Инженерный подход к разработке SW/HW

Лекция 1

План

- История метода model-checking
- Основные направления
- Основные теории
- Популярные формализмы и инструменты
- Основные области применения

История

- Метод довольно новый, бурное развитие и практическое применение примерно с середины 90-х
- Основоположниками считаются Edmund Clarke, Allen Emerson, Joseph Sifakis
- Возник благодаря предшествующим работам в области темпоральных логик (основоположником в области темпоральных логик считается Amir Pnueli)
- Прорывное развитие стало возможным благодаря двум моментам: появление всё более производительной вычислительной техники (для explicit-state model-checkers) и появление методов работы с большими логическими формулами (ROBDD для symbolic model-checkers)

Amir Pnueli (1941-2009)

- Израильско-американский математаик
- Разработал LTL логику (темпоральную логику линейного времени) в 1977 году
- Изучал применение LTL логики для выражения свойств программ и их анализа
- Считается основоположником темпоральных логик
- 1996 год – премия Тьюринга за разработку и исследования темпоральных логик и их применение в верификации систем
- Подробнее биографию можно посмотреть тут: https://cs.nyu.edu/cs/faculty/pnueli/shrtbio.html
- Список трудов тут: https://cs.nyu.edu/cs/faculty/pnueli/c-and-j.html

Edmund Clarke (1945 – н.в.)

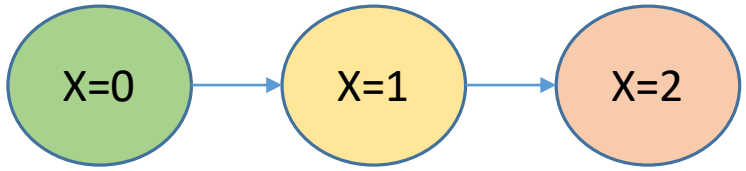
- Американский математатик
- Основоположник CTL логики (первые работы в 1980 году)
- Со своим учеником Allen Emerson предложил метод model-checking – проверка выполнимости формул темпоральной логики на структурах Крипке.
- Премия Тьюринга в 2007
- Персональная страничка: https://www.cs.cmu.edu/~emc/
- Избранные публикации: https://www.cs.cmu.edu/~emc/publications.html

Leslie Lamport (1941- н.в.)

- Американский математик
- Хоть и не является основоположником метода model-checking, тем не менее заслуживает упоминания за выдающиеся труды
- Разработал логику TLA+ (temporal logic of actions)
- Автор множества алгоритмов для распределённых систем. Например Paxos
- В 2013 году получил премию Тьюринга
- Личная страничка: http://www.lamport.org/
- Избранные работы: http://lamport.azurewebsites.net/pubs/pubs.html

Немного терминологии

- Состояние системы – в случае с программой – это совокупность значений всех переменных, содержимого памяти, регистров процессора и тд. на определённый момент времени.
- Процесс – это последовательная цепочка состояний системы.
- Например:
  - for(int x =0 ; x<3; ++x) { … } при выполнении (в процессе) даст следующие состояния:



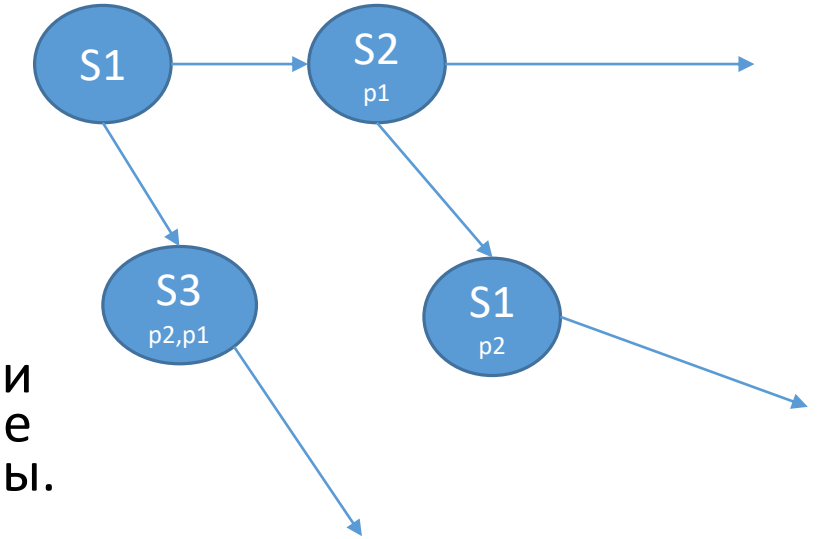
эту последовательность можно назвать процессом вычислений

Немного терминологии

Пространство состояний системы – это множество всех возможных состояний системы.

