Математическая логика

Лектор:

Подымов Владислав Васильевич

e-mail:
valdus@yandex.ru

2017, весенний семестр

Лекция 16–17

Формальная верификация

Императивные программы

Логика Хоара

Автоматическая проверка правильности программ

Верификация распределённых систем

Лекция 16–17

Модальные логики

Логика линейного времени

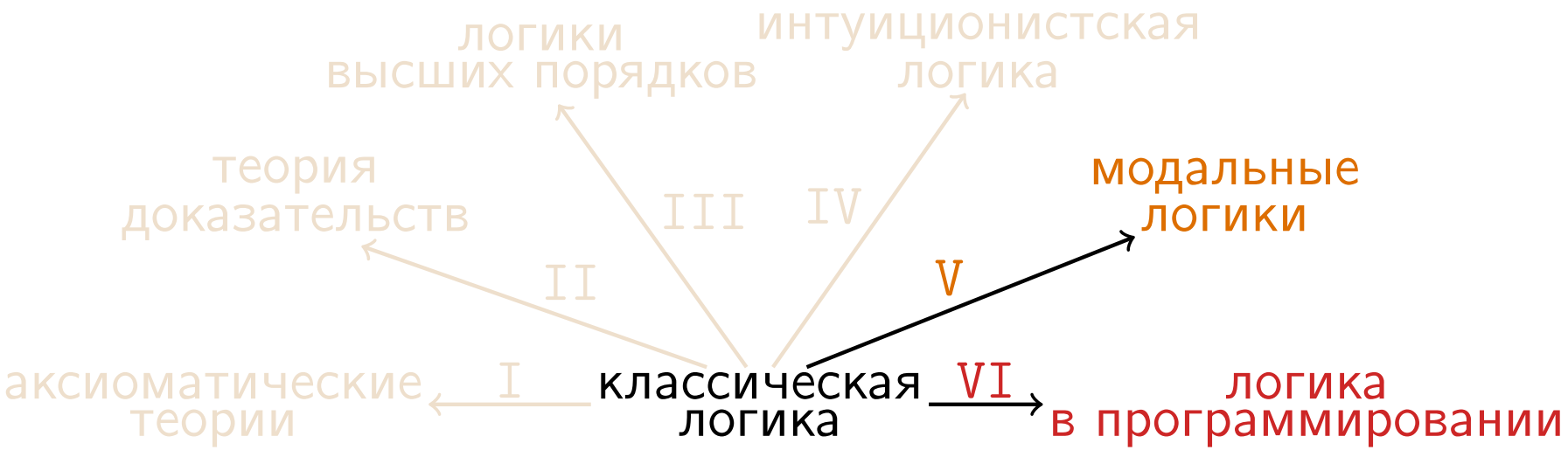
Логика деревьев вычислений

Размеченные системы переходов

Задача проверки моделей

Алгоритм проверки моделей
для логики деревьев вычислений

Что ещё есть в логике



✓	I	Специальные интерпретации логических формул
✓	II	Другие формы логического вывода
—	III	Более широкие возможности применения кванторов
—	IV	Другая семантика логических операций
	V	Другие логические операции
	VI	Логика в приложении к другим дисциплинам
	...	...

Формальная верификация

“Любая нетривиальная программа
содержит хотя бы одну ошибку”

автор неизвестен

Правильная программа ошибок не содержит и делает в точности то, что от неё требуется

А как убедиться, что программа написана правильно?

Например, так:

1. **строго** сформулировать требования правильности работы программы
2. **строго** описать процесс выполнения программы
3. убедиться, что *строго описанные* вычисления программы удовлетворяют *строго сформулированным* требованиям

Формальная верификация

Требования к вычислениям программы, записанные на каком-либо формальном языке, — это формальная спецификация программы

Строгая проверка соблюдения этих требований — это формальная верификация

Если в качестве языка спецификации программ выбрать **логический язык**, то для проверки правильности программы можно будет использовать **логические методы**

Попробуем применить логические методы к проверке правильности **императивных программ**

Императивные программы: синтаксис

Синтаксис императивных программ **сигнатуры** σ:[1]

$$\pi ::= \emptyset \mid \mathrm{x} := t \mid \pi;\ \pi \mid \textbf{if } C \textbf{ then } \pi \textbf{ else } \pi \textbf{ fi} \mid \textbf{while } C \textbf{ do } \pi \textbf{ od}$$

Здесь

- π — программа
- $\emptyset$ — пустая программа
- x — переменная
- t — терм
- C — бесквантерная формула

[1] Все знают, что такое **форма Бэкуса-Наура**?

Императивные программы: семантика

Значение программы — это вычисляемая ей функция преобразования входных данных в выходные

Операционная семантика программы — это способ описания значения программы, согласно которому определяются

- вычисление программы — последовательность **состояний вычисления**, включающих в себя
  - состояние данных — текущие значения всех переменных
  - состояние управления — выполняемая инструкция
- значение каждой инструкции — функция преобразования состояний вычисления
- значение программы на начальном состоянии данных — состояние данных последнего состояния вычисления

Императивные программы: семантика

Состояние управления — это любая программа

Состояние данных (оценка данных) — это подстановка $\theta : \mathrm{Var} \to \mathrm{GTerm}$, где GTerm — множество основных термов

Состояние вычисления — это пара $\langle \pi,\ \theta \rangle$, где π — состояние управления и θ — состояние данных

Один шаг работы программы π в интерпретации $\mathcal{I}$ описывается отношением переходов $\to_{\mathcal{I}}$ на множестве состояний вычисления:

- $\langle \mathtt{x} := t,\ \theta \rangle \to_{\mathcal{I}} \langle \emptyset,\ \{\mathtt{x}/t\}\,\theta \rangle$
- если $\mathcal{I} \models C\theta$, то
$$\langle \textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi},\ \theta \rangle \to_{\mathcal{I}} \langle \pi_1,\ \theta \rangle$$
- если $\mathcal{I} \not\models C\theta$, то
$$\langle \textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi},\ \theta \rangle \to_{\mathcal{I}} \langle \pi_2,\ \theta \rangle$$

Императивные программы: семантика

Состояние управления — это любая программа

Состояние данных (оценка данных) — это подстановка $\theta : \mathrm{Var} \to \mathrm{GTerm}$, где GTerm — множество основных термов

Состояние вычисления — это пара $\langle \pi,\ \theta \rangle$, где π — состояние управления и θ — состояние данных

Один шаг работы программы π в интерпретации $\mathcal{I}$ описывается отношением переходов $\to_{\mathcal{I}}$ на множестве состояний вычисления:

- если $\mathcal{I} \models C\theta$, то
$$\langle \textbf{while } C \textbf{ do } \pi \textbf{ od},\ \theta \rangle \to_{\mathcal{I}} \langle \pi;\ \textbf{while } C \textbf{ do } \pi \textbf{ od},\ \theta \rangle$$
- если $\mathcal{I} \not\models C\theta$, то
$$\langle \textbf{while } C \textbf{ do } \pi \textbf{ od},\ \theta \rangle \to_{\mathcal{I}} \langle \emptyset,\ \theta \rangle$$
- если $\langle \pi_1,\ \theta \rangle \to_{\mathcal{I}} \langle \pi_1',\ \eta \rangle$, то
$$\langle \pi_1;\ \pi_2,\ \theta \rangle \to_{\mathcal{I}} \langle \pi_1';\ \pi_2,\ \eta \rangle$$
- $\langle \emptyset;\ \pi,\ \theta \rangle \to_{\mathcal{I}} \langle \pi,\ \theta \rangle$

Императивные программы: семантика

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

$$\langle \pi,\ \theta \rangle \to_{\mathcal{I}} \langle \pi_1,\ \theta_1 \rangle \to_{\mathcal{I}} \langle \pi_2,\ \theta_2 \rangle \to_{\mathcal{I}} \dots$$

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

Результат конечного вычисления программы π на оценке θ в интерпретации $\mathcal{I}$ — это оценка данных последнего состояния вычисления π на θ в $\mathcal{I}$

Иными словами, если $\langle \pi,\ \theta \rangle \to^*_{\mathcal{I}} \langle \emptyset,\ \eta \rangle$, то η — результат вычисления π на θ в $\mathcal{I}$

($\to^*_{\mathcal{I}}$ — *транзитивное замыкание отношения* $\to_{\mathcal{I}}$)

Императивные программы: семантика

Пример

π: **while** $\mathrm{x} > 0$ **do** $\mathrm{x} := \mathrm{x} - 1$ **od** $\qquad \theta = \{\mathrm{x}/1\}$

$\mathcal{I}$ — целочисленная арифметическая интерпретация

Вычисление π на θ в $\mathcal{I}$:

$$\langle \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$

$$\downarrow_{\mathcal{I}}$$

$$\langle \mathrm{x} := \mathrm{x} - 1;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$

Пояснение:

$$(\mathrm{x} > 0)\{\mathrm{x}/1\} = (1 > 0)$$

$$\mathcal{I} \models (1 > 0)$$

Императивные программы: семантика

Пример

π: **while** $\mathrm{x} > 0$ **do** $\mathrm{x} := \mathrm{x} - 1$ **od** $\qquad \theta = \{\mathrm{x}/1\}$

$\mathcal{I}$ — целочисленная арифметическая интерпретация

Вычисление π на θ в $\mathcal{I}$:

$$\langle \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$

$$\downarrow_{\mathcal{I}}$$

$$\langle \mathrm{x} := \mathrm{x} - 1;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$

$$\downarrow_{\mathcal{I}}$$

$$\langle \emptyset;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1 - 1\}\rangle$$

Пояснение:

$$\{\mathrm{x}/\mathrm{x} - 1\}\,\{\mathrm{x}/1\} = \{\mathrm{x}/1 - 1\}$$

Императивные программы: семантика

Пример

π: **while** $\mathrm{x} > 0$ **do** $\mathrm{x} := \mathrm{x} - 1$ **od** $\qquad \theta = \{\mathrm{x}/1\}$

