Верификация – подтверждение на основе представления объективных свидетельств того, что требования, предназначенные для конкретного использования или применения, выполнены.

Валидация в контексте жизненного цикла системы является совокупностью действий, гарантирующих и обеспечивающих уверенность в том, что система способно реализовать своё предназначение, текущие и перспективные цели.



Задачи верификации в рамках ЖЦ ПО:

1. Выявление дефектов
2. Выявление наиболее критичных и наиболее подверженных ошибкам частей создаваемой или сопровождаемой системы.
3. Контроль и оценка качества ПО во всех его аспектах.
4. Предоставление всем заинтересованным лицам информации о текущем состоянии проекта и характеристиках его результатов.
5. Предоставление руководству проекта и разработчикам информации для планирования дальнейших работ, а также для принятия решений о продолжении проекта, его прекращении или передачи результатов заказчику.

К верификации имеют отношения 5 процессов:

1. Обеспечение качества.
2. Собственно верификация,
3. Валидация
4. Совместные экспертизы
5. Аудит

Схема классификации методов верификации.



Экспертиза позволяет выявить практически любые виды ошибок, причём делать это на этапе подготовки соответствующего артефакта. Она не может быть автоматизирована и требует активного участия людей. Эффективность её зависит от опыта и мотивации участников, организации процесса, а также обеспечения корректного взаимодействия между различными участниками.

Статический анализ свойств артефактов жизненного цикла ПО используется для проверки формализованных правил корректного построения этих артефактов и поиска часто встречающихся дефектов по некоторым шаблонам. Такой анализ хорошо автоматизируется и может быть практически полностью возложен на инструменты, хотя иногда необходимо вручную определить, например, принятые в проекте стандарты кодирования. Однако применим он лишь к коду или к определенным форматам представления проектных артефактов, и способен обнаруживать только ограниченный набор типов ошибок.

Формальные методы верификации используют для анализа свойств ПО формальные модели требований, поведения ПО и его окружения. Анализ формальных моделей выполняется с помощью специфических техник, таких как дедуктивный анализ (theorem proving), проверка моделей (model checking) или абстрактная интерпретация (abstract interpretation).

Динамические методы верификации, в рамках которых анализ и оценка свойств программной системы делаются по результатам ее реальной работы или работы некоторых ее моделей и прототипов. Примерами такого рода методов являются обычное тестирование или имитационное тестирование, мониторинг, профилирование.

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 (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 (∧, ∨...);

- исчисление предикатов (predicate calculus): + кванторы по объектным переменным (∀, ∃);

- 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, у должно стать равно 0», «после того, как x станет больше 0, спустя некоторое время y обязательно станет равно 5»

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 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.

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

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

, где P – некоторая программа. Утверждение [P]Φ означает, что всегда после выполнения программы P формула Φ истинна, а

Φ – что после выполнения P Φ может оказаться истинной. Тройка Хоара ΦPΨ может быть представлена в динамической логике как Φ ⇒ [P]Ψ.

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.

1. 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 ` 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.
2. 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 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.
3. 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.

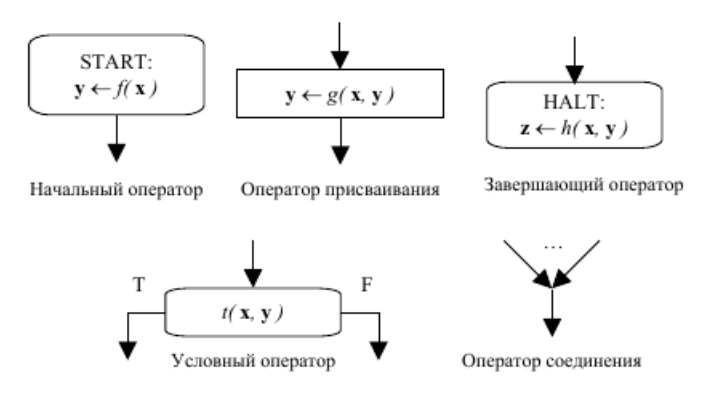
В основе этих методов лежит логика Хоара и предложенная Флойдом техника доказательства завершения циклов, основанная на инвариантах цикла и монотонно изменяющихся в ходе его выполнения оценочных функциях.