Граф состояний системы с переходами (state-transition graph) – состояния + переходы между ними

Формально обычно определяется так:
S – множество состояний
T ∈ SxS – множество переходов между состояниями



Каждому состоянию можно поставить в соответствие множество меток-предикатов которые истинны в этом состоянии:
L – метки и Labeling: S -> (2^L) – функция размеки которая кажому состоянию ставит в соответствие множество меток - предикатов, которые истинны.

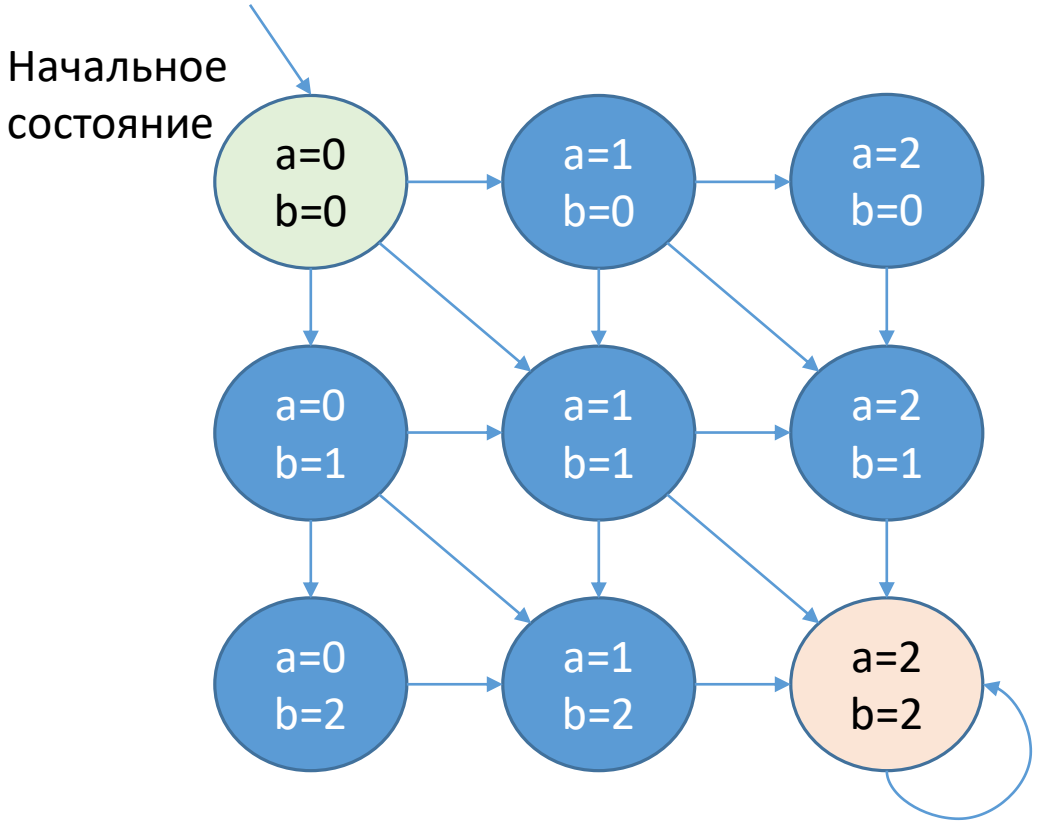
Всё это вместе даёт labelled transition system (размеченную систему переходов). Именно labelled transition system является той моделью на которой проверяется выполнимость формул model-checker’ами.