$\mathcal{I}$ — целочисленная арифметическая интерпретация

Вычисление π на θ в $\mathcal{I}$:

$$\langle \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \mathrm{x} := \mathrm{x} - 1;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \emptyset;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1 - 1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1 - 1\}\rangle$$

Пояснение:

Императивные программы: семантика

Пример

π: **while** $\mathrm{x} > 0$ **do** $\mathrm{x} := \mathrm{x} - 1$ **od** $\qquad \theta = \{\mathrm{x}/1\}$

$\mathcal{I}$ — целочисленная арифметическая интерпретация

Вычисление π на θ в $\mathcal{I}$:

$$\langle \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \mathrm{x} := \mathrm{x} - 1;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \emptyset;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1 - 1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1 - 1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \emptyset,\ \{\mathrm{x}/1 - 1\}\rangle$$

Пояснение:

$$(\mathrm{x} > 0)\,\{\mathrm{x}/1 - 1\} = (1 - 1 > 0)$$
$$\mathcal{I} \not\models (1 - 1 > 0)$$

Императивные программы: семантика

Пример

π: **while** $\mathrm{x} > 0$ **do** $\mathrm{x} := \mathrm{x} - 1$ **od** $\qquad \theta = \{\mathrm{x}/1\}$

$\mathcal{I}$ — целочисленная арифметическая интерпретация

Вычисление π на θ в $\mathcal{I}$:

$$\langle \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \mathrm{x} := \mathrm{x} - 1;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \emptyset;\ \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1 - 1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \textbf{while } \mathrm{x} > 0 \textbf{ do } \mathrm{x} := \mathrm{x} - 1 \textbf{ od},\ \{\mathrm{x}/1 - 1\}\rangle$$
$$\downarrow_{\mathcal{I}}$$
$$\langle \emptyset,\ \{\mathrm{x}/1 - 1\}\rangle$$

Пояснение:

$\{\mathrm{x}/1 - 1\}$ — результат вычисления

Задача верификации программ

И что же такое “правильная программа”?

Неформально:

Программа частично корректна относительно предусловия φ и постусловия ψ (*в интерпретации* $\mathcal{I}$), если для любых начальных данных, удовлетворяющих φ, результат любого конечного вычисления программы удовлетворяет ψ

Программа тотально корректна, если

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

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

Задача верификации программ

И что же такое “правильная программа”?

Формально:

Предусловие φ и постусловие ψ — это формулы логики предикатов, а π — императивная программа

Требование корректности π относительно φ, ψ записывается в виде триплета Хоара (или тройки Хоара):

$$\{\varphi\}\,\pi\,\{\psi\}$$

Триплет Хоара $\{\varphi\}\,\pi\,\{\psi\}$ выполним в интерпретации $\mathcal{I}$ ($\mathcal{I} \models \{\varphi\}\,\pi\,\{\psi\}$), если для любых оценок θ, η верно:

если $\mathcal{I} \models \varphi\theta$ и $\langle \pi,\ \theta\rangle \to^*_{\mathcal{I}} \langle \emptyset,\ \eta\rangle$, то $\mathcal{I} \models \psi\eta$

Программа π частично корректна в интерпретации $\mathcal{I}$ относительно предусловия φ и постусловия ψ, если

$$\mathcal{I} \models \{\varphi\}\,\pi\,\{\psi\}$$

Задача верификации программ

Чтобы научиться доказывать частичную корректность программ, достаточно придумать систему **правил вывода**, аналогичную правилам табличного вывода для логики предикатов

Правила такой системы[1] могут иметь следующий вид:

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

($\varphi, \psi \in$ Form; Φ, Ψ_1, Ψ_2 — триплеты Хоара)

Содержательно эти правила прочитываются так:

триплет Φ выполним

$\Leftrightarrow$

выполнимы все триплеты и истинны все формулы, записанные под чертой

---

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

Логика Хоара

Вот эти правила:

SKIP: $\dfrac{\{\varphi\}\,\emptyset\,\{\varphi\}}{\textbf{true}}$ **AS**: $\dfrac{\{\varphi\,\{\mathrm{x}/t\}\}\,\mathrm{x} := t\,\{\varphi\}}{\textbf{true}}$

(*переменная* x *свободна для терма* t *в формуле* φ)

INF: $$\frac{\{\varphi\}\,\pi\,\{\psi\}}{\varphi \to \varphi', \{\varphi'\}\,\pi\,\{\psi'\}, \psi' \to \psi}$$

COMP: $$\frac{\{\varphi\}\,\pi_1;\ \pi_2\,\{\psi\}}{\{\varphi\}\,\pi_1\,\{\chi\}, \{\chi\}\,\pi_2\,\{\psi\}}$$

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

WHILE: $$\frac{\{\varphi\}\,\textbf{while } C \textbf{ do } \pi \textbf{ od}\,\{\varphi \& \neg C\}}{\{\varphi \& C\}\,\pi\,\{\varphi\}}$$

Формула φ в правиле **WHILE** называется инвариантом цикла

Логика Хоара

Вывод триплета $\{\varphi\}\,\pi\,\{\psi\}$ — это корневое ориентированное дерево следующего вида:

- вершины размечены триплетами Хоара и формулами логики предикатов
- корень помечен триплетом $\{\varphi\}\,\pi\,\{\psi\}$
- потомки вершины определяются так же, как и в дереве табличного вывода для логики предикатов, но для системы правил **SKIP**, **AS**, **INF**, **COMP**, **IF**, **WHILE**
- листья помечены формулами логики предикатов

Успешный вывод триплета $\{\varphi\}\,\pi\,\{\psi\}$ в интерпретации $\mathcal{I}$ — это конечный вывод, все листья которого помечены формулами, истинными в $\mathcal{I}$

Логика Хоара

Теорема корректности правил вывода Хоара

Для любой интерпретации $\mathcal{I}$ и любого правила вывода логики Хоара

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

если $\mathcal{I} \models \Psi_1$, $\mathcal{I} \models \Psi_2$ и формулы φ, ψ истинны в $\mathcal{I}$, то $\mathcal{I} \models \Phi$

Доказательство.

Подробно рассмотрим только правило **AS**: $\dfrac{\{\varphi\{\mathtt{x}/t\}\}\,\mathtt{x} := t\,\{\varphi\}}{\textbf{true}}$

Рассмотрим произвольную оценку θ, такую что $\mathcal{I} \models \varphi\{\mathtt{x}/t\}\,\theta$, и соотношение $\langle \mathtt{x} := t,\ \theta\rangle \to_{\mathcal{I}} \langle \pi,\ \eta\rangle$

Операционная семантика программ: $\pi = \emptyset$ и $\eta = \{\mathtt{x}/t\}\,\theta$

Значит, $\mathcal{I} \models \eta$ и $\mathcal{I} \models \{\varphi\{\mathtt{x}/t\}\}\,\mathtt{x} := t\,\{\varphi\}$

Корректность остальных правил можете попробовать обосновать самостоятельно

▼

Логика Хоара

Теорема корректности правил вывода Хоара

Для любой интерпретации $\mathcal{I}$ и любого правила вывода логики Хоара

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

если $\mathcal{I} \models \Psi_1$, $\mathcal{I} \models \Psi_2$ и формулы φ, ψ истинны в $\mathcal{I}$, то $\mathcal{I} \models \Phi$

Следствие: корректность метода Хоара

Если существует успешный вывод триплета $\{\varphi\}\,\pi\,\{\psi\}$ в интерпретации $\mathcal{I}$, то программа π частично корректна в $\mathcal{I}$ относительно предусловия φ и постусловия ψ

А насколько сложно сформулировать и доказать теорему полноты метода Хоара?

Логика Хоара

Пример: алгоритм Эвклида

Рассмотрим такую программу π:

while $\mathtt{x} \neq \mathtt{y}$ **do if** $\mathtt{x} > \mathtt{y}$ **then** $\mathtt{x} := \mathtt{x} - \mathtt{y}$ **else** $\mathtt{y} := \mathtt{y} - \mathtt{x}$ **fi od**

Рассмотрим такие формулы:

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

Для **доказательства** того, что по выполнении программы π в целочисленной арифметической интерпретации $\mathcal{I}$ в переменную $\mathtt{x}$ записывается наибольший общий делитель исходных *положительных* чисел $\mathtt{x}$, $\mathtt{y}$, достаточно построить успешный вывод триплета $\{\varphi\}\,\pi\,\{\psi\}$ в интерпретации $\mathcal{I}$

Логика Хоара

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

χ_1: $\mathtt{x} > \mathbf{0} \,\&\, \mathtt{x} > \mathbf{0} \,\&\, \gcd(\mathtt{x}, \mathtt{y}, \mathtt{z}) \to \mathtt{x} > \mathbf{0} \,\&\, \mathtt{y} > \mathbf{0} \,\&\, \gcd(\mathtt{x}, \mathtt{y}, \mathtt{z})$

χ_2: $\mathtt{x} > \mathbf{0} \,\&\, \mathtt{y} > \mathbf{0} \,\&\, \gcd(\mathtt{x}, \mathtt{y}, \mathtt{z}) \,\&\, \neg(\mathtt{x} \neq \mathtt{y}) \to \mathtt{x} = \mathtt{z}$

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

INF: $\dfrac{\{\varphi\}\,\pi\,\{\psi\}}{\varphi \to \varphi', \{\varphi'\}\,\pi\,\{\psi'\}, \psi' \to \psi}$

χ_1 истинна в $\mathcal{I}$

χ_2 истинна в $\mathcal{I}$

Логика Хоара

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

χ_1: $\mathtt{x} > \mathbf{0} \,\&\, \mathtt{x} > \mathbf{0} \,\&\, \gcd(\mathtt{x},\mathtt{y},\mathtt{z}) \to \mathtt{x} > \mathbf{0} \,\&\, \mathtt{y} > \mathbf{0} \,\&\, \gcd(\mathtt{x},\mathtt{y},\mathtt{z})$