Методы Флойда-Хоара.

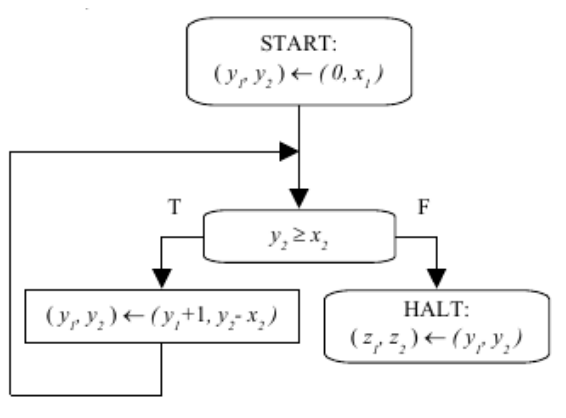
1. Спецификация программы в виде ее предусловия и постусловия определяется формально, например, в рамках исчисления высказываний.
2. В коде программы или на ее блок-схеме выбираются точки сечения, так чтобы любой цикл содержал по крайней мере одну такую точку. Начало и конец программы (все возможные точки выхода из программы можно свести к одной) тоже считаются точками сечения.
3. Для каждой точки сечения i находится предикат Φi , 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). 20 /
4. В результате программа разбивается на набор возможных линейных путей между парами точек сечения. Для каждого такого пути Pij между точками i и j нужно проверить истинность тройки ΦiPijΦj . 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 Pij , для которого можно доказать {Φi & ϕ = a}Pij{Φj & ϕ < 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.

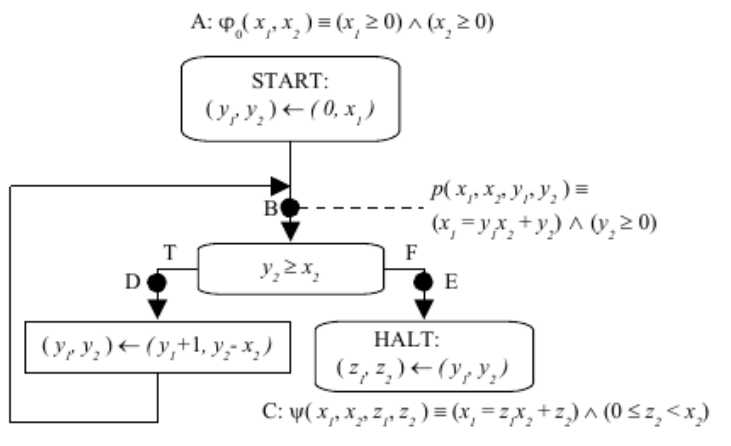
Каждая программа работает с конечным числом переменных. Переменные разделяются на три типа: входные, промежуточные и выходные.Каждая переменная v может принимать значения из некоторого множества Dv, которое называется доменом переменной. Основные операторы программы:



Блок-схема целочисленного деления.



Целочисленное деление с предикатами.



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.

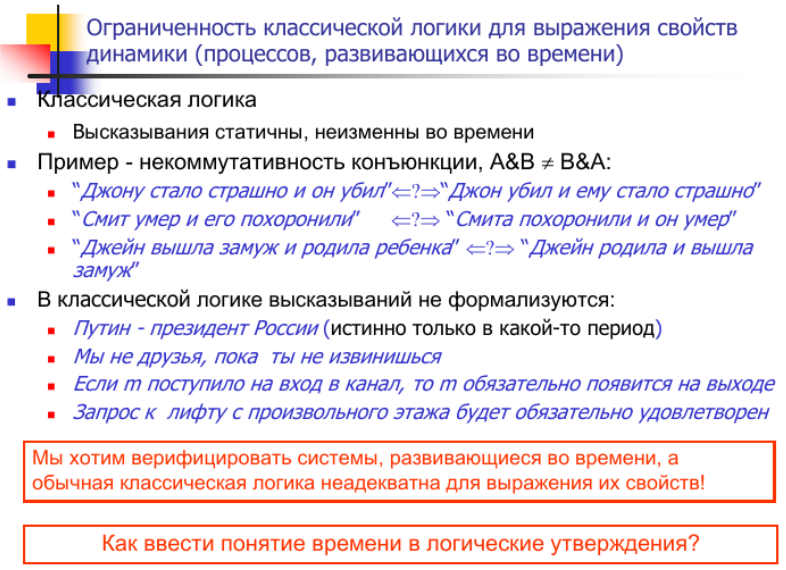
Дедуктивный анализ, инструменты.