Пример системы переходов (и пространства состояний)

Process1: for(int a = 0; a<3; ++a) { }

Process2: for(int b = 0; b<3; ++b) { }

Процессы стартуют одновременно и выполняются параллельно, останавливаются после выполнения цикла

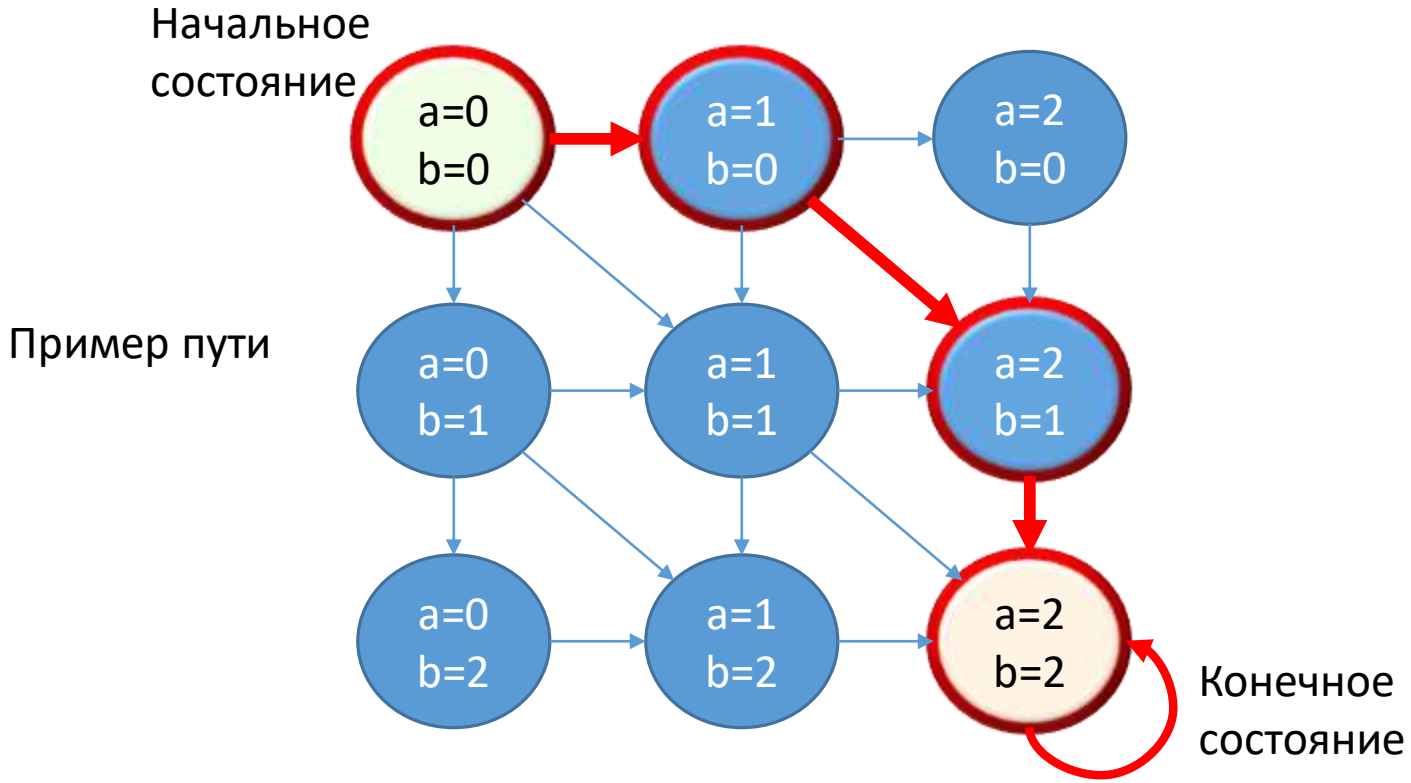


Процесс – это путь из начального в конечное состояние.

Как можем видеть, в пространстве состояний содержатся все возможные процессы в системе.

Траектория системы (процесс, путь)

- Процесс, путь в пространстве состояний, траектория системы – это последовательная цепочка из состояний в пространстве состояний системы.