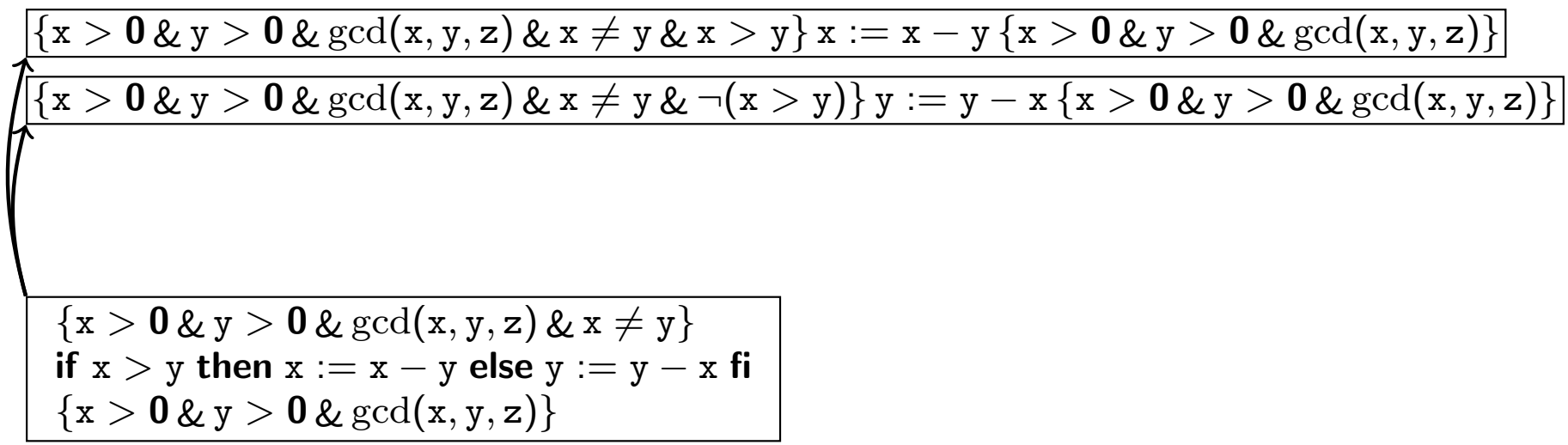
χ_2: $\mathtt{x} > \mathbf{0} \,\&\, \mathtt{y} > \mathbf{0} \,\&\, \gcd(\mathtt{x},\mathtt{y},\mathtt{z}) \,\&\, \neg(\mathtt{x} \neq \mathtt{y}) \to \mathtt{x} = \mathtt{z}$

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

$\{\mathtt{x} > \mathbf{0} \,\&\, \mathtt{y} > \mathbf{0} \,\&\, \gcd(\mathtt{x},\mathtt{y},\mathtt{z}) \,\&\, \mathtt{x} \neq \mathtt{y}\}$
if $\mathtt{x} > \mathtt{y}$ **then** $\mathtt{x} := \mathtt{x} - \mathtt{y}$ **else** $\mathtt{y} := \mathtt{y} - \mathtt{x}$ **fi**
$\{\mathtt{x} > \mathbf{0} \,\&\, \mathtt{y} > \mathbf{0} \,\&\, \gcd(\mathtt{x},\mathtt{y},\mathtt{z})\}$

WHILE: $$\frac{\{\varphi\}\ \textbf{while}\ C\ \textbf{do}\ \pi\ \textbf{od}\ \{\varphi \& \neg C\}}{\{\varphi \& C\}\ \pi\ \{\varphi\}}$$

Логика Хоара



IF: $\dfrac{\{\varphi\}\,\textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi}\,\{\psi\}}{\{\varphi \& C\}\,\pi_1\,\{\psi\}\,,\{\varphi \& \neg C\}\,\pi_2\,\{\psi\}}$

Логика Хоара

$\{\mathtt{x}-\mathtt{y}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\,\{\mathtt{x}/\mathtt{x}-\mathtt{y}\}\}\,\mathtt{x}:=\mathtt{x}-\mathtt{y}\,\{\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\}$

χ_3: $\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\,\&\,\mathtt{x}\neq\mathtt{y}\,\&\,\mathtt{x}>\mathtt{y}\to\mathtt{x}-\mathtt{y}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\,\{\mathtt{x}/\mathtt{x}-\mathtt{y}\}$

χ_4: $\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\to\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})$

$\{\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\,\&\,\mathtt{x}\neq\mathtt{y}\,\&\,\mathtt{x}>\mathtt{y}\}\,\mathtt{x}:=\mathtt{x}-\mathtt{y}\,\{\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\}$

$\{\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\,\&\,\mathtt{x}\neq\mathtt{y}\,\&\,\neg(\mathtt{x}>\mathtt{y})\}\,\mathtt{y}:=\mathtt{y}-\mathtt{x}\,\{\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\}$

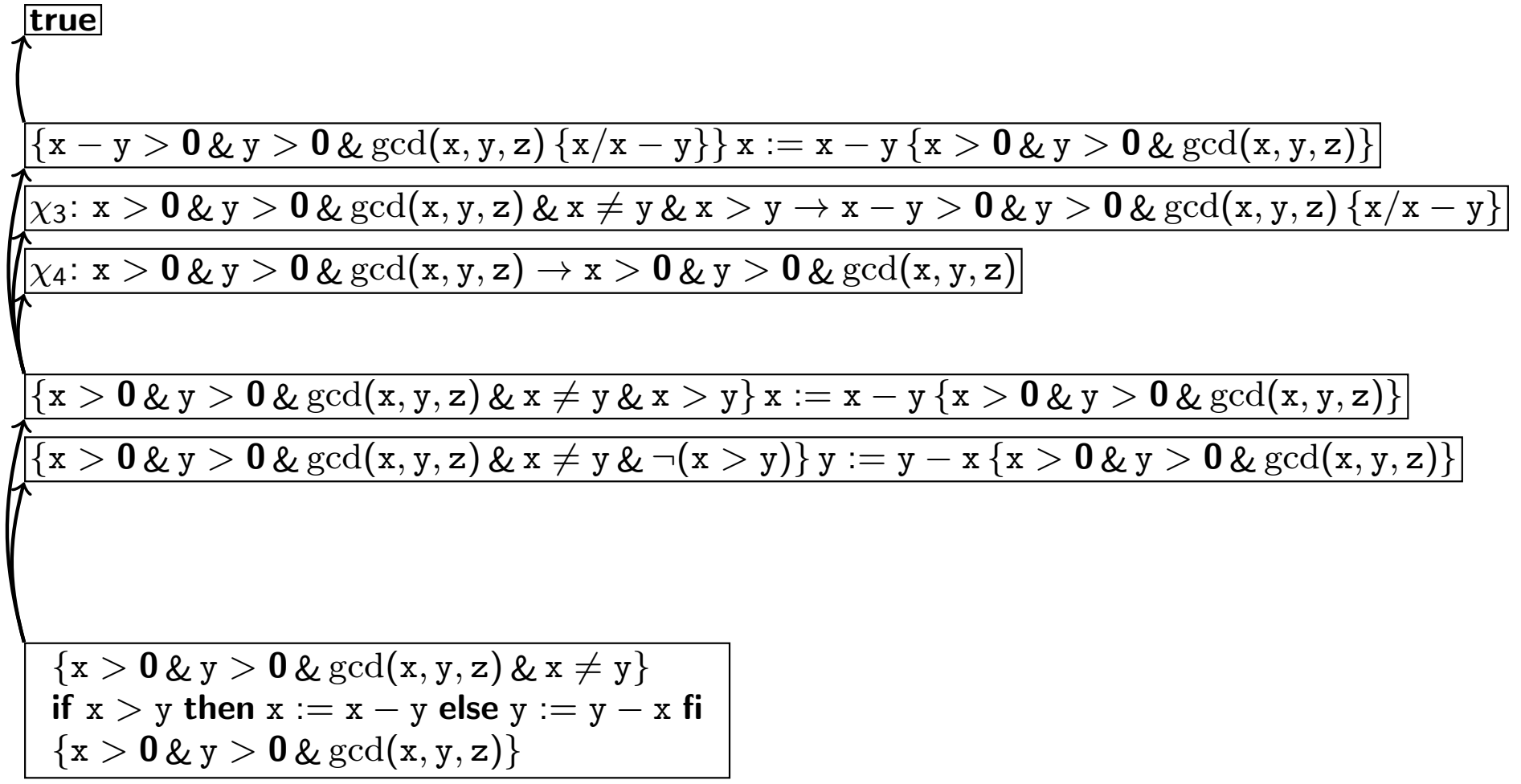
$\{\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\,\&\,\mathtt{x}\neq\mathtt{y}\}$
if $\mathtt{x}>\mathtt{y}$ **then** $\mathtt{x}:=\mathtt{x}-\mathtt{y}$ **else** $\mathtt{y}:=\mathtt{y}-\mathtt{x}$ **fi**
$\{\mathtt{x}>\mathbf{0}\,\&\,\mathtt{y}>\mathbf{0}\,\&\,\gcd(\mathtt{x},\mathtt{y},\mathtt{z})\}$

INF: $\dfrac{\{\varphi\}\,\pi\,\{\psi\}}{\varphi\to\varphi',\{\varphi'\}\,\pi\,\{\psi'\},\psi'\to\psi}$