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.

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.



Использование модальностей:

Fq – случится когда-нибудь в будущем.

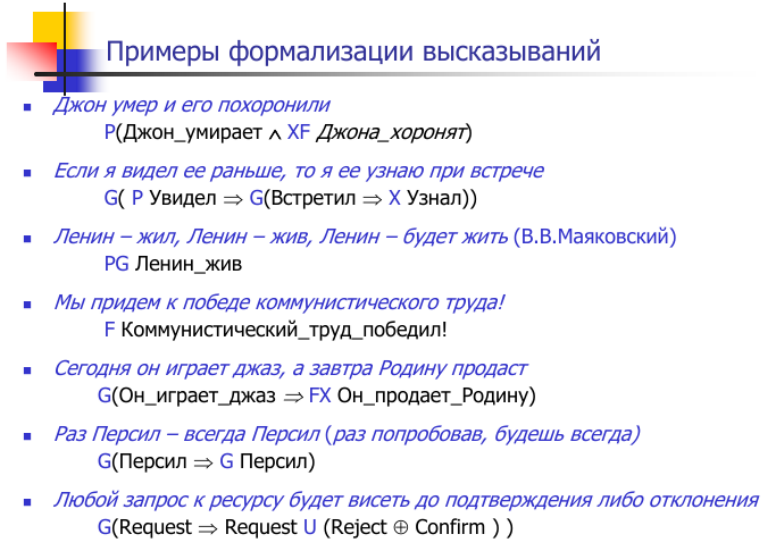
Pq – случилось когда-то в прошлом.

Gq – всегда будет в будущем.

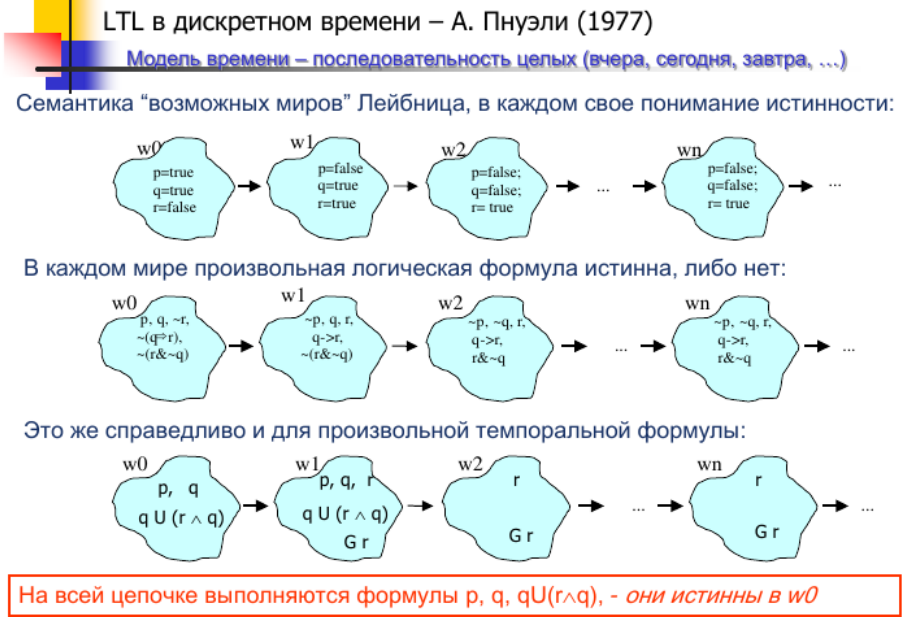
Hq – всегда было в прошлом.