Основные направления

- Explicit model-checking
- Symbolic model-checking
- Probabilistic model-checking
- Model-finding

Explicit state model-checking

- Основан на явном “переборе” пространства состояний.
- Высокая точность, так как в состоянии учитываются конкретные значения переменных состояния
- Можно проверять разнообразные свойства
- Можно проверять подмножество пространства состояний
- Из минусов: ограничения по размеру пространства состояний, около 10 в 8-10 степени
- Часто используется для анализа алгоритмов и протоколов
- Основные представители:
  - SPIN/PROMELA: http://spinroot.com/spin/Man/index.html, https://en.wikipedia.org/wiki/Promela
  - TLA+/TLC: https://en.wikipedia.org/wiki/TLA%2B

Symbolic model-checking

- Проверка пространства состояний в символьном виде, то есть за один шаг проверяется сразу целое множество состояний
- В основе – ROBDD (https://en.wikipedia.org/wiki/Binary_decision_diagram), бинарная диаграмма решений – компактный способ представления очень больших логических формул. Логические формулы с десятками тысяч и миллионами переменных могут быть анализированы на обычных компьютерах
- Можно анализировать системы с огромными пространствами состояний 10 в 200 – не предел
- Из минусов: Сложнее алгоритмы. Меньше предсказуемость по ресурсам. Может на простой модели и формуле, на которой отработает explicit model-checker, исчерпать ресурсы машины и не справиться с задачей (это бывает крайне редко из-за неудачного построения ROBDD)
- Активно применяется для анализа цифровых схем
- Основные представители:
  - NuSmv: http://nusmv.fbk.eu/
  - muCLR2: https://www.mcrl2.org/web/user_manual/index.html

Probabilistic model-checking

- К дугам переходов между состояниями системы добавлены вероятности перехода
- Можно получать не только качественные ответы “да”/”нет” относительно проверяемых свойств системы, но и количественные, например: “свойство P будет выполнено с вероятностью 1%”, “99% запросов к серверу будут обработаны без ошибок” и т.д.
- Из минусов: высокая сложность алгоритмов, проверемые пространства состояний (сложность системы) существенно меньше, чем у symbolic model-checkers.
- Заметные представители:
  - PRISM: https://www.prismmodelchecker.org/
  - STORM: https://www.stormchecker.org/

Model-finders

- Решают немного другую задачу – не проверяют выполнимости формул на моделях, а ищут модели, которые удовлетворяют заданным формулам.
- Очень полезны при анализе структур и конфигураций систем и операций над ними (можно анализировать широкий спектр систем: от структур данных до сетевых топологий)
- Отлично дополняют model-checkers
- Самый известный представитель:
  - Alloy analyzer: https://alloytools.org/

Основные теории

- Темпоральные логики:
  - CTL*, CLT, LTL (в книге Юрия Карпова “Model-checking” очень доступное введение в темпоральные логики)
  - Немного особняком стоит TLA+: temporal logic of actions
- Двоичные решающие диаграммы: ROBDD. Это основа алгоритмов symbolic model-checkers. (хороший курс по матлогике и BDD тут: https://ru.coursera.org/learn/matematicheskaya-logika-politekhnicheskiy-vzglyad?action=enroll)
- Матлогика и теория моделей: https://en.wikipedia.org/wiki/Model_theory
- Mu-calculus.
- В рамках этого курса углубляться в теорию не будем, если кому интересно будет, может самостоятельно посмотреть основы по приведённым ссылкам. Так же буду рад помочь заинтересованным студендам и могу порекомендовать разную литературу и другие материалы для углублённого изучения.

Немного определений

- Формализмом в математике называют подход к решению математических проблем с использованием формальных систем (https://en.wikipedia.org/wiki/Formalism_(philosophy_of_mathematics))
- Формальная система – это формальный язык + семантика
- Формальный язык – это язык который задаётся точными грамматическими правилами. Например, возьмём алфавит A = {a,b}, и набор правил для построения выражений языка G = A|B; A=a|aA; B=b|bA, тогда наш формальный язык состоит из фраз: “aaaaa”, “ba”, “b”, “baaaaa”, “a” и тд.
- Семантика – это правила, которые позволяют поставить в соответствие выражениям на формальном языке некоторые математические объекты. Грубо говоря, наделяют фразы формального языка неким смыслом.
- Пример формальной системы – математическая логика, например, исчисление высказываний.

Немного определений

- Под формализмом в данном случае так же понимается конкретная формальная система (язык + семантика), которая используется для описания моделей и проверяемых свойств.