χ_3 истинна в $\mathcal{I}$

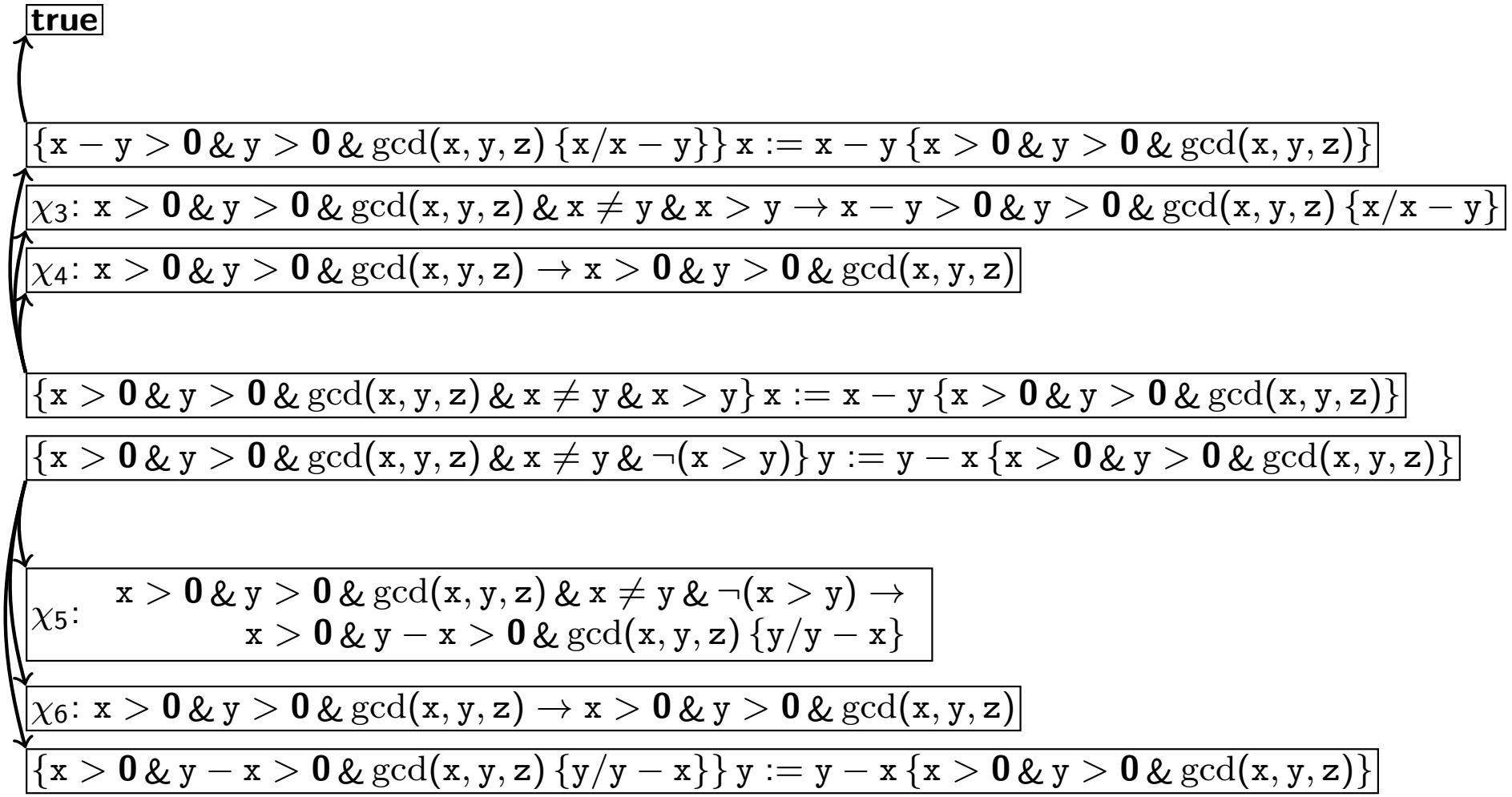
χ_4 истинна в $\mathcal{I}$

Логика Хоара



AS: $\dfrac{\{\varphi\,\{\mathtt{x}/t\}\}\,\mathtt{x}:=t\,\{\varphi\}}{\mathbf{true}}$

Логика Хоара

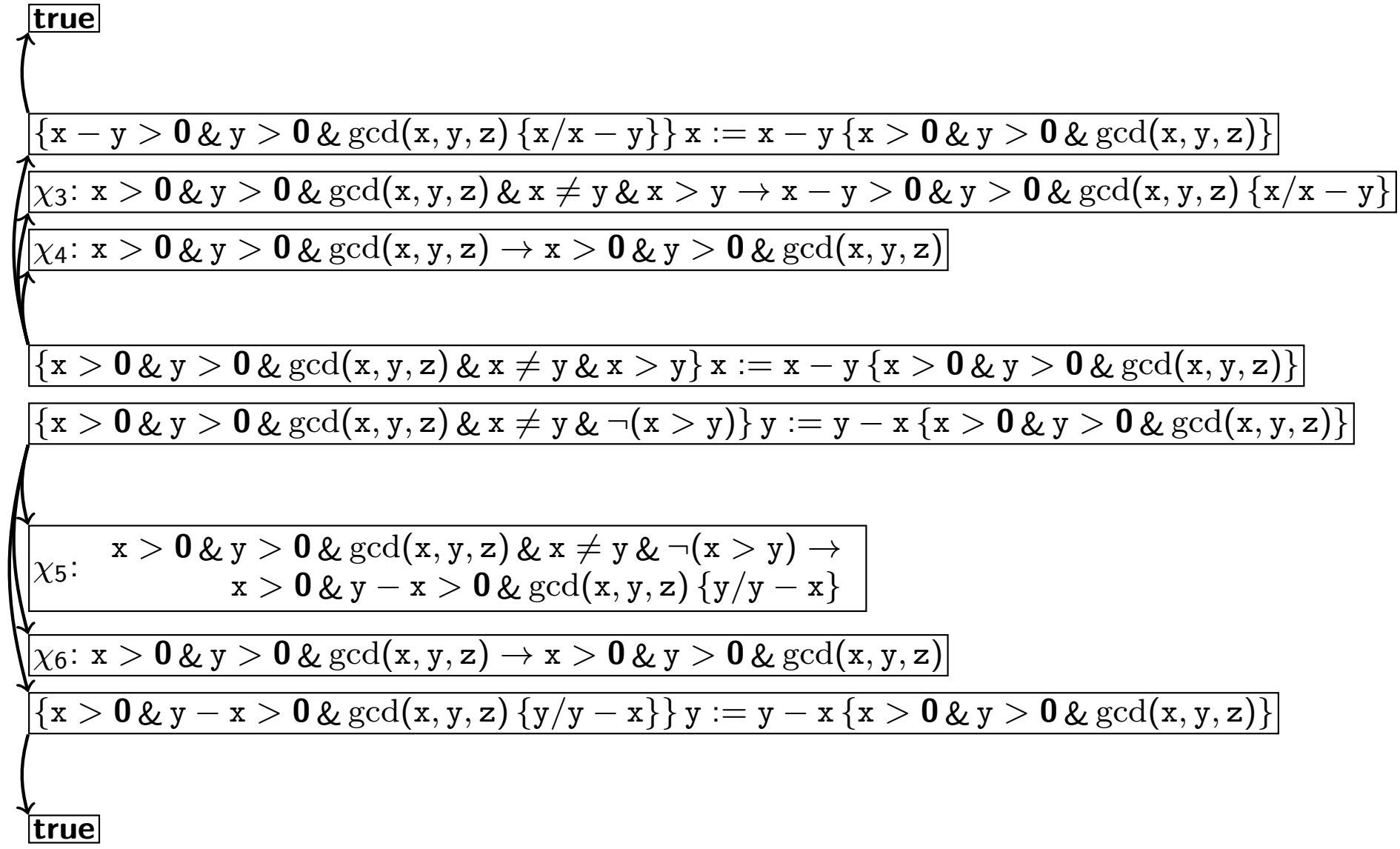


INF: $\dfrac{\{\varphi\}\,\pi\,\{\psi\}}{\varphi \to \varphi',\ \{\varphi'\}\,\pi\,\{\psi'\},\ \psi' \to \psi}$

χ_5 истинна в $\mathcal{I}$

χ_6 истинна в $\mathcal{I}$

Логика Хоара



AS: $\dfrac{\{\varphi\,\{x/t\}\}\, x := t\, \{\varphi\}}{\text{true}}$

Автоматическая проверка правильности программ

А всё-таки, полна ли система правил логики Хоара, и можно ли с её помощью автоматически проверять корректность императивных программ?

Нет, и проблема возникает не одна

I. Необходимо уметь проверять истинность формул логики предикатов в заданной интерпретации $\mathcal{I}$

Если $\mathcal{I}$ — арифметическая интерпретация, то проверка истинности формул невозможна даже в простых случаях:

- теорема Гёделя о неполноте: не существует рекурсивно перечислимой системы правил анализа арифметического смысла формул, использующих $+$, $\times$ и $=$
- доказательство теоремы Гёделя о неполноте: для любой вычислимой арифметической функции f существует формула, арифметический смысл которой — график f

Автоматическая проверка правильности программ

II. При формулировке пред- и постусловий и при построении успешного вывода может потребоваться использование *достаточно хорошей* формулы:

$$\textbf{INF}: \quad \frac{\{\varphi\}\,\pi\,\{\psi\}}{\varphi \to \varphi',\, \{\varphi'\}\,\pi\,\{\psi'\},\, \psi' \to \psi}$$

$$\textbf{COMP}: \quad \frac{\{\varphi\}\,\pi_1;\ \pi_2\,\{\psi\}}{\{\varphi\}\,\pi_1\,\{\chi\},\, \{\chi\}\,\pi_2\,\{\psi\}}$$

Если сигнатура программ подобрана неудачно, то такой формулы может и не существовать

Например, при доказательстве корректности императивных программ, из всех арифметических операций использующих **только сложение** чисел, в общем случае требуются формулы, использующие и **умножение** чисел[1]

[1] А если включить умножение в сигнатуру, то возникает проблема *I*

Автоматическая проверка правильности программ

III. При построении успешного вывода для программы, содержащей циклы, необходимо *подходящим образом* применить правило

$$\textbf{WHILE}: \frac{\{\varphi\}\,\textbf{while}\ C\ \textbf{do}\ \pi\ \textbf{od}\,\{\varphi\&\neg C\}}{\{\varphi\&C\}\,\pi\,\{\varphi\}}$$

Это единственное правило логики Хоара, позволяющее устранить цикл при построении вывода

Для применения этого правила требуется предварительно найти подходящую формулу φ — **инвариант цикла**

