прошлое при анализе технических систем не важно

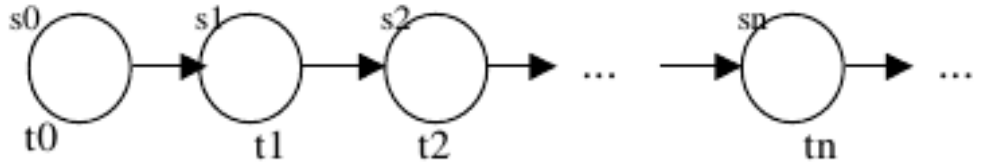
Примеры

Пусть p означает “Я люблю Машу”, а q — “Я люблю Дашу”, тогда:

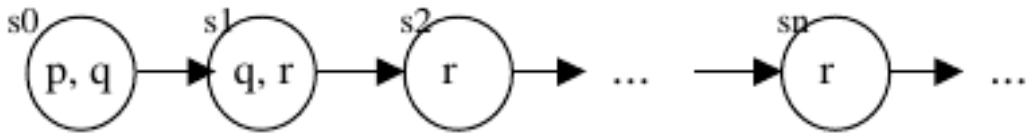
- Fp — “Я когда-нибудь обязательно полюблю Машу”;
- Gq — “Я люблю Дашу и буду любить ее всегда”;
- $G(\neg p \wedge \neg q)$ — “Я однолюб, и никогда не буду любить и Машу, и Дашу одновременно”;
- qUp — “В будущем я полюблю Машу, а до той поры я буду любить Дашу”;
- FGp — “Когда-нибудь я полюблю Машу навечно”;
- GFq — “Я буду бесконечно влюбляться в Дашу”.

TL и анализ дискретных технических систем

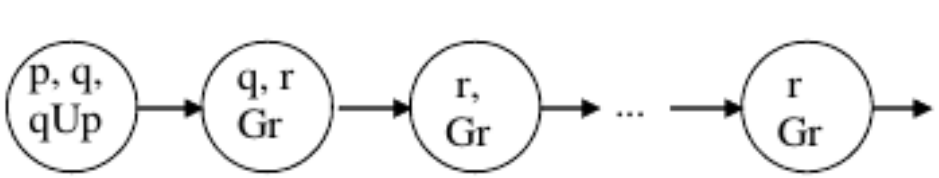
Последовательность "*миров*" в TL можно толковать как бесконечную последовательность состояний дискретной системы, а отношение достижимости – как дискретные переходы системы:



Атомарные предикаты - базисные свойства процесса в состояниях:



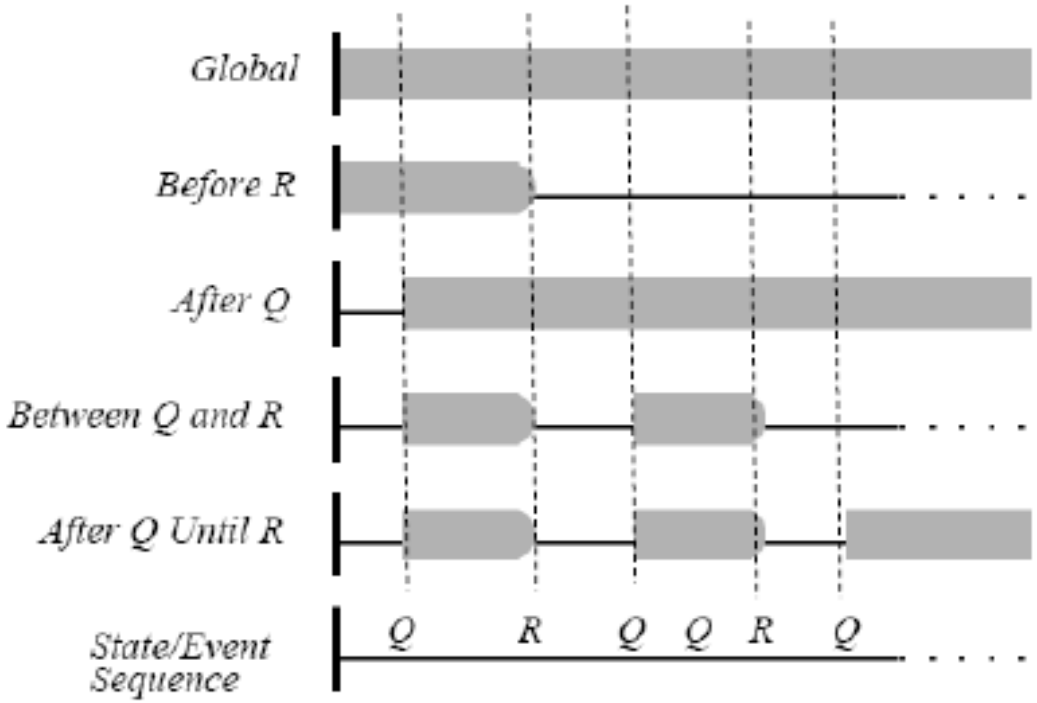
Производные темпоральные формулы в состояниях – это свойства вычисления в будущем, динамики процесса:



qUp – выполняется на вычислении

Gr – не выполняется на вычислении

Шаблоны выполнения темпоральных свойств в технических системах



$G\varphi$

$FR \Rightarrow \varphi UR$

$G\,(Q \Rightarrow G\varphi)$

$G((Q \wedge XFR) \Rightarrow (\varphi \wedge X(\varphi\ U\ R)))$

$G(Q \Rightarrow \varphi \wedge X(\varphi UR \vee G(\neg R \wedge \varphi))$

Набор упражнений 2

Запишите следующие утверждения с помощью нотации LTL:

1. “q встретится в будущем точно один раз”.
2. “всегда если запрос p будет подан, реакция q на него обязательно будет получена, а до ее получения запрос p не сбросится”.
3. “если событие q наступит, то до его наступления событие p не наступит”.
4. “предикаты p и q выполняются попеременно (т.е. после p не встретится p, пока не встретится q)”.

Другая нотация формул LTL (используется на практике)

Иногда вместо значков F, G, X пишут $\langle\rangle$,[] и O. В этой нотации формула

$$\mathsf{Gp} \Rightarrow \mathsf{FrXq}$$

может выглядеть так

$$[]\mathsf{p} \Rightarrow \langle\rangle \mathsf{rOq}$$

Q&A

Вопросы?