Основные формализмы

- TLA, Temporal Logic of Actions
- First-order relational logic with closure operator (используется как основа в Alloy Analyzer)
- CTL*, CTL, LTL
- PCTL – Probabilistic CTL
- DTMC - Discrete-time Markov chains
- CTMC - Continuous-time Markov chains
- MDP - Discrete-time Markov decision processes
- Petri Nets
- GSPN – Generalized Stochastic Petri Nets

Некоторые популярные инструменты

- Как правило в основе конкретного инструмента лежит тот или иной (или сразу несколько) формализм.
- Поверх этого формализма часто разрабатывается специализированный язык для упрощения описания проверяемых моделей и свойств.
- Примеры таких инструментов:
  - TLA: языки TLA+, PlusCal, model-checker: TLC, среда разработки TLA-Toolbox (https://lamport.azurewebsites.net/tla/toolbox.html)
  - LTL + CSP: язык PROMELA, model-checker SPIN, среда разработки SPIN/PROMELA (http://spinroot.com/spin/whatispin.html)
  - PCTL + DTMC + CTMC + MDP: язык PRISM, model-checker PRISM или STORM, скачать можно тут: https://www.prismmodelchecker.org/, https://www.stormchecker.org/
  - CTL* + ROBDD + SMT: NuSMV, http://nusmv.fbk.eu/
  - Mu-Calculus + … : muCRL2, https://www.mcrl2.org//web/user_manual/index.html