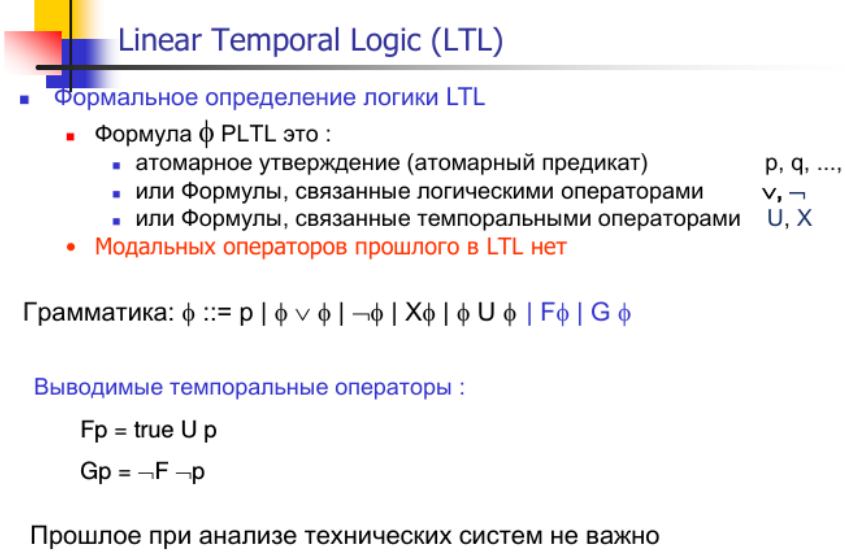
pUq - q случится в будущем, а до него непрерывно будет выполняться p.

Хp – выполнится в следующий момент.



В темпоральной логике иногда используется оператор W, так называемый “слабый Until” (Unless), имеющий следующую семантику: p W q = p U q ∨ 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.





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

Fp — “Я когда-нибудь обязательно полюблю Машу”;

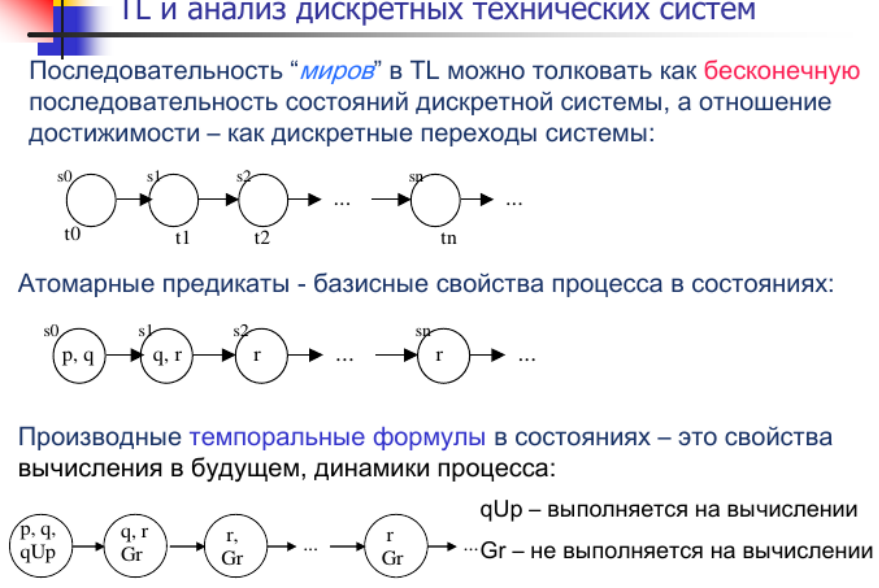
Gq — “Я люблю Дашу и буду любить ее всегда”;

G(¬p ∧ ¬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”.





Иногда вместо значков F, G, X пишут hi,[] и O. В этой нотации формула Gp ⇒FrXq может выглядеть так []p ⇒hirOq

Реагирующие системы (reactive systems) это класс информационных систем, основной функцией которых является поддержание взаимодействия с окружением, а не преобразование информации.

Вычисление и поведение реагирующей системы Вычисление реагирующей системы — это бесконечная последовательность состояний, которые система проходит во времени. Поведение реагирующей системы — это все возможные ее вычисления.