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

Автоматическая проверка правильности программ

Императивные программы **полны по Тьюрингу**,[1] а значит, их анализ настолько же труден, как и анализ машин Тьюринга

Если ограничить синтаксис анализируемых программ настолько, чтобы не допустить полноты по Тьюрингу, то проверка их корректности может стать довольно простой

Например, корректность программы

$$\mathtt{x} := \mathtt{x} + 500$$

проверить даже проще, чем корректность реализации алгоритма Эвклида

---

[1] Значениями программ со сложением и умножением в арифметической интерпретации являются **всевозможные** вычислимые функции

Автоматическая проверка правильности программ

Проверка корректности *простых* программ тоже достаточно полезна,[1] но модели императивных программ для этого может быть недостаточно:

- что делать, если требуется проверить корректность не одной изолированной программы, а системы из параллельно работающих программ, взаимодействующих между собой и с окружающей средой?[2]
  - В современном компьютерном мире программа **никогда** не работает изолированно

---

[1] Например, **драйверы** устройств, **планировщики** и **базовые библиотеки** нередко имеют довольно простое устройство

[2] То есть образуют **распределённую систему**

Автоматическая проверка правильности программ

Проверка корректности *простых* программ тоже достаточно полезна,[1] но модели императивных программ для этого может быть недостаточно:

- что делать, если компоненты системы написаны не в стиле императивных программ?
  - Алгоритмические языки, используемые современными программистами, относятся к **огромному** числу существенно разных парадигм и имеют **огромное** число неочевидных стандартных особенностей поведения
- что делать, если компоненты системы **не являются программами**?
  - Программе неважно, с кем она общается: с другой программой, аппаратным устройством, человеком или чем-то другим — важно только то, как происходит обмен информацией при общении

Верификация распределённых систем

Пример: рассмотрим две функции, **параллельно** и **независимо** изменяющие переменную счёт

```
void премия() {
  счёт += 1 000 000;
}
```

```
void зарплата() {
  счёт += 1 000;
}
```

Как изменится счёт после одной зарплаты и одной премии?

$\{\text{счёт} = x\}$
$\quad \text{счёт} := \text{счёт} + 1\,000\,000;$
$\quad \text{счёт} := \text{счёт} + 1\,000$
$\{\text{счёт} = x + 1\,001\,000\}$

$\{\text{счёт} = x\}$
$\quad \text{счёт} := \text{счёт} + 1\,000;$
$\quad \text{счёт} := \text{счёт} + 1\,000\,000$
$\{\text{счёт} = x + 1\,001\,000\}$

Результат верификации программы полезен ровно настолько, насколько адекватна модель, применяемая для верификации

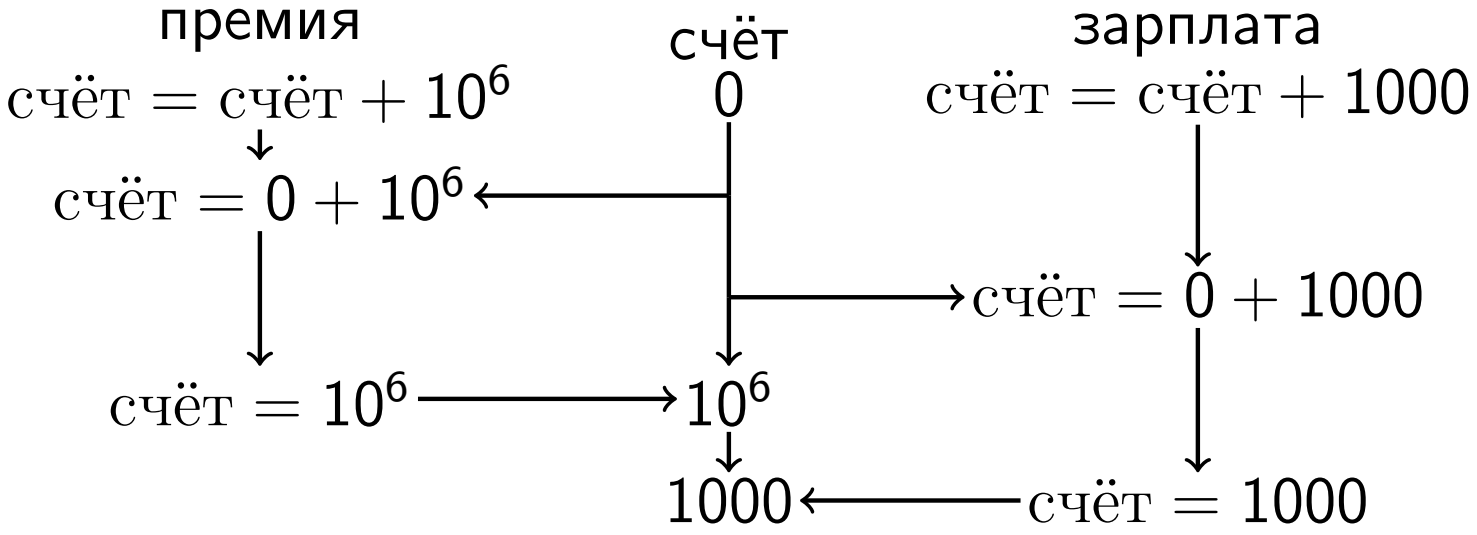
Насколько разумно использование императивных программ в этом примере?

Верификация распределённых систем

Пример: рассмотрим две функции, **параллельно** и **независимо** изменяющие переменную счёт

```
void премия() {
  счёт += 1 000 000;
}
```

```
void зарплата() {
  счёт += 1 000;
}
```

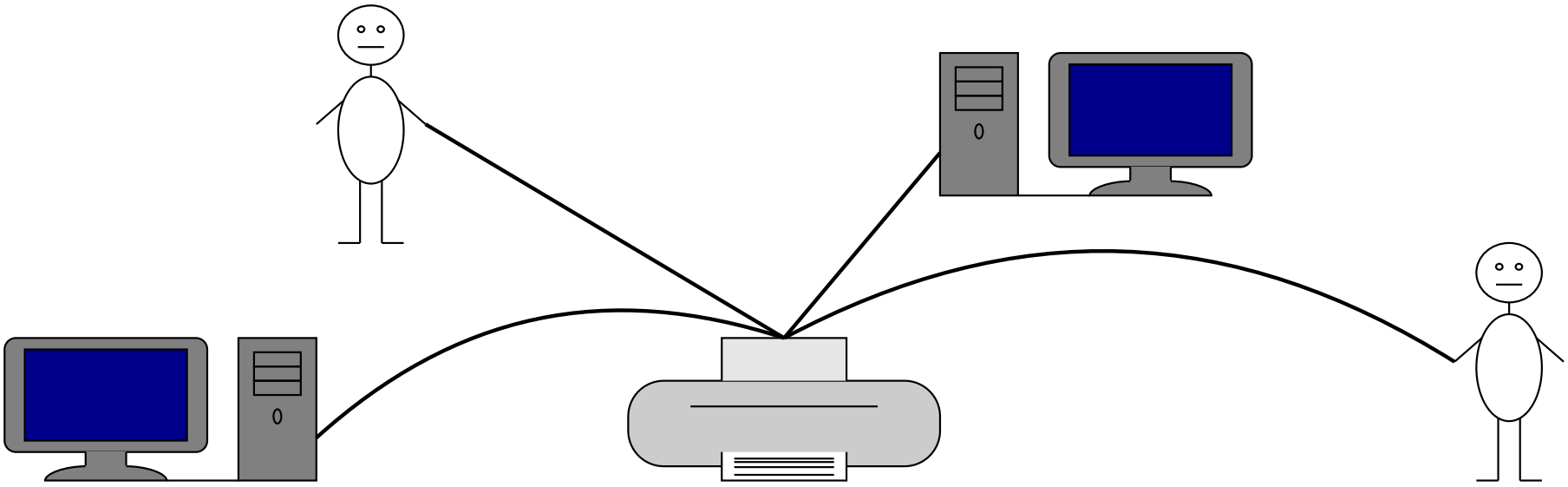


Где мой миллион?!

Такая ошибка возникает крайне редко, и потому считается **труднообнаружимой** и **невоспроизводимой**

Верификация распределённых систем

Ещё один 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

Как могут выглядеть ошибки выполнения такой системы, и как проверить их отсутствие?

Верификация распределённых систем

К чему приводят такие ошибки?[1]

- *1996* взорвалась ракета “Ариан 5”; причина: зависание при преобразовании чисел с плавающей точкой в целые числа в навигационной программе бортового компьютера; ущерб: 500 млн.$, срыв программы коммерческих запусков спутников
- *1994* массовая замена дефектных процессоров компанией Intel; причина: некорректная аппаратная реализация инструкции деления чисел с плавающей точкой; ущерб: сотни миллионов $
- *1982* при лечении аппаратом облучения раковой опухоли Therac-25 погибло 2 человека, несколько остались инвалидами; причина: редко проявлявшееся состояние гонки (race condition) при параллельной работе многих подпрограмм, обслуживающих аппарат, в результате которого интенсивность облучения возрастала на 2 порядка
- *1995* крушение самолёта “Боинг-757”, 159 погибших; причина: ошибка в одном символе программной системы управления полётом

---

[1] Карпов. Model checking. 2010

Верификация распределённых систем

Как обнаруживать такие ошибки?

Например, если все компоненты распределённой системы — простые программы, то что мешает явно перебрать все сценарии работы и для каждого из них убедиться в отсутствии ошибок?

Пусть в системе параллельно работают 70 программ, каждая из которых совершает одно действие и завершается

Тогда всевозможных сценариев работы будет **70!**

Гугол

(и даже немного больше)

При этом разные последовательности действий могут приводить к совершенно разным и неожиданным результатам

(*где мой миллион?!*)

Верификация распределённых систем

Как обнаруживать такие ошибки?