  - RelFOL: Alloy language, Alloy Analyzer : https://alloytools.org/

Основные области применения model-checking

- Проверка спецификаций аппаратного и програмного обеспечений:
  - Анализ логики в процессорах
  - Анализ управляющей логики в промышленных контроллерах
- Проверка свойств протоколов:
  - Протокол когерентности кэшей, без него не было бы многоядерных процессоров
  - Анализ протоколов обмена данными
- Разработка и анализ алгоритмов, особенно распределённых:
  - Например, разные алгоритмы консенсуса, без чего невозможны современные базы данных в распределённых конфигурациях
  - Алгоритмы управления и синхронизации в облаках, например системный софт Amazon Web Services
  - DHT алгоритмы, торрентами же все пользуются?
- Разработка систем высокой надёжности с гарантированными свойствами:
  - ПО и аппаратура для самолётов, ракет и пр.
  - ПО и аппаратура управления промышленными системами: электростанции, силовые агрегаты (турбины, двигатели и пр)
  - ПО и аппаратура в медицине, системы жизнеобеспечения и пр.

Пример формализации задачи и свойств системы

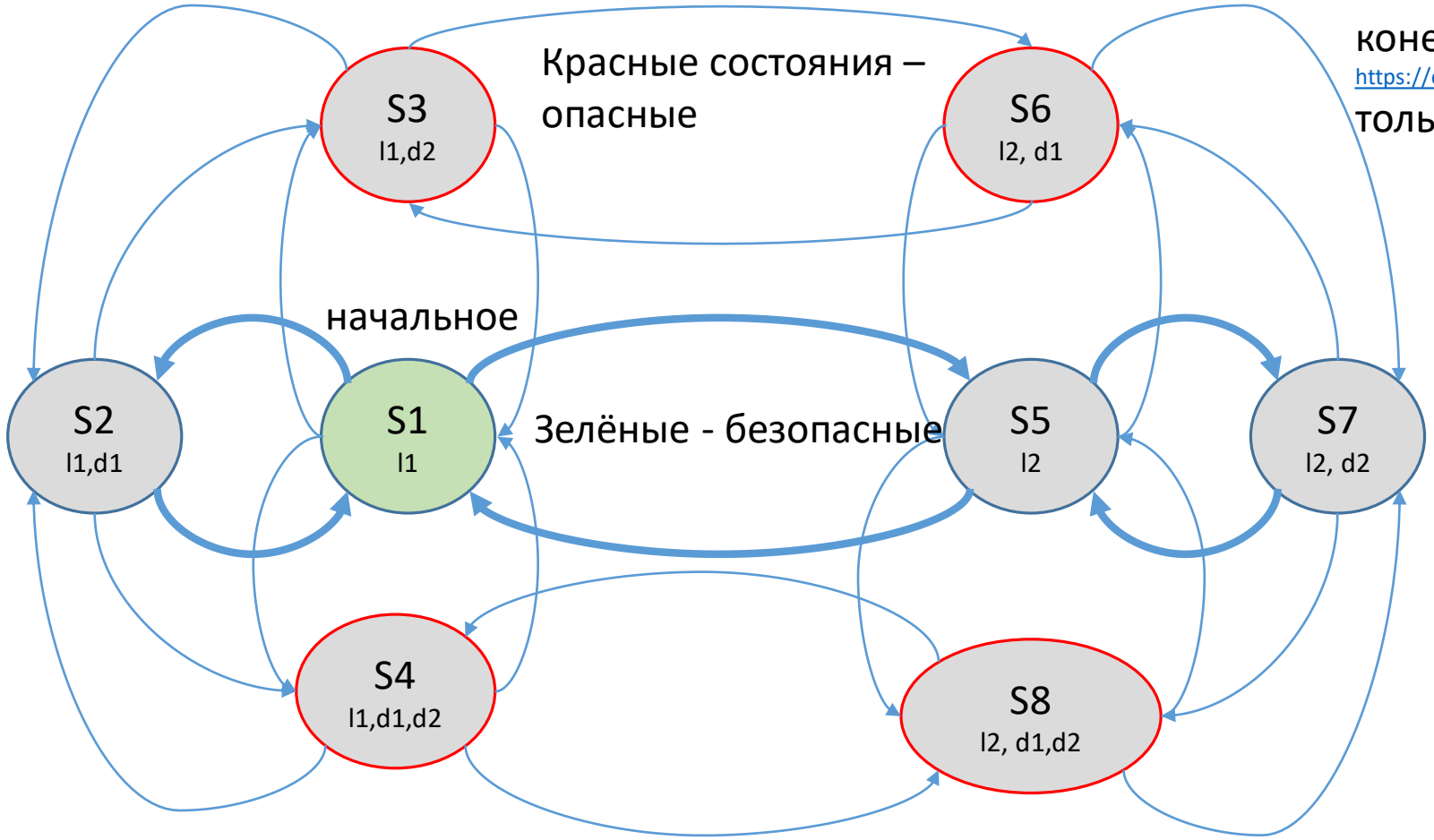
- Лифт в двухэтажном доме
- Состояния системы:

Состояние	На каком этаже лифт	На каком этаже открыты двери
S1	1	
S2	1	1
S3	1	2
S4	1	1,2
S5	2	
S6	2	1
S7	2	2
S8	2	1,2

- Предикаты:
  l1 – лифт на первом
  l2 – на втором
  d1 – двери открыты на первом этаже
  d2 – на втором

Структура Крипке (одна из возможных)

По сути – недеретминированный конечный автомат
https://en.wikipedia.org/wiki/Nondeterministic_finite_automaton
только без меток на дугах

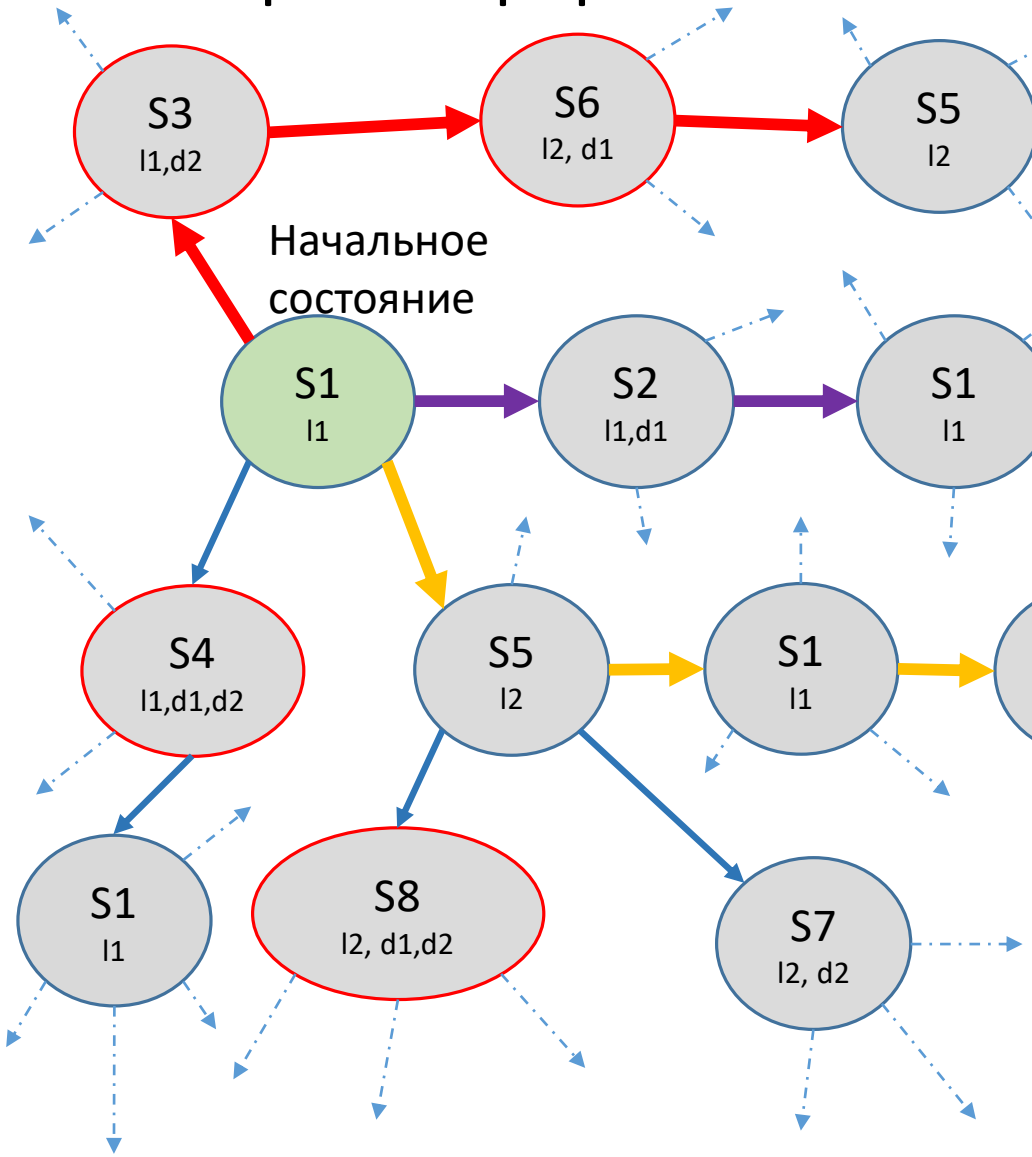


Введена Солом Крипке – американским логиком (https://en.wikipedia.org/wiki/Saul_Kripke) – в рамках работ над модальными логиками, используется в model-checking, как модель системы

Развёртка структуры Крипке

- Это бесконечное дерево состояний системы и переходами между ними.
- Пути в этом дереве – это возможные процессы в системе.
- Все возможные процессы в системе называются поведением системы (то есть под поведением системы можно подразумевать развёртку структуры Крипке и все возможные пути в ней)

Пример развётки структуры Крипке



Опасное поведение, есть вероятность падения в шахту лифта, либо быть раздавленным лифтом

Лифт стоит на 1ом этаже и хлопает дверями, безопасно и бесполезно

Лифт катается между 1ым и 2ым этажами и не открывает двери, безопасно и бесполезно

Как можно видеть, в этой развёртке структуры Крипке есть как опасные поведения лифта, так и безопасные, как полезные, так и бесполезные.

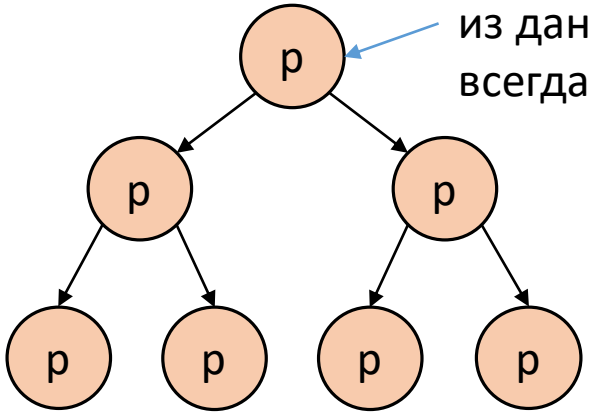
С помощью model-checking можно проверить систему, что все её траектории как минимум безопасные, а как максимум безопасные и полезные.