Математическая модель для исследования поведения реагирующих систем По заданному вычислению — бесконечной цепочке состояний с определенным в каждом состоянии набором атомарных предикатов, истинных в этом состоянии, определяются значения истинности булевых и темпоральных формул.

Даже если система переходов имеет конечное число состояний, в общем случае количество возможных вычислений в ней бесконечно и каждое вычисление также бесконечно.

Логика LTL: линейный мир — после сегодняшнего дня будет завтра, потом послезавтра ... линейная последовательность дискретных возрастающих значений. Поведения реальных систем (информационных в том числе) имеют альтернативы, выбор которых осуществляется на основе внешних событий, различий в содержании принятых сообщений и т.п.

Модель системы переходов с конечным числом состояний.

Структура Крипке M — это пятерка M = (S, S0, R, AP, L), где: S — конечное непустое множестов состояний; S0 ⊆ S — непустое множество начальных состояний; R ⊆ S × S — тотальное отношение на S, т.е. множество переходов, удовлетворяющих требованию: (∀s ∈ S)(∃s 0 ∈ S)(s,s 0 ) ∈ R; AP — конечное множество атомарных предикатов; L : S → 2 AP — функция пометок (каждому состоянию отображение L сопоставляет множество истинных в нем атомарных предикатов).

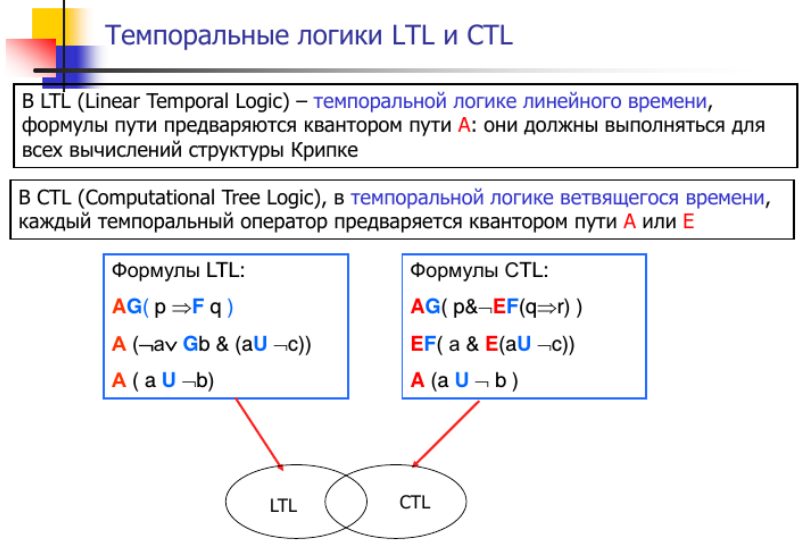
Вычислением структуры Крипке M называется любая бесконечная цепочка σ = q0q1q2q3..., такая что q0 ∈ S0 и (qi , qi+1) ∈ R. Формально вычисление структуры Крипке M можно представить как отображение σ : N → S, где N — множество натуральных чисел, т.е. σ(i) — это некоторое состояние структуры Крипке.

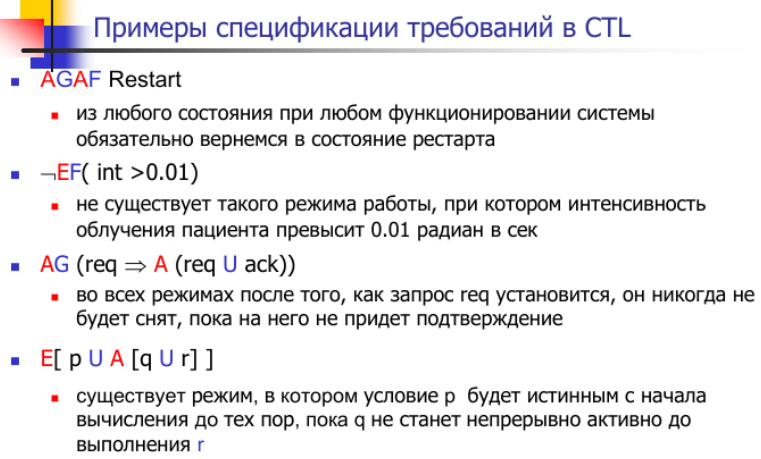
Траекторией структуры Крипке M, индуцированной вычислением σ = q0q1q2q3..., называется бесконечная цепочка L(σ) = L(q0)L(q1)L(q2)L(q3)..., т.е. бесконечная цепочка подмножеств атомарных предикатов, истинных в соответствующих состояниях вычисления σ. Формально траектория — это отображение натурального ряда в множество 2AP