- **тестирование**: анализировать отклонение выполнения системы на особым образом выбираемых входных данных от ожидаемых результатов — **не подходит** (почему?)
- **формальная верификация**

Прежде всего остановимся на том, какие **требования** обычно предъявляются к распределённым системам

В таких требованиях часто присутствует время:

- *в тот момент, когда* функции завершат работу, на счёт поступит 1 001 000
- *в какой бы момент времени* ни пришёл запрос на печать, *когда-нибудь в будущем* документ обязательно напечатается
- *в любой момент времени* принтером обслуживается не более чем один участник

Верификация распределённых систем

Язык *классической* логики **никак** не учитывает время:

- атомарные формулы безусловно истинны или ложны в интерпретации
- значение более сложных формул в интерпретации зависит от элементарных событий и не зависит от времени их наступления
- истинность элементарных высказываний в требованиях к распределённым системам зависит от момента времени, в который обозревается система:
  - сейчас принтером обслуживается один участник, а через минуту — три
  - сейчас мы наблюдаем пришедший на принтер запрос, а через пять секунд запроса больше нет, а принтер печатает

А как учитывать такие “оттенки” истинности:
утверждение истинно, но с некоторыми поправками?

Модальные логики

Начнём с примера:

1: **Зима близко**

2: **Зима всегда близко**

3: **Зима бывает близко**

Эти утверждения несут разный смысл, а значит, должны быть описаны разными формулами

Тем не менее утверждения связаны по смыслу, и эту взаимосвязь неплохо было бы отразить в формулах

Утверждения отличаются только словами всегда и бывает:

модальностями времени

Насколько просто анализировать эти модальности при помощи кванторов?

Модальные логики

Начнём с примера:

1: Зима близко

2: Зима должна быть близко

3: Зима имеет право быть близко

Должен и имеет право — это деонтические модальности

А как применить кванторы для описания этих модальностей?

Модальные логики

Начнём с примера:

1: **Зима близко**

2: **Известно, что зима близко**

3: **Можно допустить, что зима близко**

Известно и допустимо — это эпистемологические модальности

А если для всех видов модальностей единообразно применять кванторы, то как отличить эту модальность от предыдущих?

Модальные логики

Модальность — это выражение, описывающее “оттенок истинности” высказывания (уверенность, необходимость, доказуемость, осведомлённость, . . .)

Такие оттенки можно разбить на две категории:

Модальность необходимого	**Модальность возможного**
необходимо	возможно
обязательно	не исключено
всегда	иногда
должен	имеет право
знает	предполагает
доказуемо	непротиворечиво
$\Box$	$\Diamond$

Модальные логики

Синтаксис модальных формул

- Начнём с синтаксиса формул логики высказываний
- Добавим к алфавиту модальные операторы (модальности):

$$\Box, \quad \Diamond$$

- Добавим к синтаксису такое правило:
  - $(\Box\varphi)$ и $(\Diamond\varphi)$ — формулы (φ — формула)
- Считаем, что модальности имеют наивысший приоритет

Модальности $\Box$, $\Diamond$ могут иметь совершенно разное значение с общим “оттенком” необходимости или возможности

А как строго определить значение модальностей, чтобы при этом охватить всевозможные оттенки?

Модальные логики

Прежде чем перейти к описанию семантики модальных формул, рассмотрим такой **пример:**

верна ли формула $\Box\varphi \to \varphi$**?**

Да, если $\Box$ — модальность времени:

если зима всегда близко, то она близко

Нет, если $\Box$ — деонтическая модальность:

если зима должна быть близко, то она близко

Как же учесть все трактовки $\Box$, не основываясь на правдивости прогноза погоды?

Семантика Крипке

— это наиболее распространённый способ описания значения модальных формул

Пусть $\mathcal{P}$ — множество пропозициональных переменных

Тогда модель Крипке — это система (W, R, ξ), где

- W — множество состояний (возможных миров)
- $R \subseteq W \times W$ — отношение достижимости миров
- $\xi : W \to 2^{\mathcal{P}}$ — оценка переменных

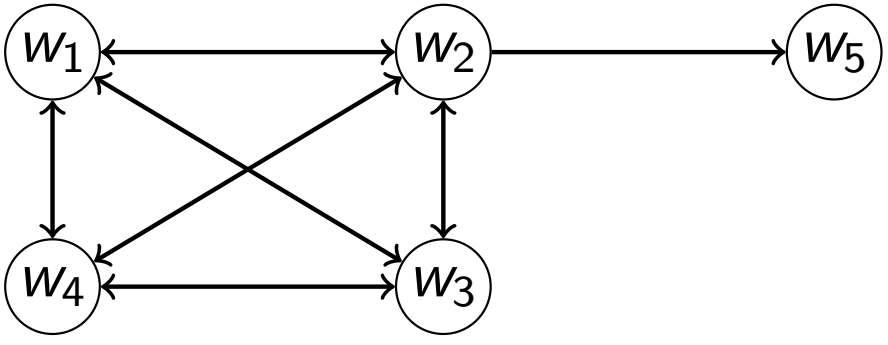
Шкала Крипке (*Kripke frame*), на которой основывается модель (W, R, ξ), — это пара (W, R)

w' — альтернативный мир для w (или w-альтернатива), если $(w, w') \in R$

Модель Крипке — это **интерпретация** модальных формул

Семантика Крипке

Например, $(\mathcal{P} = \{a\})$



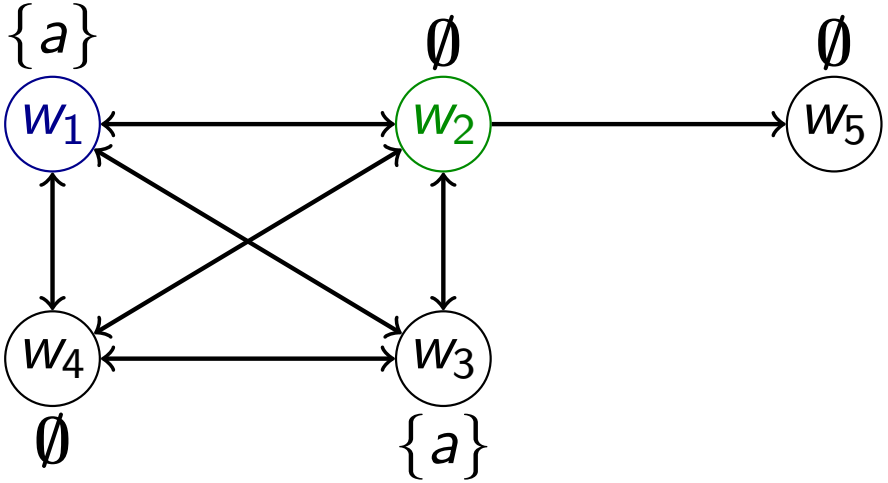
— это шкала Крипке

Семантика Крипке

Отношение выполнимости $\models$ для модели $\mathcal{I} = (W, R, \xi)$ и мира $w \in W$ определяется так:

- $\mathcal{I}, w \models p \quad \Leftrightarrow \quad p \in \xi(w)$ $\qquad (p \in \mathcal{P})$
- $\mathcal{I}, w \models \varphi \& \psi \quad \Leftrightarrow \quad \mathcal{I}, w \models \varphi$ и $\mathcal{I}, w \models \psi$
- $\mathcal{I}, w \models \varphi \vee \psi \quad \Leftrightarrow \quad \mathcal{I}, w \models \varphi$ или $\mathcal{I}, w \models \psi$

Например, $\qquad (\mathcal{P} = \{a\})$



— это модель Крипке $(\mathcal{I})$

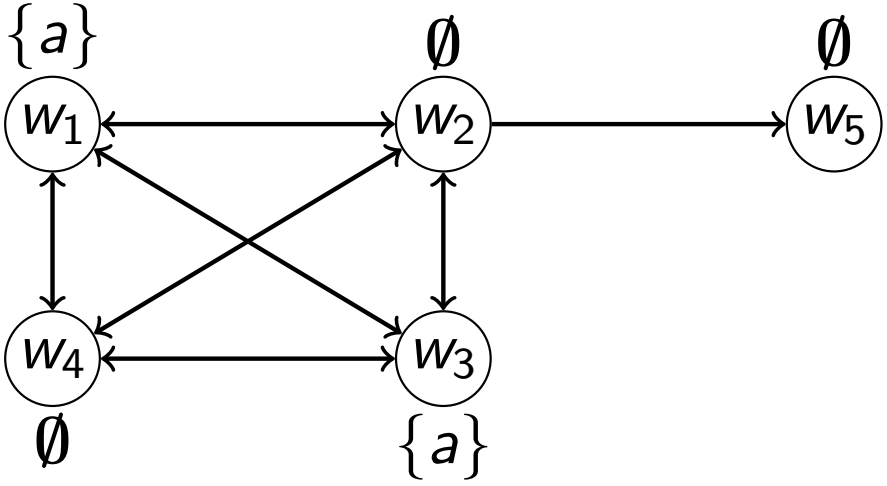
$\mathcal{I}, w_1 \models a, \qquad \mathcal{I}, w_2 \not\models a$

Семантика Крипке

Отношение выполнимости $\models$ для модели $\mathcal{I} = (W, R, \xi)$ и мира $w \in W$ определяется так:

- $\mathcal{I}, w \models \varphi \to \psi \quad \Leftrightarrow \quad \mathcal{I}, w \not\models \varphi$ или $\mathcal{I}, w \models \psi$
- $\mathcal{I}, w \models \neg\varphi \quad \Leftrightarrow \quad \mathcal{I}, w \not\models \varphi$