Формулировка свойств системы CTL*, CTL, LTL

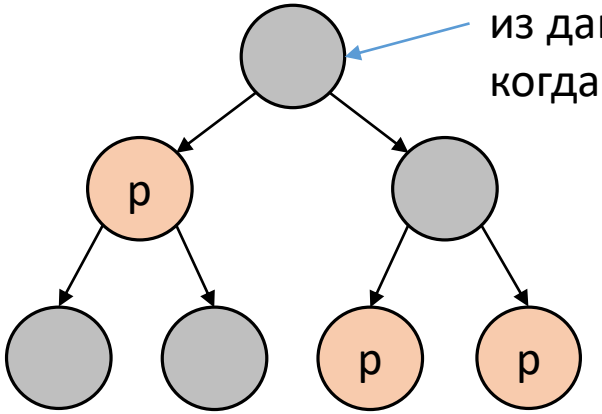
- CTL – формулы на поддеревьях развёртки структуры Крипке
- LTL – формулы на путях в развёртке структуры Крипке
- CTL* - CTL + LTL
- CTL кванторы:
  - A (f) – для всех путей начинающихся в данной вершине выполняется f
  - E (f) – существует путь из данной вершины, где выполняется f
  - Кванторы путей из LTL могут использоваться, только в связке с предваряющим A или E: AF, AG, EG, EF.
- LTL кванторы:
  - G (f) – всегда на цепочке состояний выполняется f
  - F (f) – когда-нибудь в будущем на цепочке состояний выполнится f
  - X (f) – на следующем состоянии выполнится f

CTL наглядно

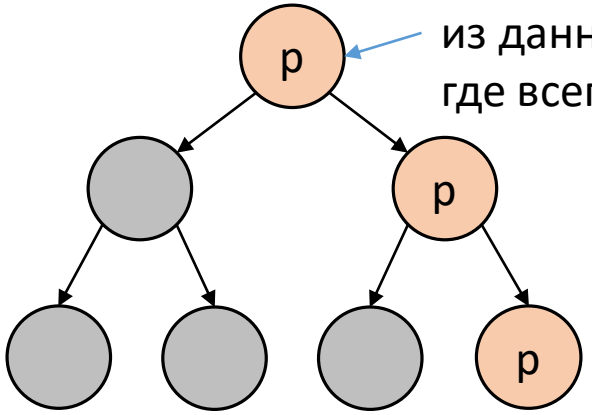
AG(p) – на всех путях
из данной вершины
всегда выполняется p



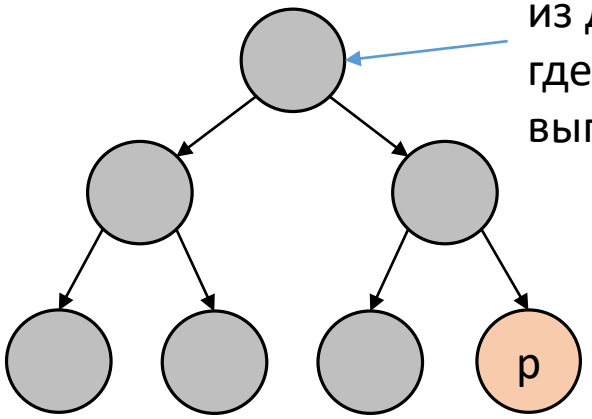
AF(p) – на всех путях
из данной вершины
когда-нибудь выполнится p



EG(p) – есть путь
из данной вершины
где всегда выполняется p



EF(p) – есть путь
из данной вершины
где когда-нибудь
выполняется p



Сформулируем свойство безопасности лифта в логике CTL

- Всегда должно выполнятся: двери могут быть открыты только на том этаже, где находится лифт.
- AG((l1 /\ d1 /\ not(d2)) or (l2 /\ d2 /\ not(d1)) or (l1 /\ not(d1) /\ not(d2)) or (l2 /\ not(d1) /\ not(d2))):
  - ‘AG’ – всегда для всех поведений лифта должно выполнятся:
  - ‘l1 /\ d1 /\ not(d2)’ – лифт на первом этаже и двери на первом этаже открыты, а на втором закрыты, или
  - ‘l2 /\ d2 /\ not(d1)’ – лифт на втором этаже, двери на первом закрыты, на втором – открыты, или
  - ‘l1 /\ not(d1) /\ not(d2)’ – лифт на первом этаже и двери на всех этажах закрыты, или
  - ‘l2 /\ not(d1) /\ not(d2)’ – лифт на втором этаже и двери на всех этажах закрыты

Инструменты, которые будем использовать в рамках текущего курса

- TLA-Toolbox: https://lamport.azurewebsites.net/tla/toolbox.html
- Alloy Analyzer: https://github.com/AlloyTools/org.alloytools.alloy/releases