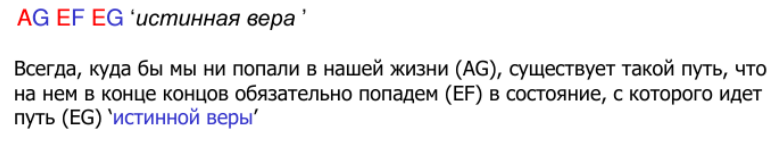
Траектория: наблюдаемые свойства, которые могут выполниться в анализируемых системах. Формулы темпоральной логики описывают свойства траекторий.

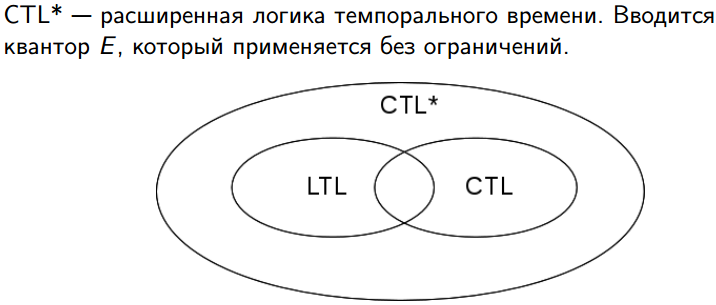
Множество всех ω-слов структуры Крипке M называется ω-языком, допускаемым M.