Например, $(\mathcal{P} = \{a\})$



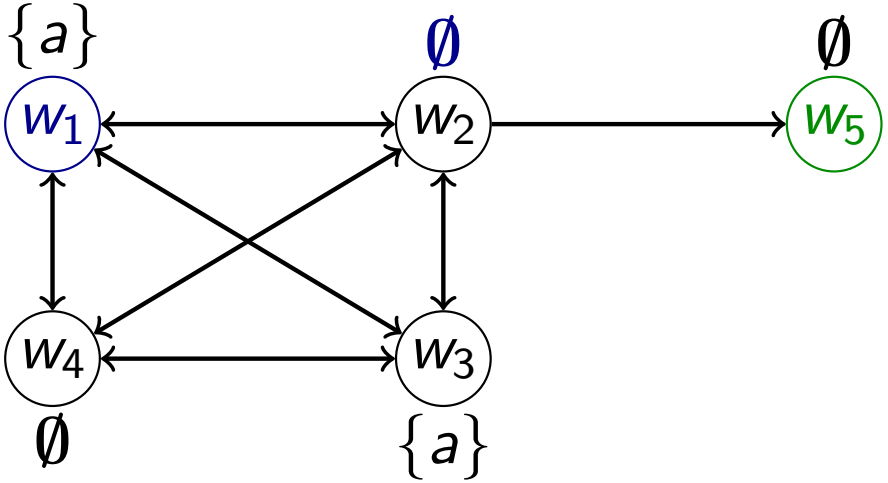
— это модель Крипке $(\mathcal{I})$

Семантика Крипке

Отношение выполнимости $\models$ для модели $\mathcal{I} = (W, R, \xi)$ и мира $w \in W$ определяется так:

- $\mathcal{I}, w \models \Box\varphi \quad \Leftrightarrow$
  для любой w-альтернативы w' верно $\mathcal{I}, w' \models \varphi$

Например, $(\mathcal{P} = \{a\})$



— это модель Крипке $(\mathcal{I})$

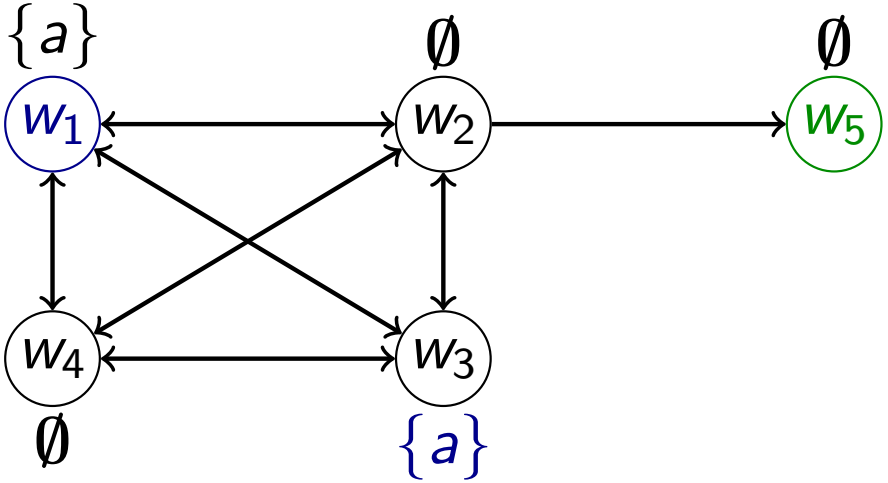
$\mathcal{I}, w_1 \not\models \Box a, \qquad \mathcal{I}, w_5 \models \Box a$

Семантика Крипке

Отношение выполнимости $\models$ для модели $\mathcal{I} = (W, R, \xi)$ и мира $w \in W$ определяется так:

- $\mathcal{I}, w \models \Diamond\varphi \quad \Leftrightarrow$
  существует w-альтернатива w',
  такая что верно $\mathcal{I}, w' \models \varphi$

Например, $(\mathcal{P} = \{a\})$

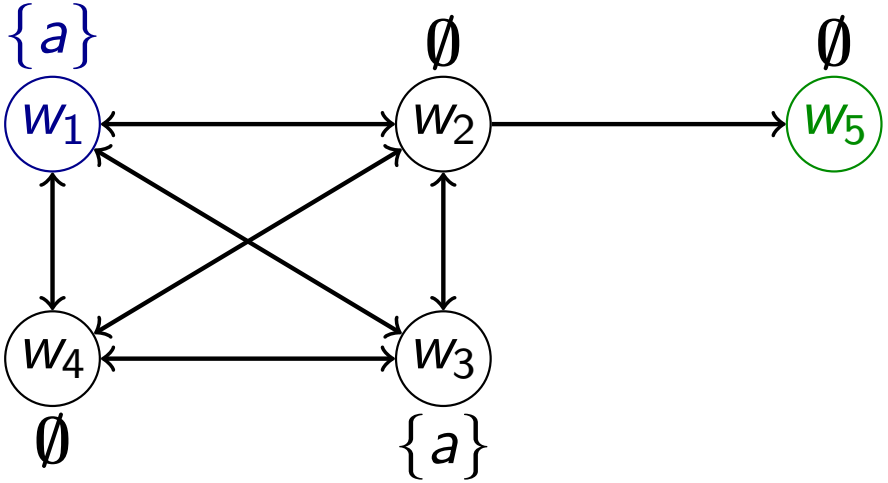


— это модель Крипке $(\mathcal{I})$

$\mathcal{I}, w_1 \models \Diamond a, \qquad \mathcal{I}, w_5 \not\models \Diamond a$

Семантика Крипке

Например, $(\mathcal{P} = \{a\})$

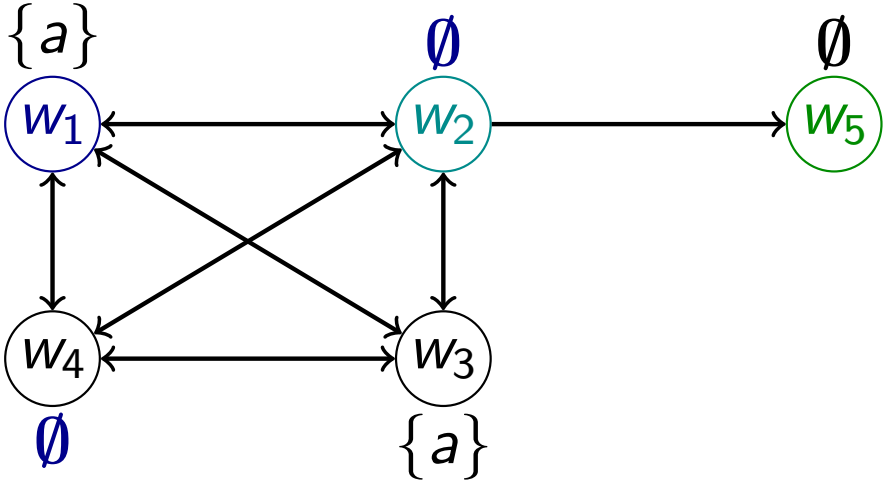


— это модель Крипке $(\mathcal{I})$

$\mathcal{I}, w_1 \models \Box\Diamond a,$ $\mathcal{I}, w_5 \models \Box\Diamond a$

Семантика Крипке

Например, $(\mathcal{P} = \{a\})$



— это модель Крипке $(\mathcal{I})$

$\mathcal{I}, w_1 \not\models \Diamond\Box a,$ $\mathcal{I}, w_2 \models \Diamond\Box a,$ $\mathcal{I}, w_5 \not\models \Diamond\Box a$

Законы модальных логик Эпистемические логики

Модальная логика, как и любая другая, имеет свои законы и свойства, например:

$$\boxed{\Diamond\varphi \approx \neg\Box\neg\varphi} \qquad \boxed{\models \varphi \quad \Rightarrow \quad \models \Box\varphi} \qquad \boxed{\models \Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)}$$

Смыслом модальностей могут определяться и другие законы

Например, в **эпистемической логике** изучаются модальности знания ($\Box$) и допущения ($\Diamond$) и причинно-следственные связи между ними

Основные аксиомы эпистемической логики:

- аксиома адекватности знания:
  $\Box\varphi \to \varphi$ (*мои знания верны*)
- аксиома позитивной интроспекции:
  $\Box\varphi \to \Box\Box\varphi$ (*мне известно, что именно я знаю*)
- аксиома негативной интроспекции:
  $\neg\Box\varphi \to \Box\neg\Box\varphi$ (*мне известно, что именно я не знаю*)

Эпистемические логики

Пример: задача о трёх мудрецах

Король призвал трёх мудрецов и, чтобы убедиться в их мудрости, придумал такое испытание: показал им три чёрные шапки и две белые, завязал глаза, надел на каждого чёрную шапку и развязал глаза

Затем спросил: "Знаете ли вы, какая на вас шапка?"

"Нет, не знаем", хором ответили мудрецы

Он задал тот же вопрос второй раз

"Нет, не знаем", хором ответили мудрецы

Он спросил то же в третий раз

"**Да, чёрная**", хором ответили мудрецы

А как строго описать законы и ход рассуждения мудрецов при помощи эпистемической логики?

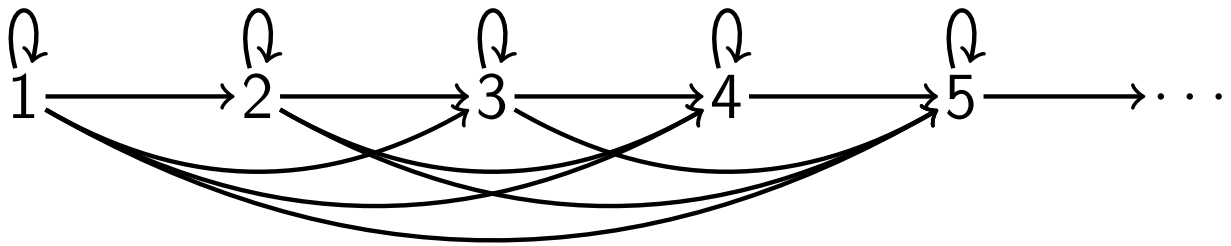
Модальные логики

Аксиомами модальной логики определяется

- точное значение модальностей $\Box$, $\Diamond$
- совокупность шкал и миров Крипке, адекватно подходящих для описания значения модальностей

Значение модальностей можно определить не только при помощи аксиом, но и явным описанием рассматриваемого множества шкал и миров

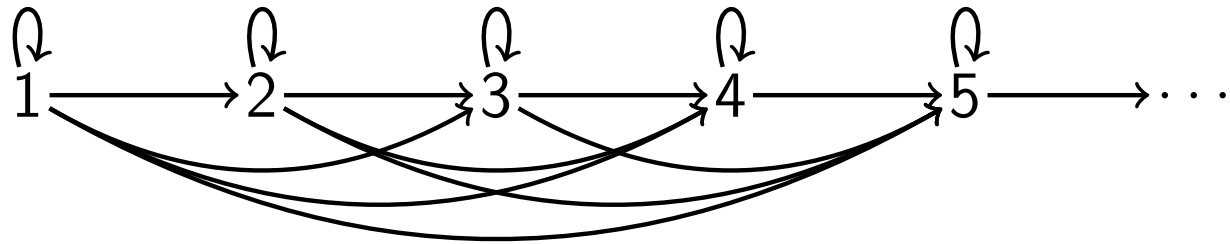
Пример:



Какой содержательный смысл могут иметь соотношения $\mathcal{I}, 1 \models \Box p$ и $\mathcal{I}, 1 \models \Diamond p$ для интерпретации $\mathcal{I}$, основанной на такой шкале?