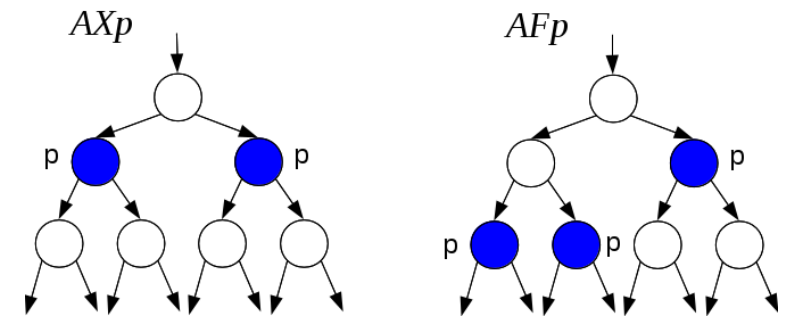


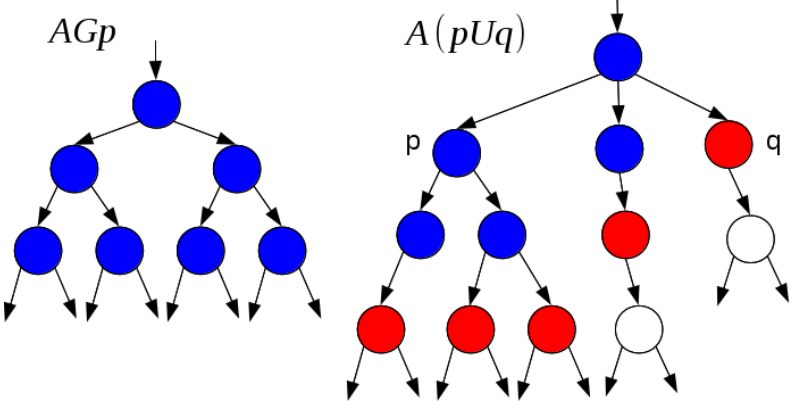


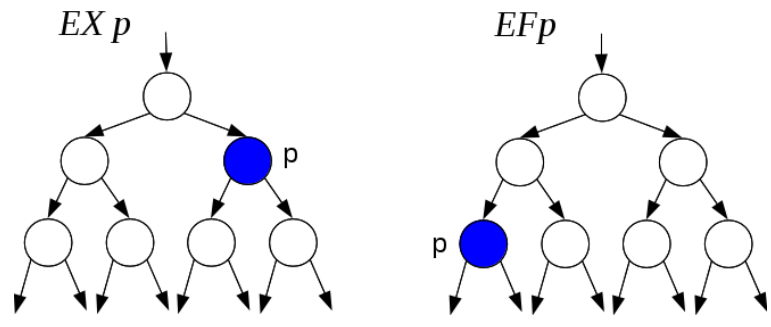


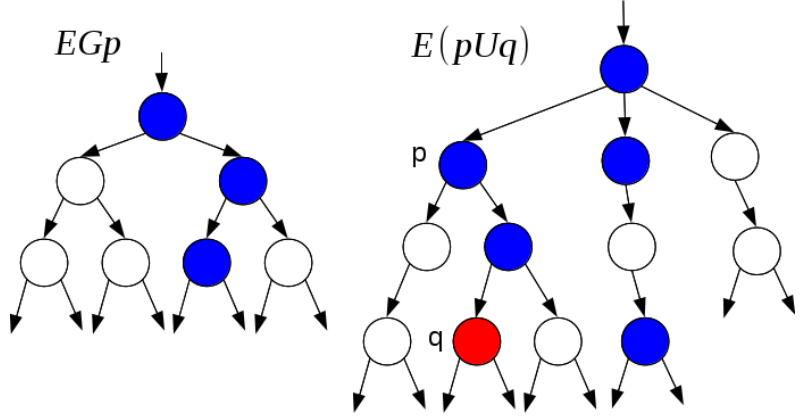












Пусть p означает “Я люблю Машу”, q означает “Я люблю Дашу”. Тогда:

1 формула AGp является формулой CTL, она формализует следующее высказывание: “Я люблю Машу, и, что бы ни случилось, я буду любить ее всегда”;

2 формула AFG(p ∧ ¬q) является формулой LTL, она формализует следующее высказывание “Что бы ни случилось, в будущем я полюблю Машу навсегда, а Дашу не буду любить!”;

3 формула E(p ∪ (AGq)) является формулой СTL, она формализует следующее высказывание “Я не исключаю такого развития событий (E), что я буду любить Машу до тех пор (U), пока, наконец я не полюблю Дашу навечно (AG)”.

Сравнение логик CTL и LTL

Формулы этих логик характеризуют свойства разных типов: формулы LTL являются формулами пути; в логике LTL вообще нет кванторов пути; формулы CTL являются формулами состояний; в логике CTL любой темпоральный оператор предварен квантором пути.

Формулы обеих логик интерпретируются (принимают истину или ложь) на структурах Крипке, но: формулы логики LTL интерпретируются на всех вычислениях структуры Крипке, начинающихся в начальном состоянии (большие трудности при проверке); формулы логики CTL интерпретируются на деревьях вычислений структуры Крипке (конечное число состояний).

Техника верификации для LTL и CTL совершенно различна как по применяемым алгоритмам, так и по их сложности: для LTL — Spin. Сложность алгоритмов верификации линейна относительно числа состояний стр-ры Крипке и экспоненциальна относительно сложности (числа подформул) формулы LTL (но на практике они малы); для CTL — SMV. Сложность алгоритмов верификации пропорциональна числу состояний структуры Крипке и сложности (числу подформул) формулы CTL.

Многие свойства технических систем могут быть выражены как формулами логики CTL, так и формулами логики LTL. Например, “каждое сообщение когда-нибудь в будущем будет подтверждено”: для LTL — G(send ⇒ Freq); для CTL — AG(send ⇒ AFreq). Однако неверно утверждение, что если в формуле LTL каждый темпоральный оператор будет предварен квантором пути A, то построенная так формула CTL будет эквивалентна исходной формуле.

Выразительная мощность этих двух логик несравнима, т.е. некоторые свойства могут быть выражены в CTL, но не могут быть выражены в LTL и наоборот: например, формула AGEFφ логики CTL, выражающая “достижимость свойства φ”, не выразима в LTL; например, формула FGφ логики LTL, выражающая “стабилизацию свойства φ когда-нибудь в будущем”, не выразима в CTL.