Темпоральные логики

Пример:



Если $1, 2, \ldots$ — моменты времени и p — свойство, истинность которого зависит от выбранного момента времени, то

- $\Box p$ означает "свойство p будет верным всегда"
- $\Diamond p$ означает "обязательно наступит момент времени, когда свойство p станет верным"

Темпоральная (временная) логика — это модальная логика (*или её расширение*), в которой модальности $\Box$ и $\Diamond$ имеют значение всегда в будущем и когда-нибудь в будущем

Пропозициональные переменные формул темпоральных логик иногда называют элементарными событиями, происходящими в моменты времени, размеченные этими событиями

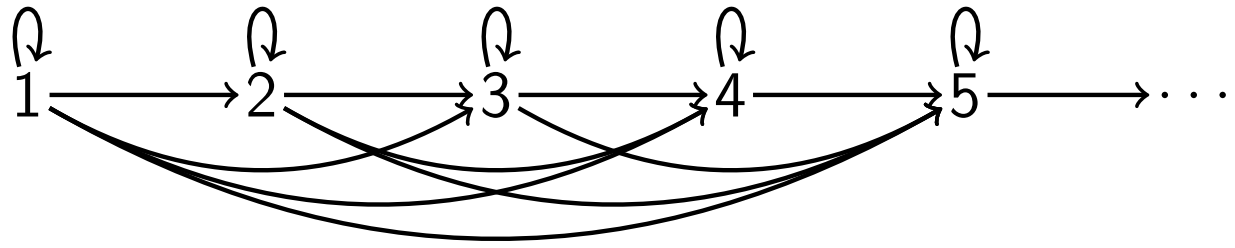
Темпоральные логики

Время может истолковываться по-разному, и в зависимости от истолкования (*то есть точного вида рассматриваемых шкал*) могут получаться разные темпоральные логики, например:

- Логика линейного времени
  (**LTL**, **L**inear **T**emporal **L**ogic)
  - время линейно течёт вперёд
  - формула — это свойство линейного развития событий
- Логика деревьев вычислений
  (**CTL**, **C**omputation **T**ree **L**ogic)
  - время — это частично упорядоченное множество
  - в каждый момент времени происходит выбор альтернативы развития событий
  - в формуле явно указаны альтернативы, к которым она относится

Логика линейного времени

Шкала Крипке LTL — это естественно упорядоченный натуральный ряд:



Модальности $\Box$ и $\Diamond$ в логике линейного времени обозначаются символами **G** (**G**lobally) и **F** (**F**uture)

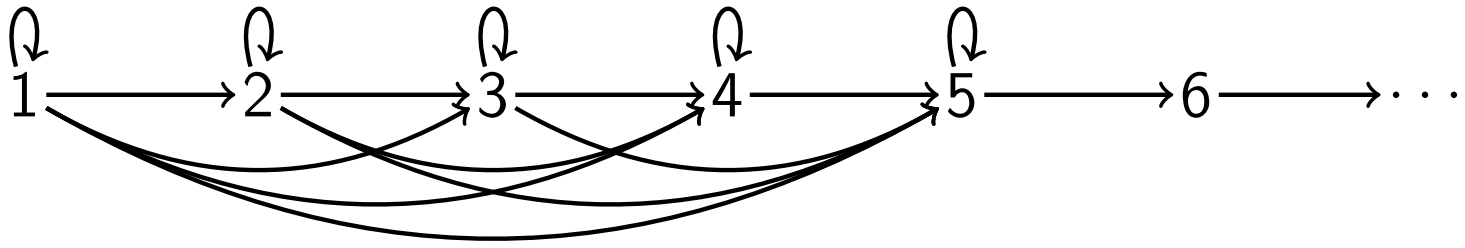
К ним добавляются и другие темпоральные операторы:

X (ne**X**ttime) и **U** (**U**ntil)

- $\mathcal{I}, n \models \mathbf{X}\varphi \Leftrightarrow \mathcal{I}, n+1 \models \varphi$
- $\mathcal{I}, n \models \varphi \mathbf{U} \psi \Leftrightarrow$ существует момент времени k, $k \geq n$, такой что $\mathcal{I}, k \models \psi$ и для всех моментов k', $n \leq k' < k$, верно $\mathcal{I}, k' \models \varphi$

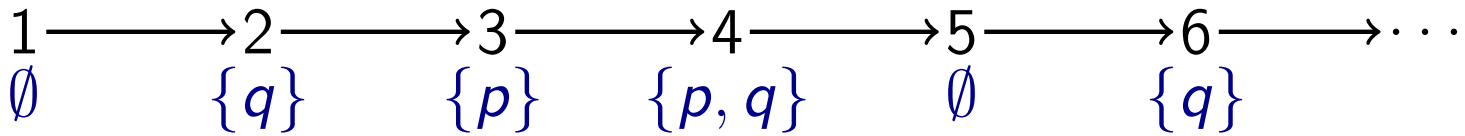
Логика линейного времени

Пример:



Логика линейного времени

Пример: рассмотрим такую модель Крипке $\mathcal{I}$:



Справедливы следующие соотношения:

$\mathcal{I}, 1 \not\models p$

$\mathcal{I}, 1 \not\models \mathbf{X}p$

$\mathcal{I}, 1 \models \mathbf{F}p$

$\mathcal{I}, 1 \not\models \mathbf{FG}p$

$\mathcal{I}, 1 \not\models q\mathbf{U}p$

$\mathcal{I}, 1 \models p \to q$

$\mathcal{I}, 1 \models \mathbf{XX}p$

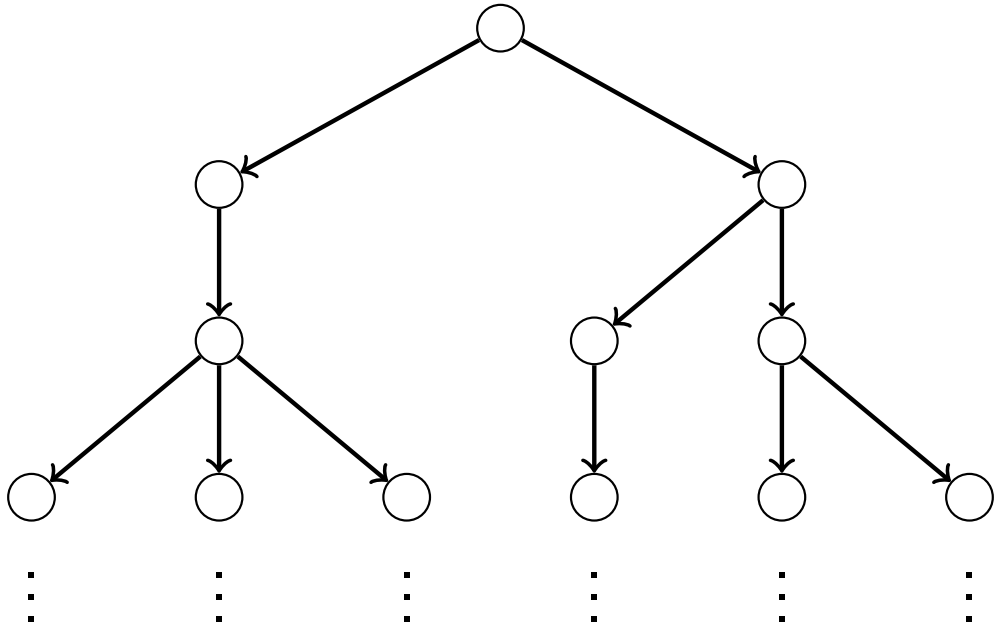
$\mathcal{I}, 1 \not\models \mathbf{G}p$

$\mathcal{I}, 1 \models \mathbf{GF}p$

$\mathcal{I}, 1 \models (q \vee \mathbf{X}q)\mathbf{U}(p \,\&\, q)$

Логика деревьев вычислений

Течение времени в CTL можно описать ориентированным деревом, содержащим только бесконечные ветви:



CTL-шкала — это шкала Крипке, являющаяся рефлексивно-транзитивным замыканием такого дерева

CTL-интерпретация — это модель Крипке, основанная на CTL-шкале

Логика деревьев вычислений

В логике деревьев вычислений модальности $\Box$, $\Diamond$ обозначаются записями **AG** и **EF**

CTL содержит и другие модальности: **EG**, **AF**, **EX**, **AX**, **EU**, **AU**

Значение этих модальностей определяется так:

- $\mathcal{I}, v \models \mathbf{EG}\varphi \Leftrightarrow$ существует ветвь дерева, исходящая из v, такая что для каждой вершины v' этой ветви верно $\mathcal{I}, v' \models \varphi$
- $\mathcal{I}, v \models \mathbf{AF}\varphi \Leftrightarrow$ в каждой ветви дерева, исходящей из v, существует вершина v', такая что $\mathcal{I}, v' \models \varphi$
- $\mathcal{I}, v \models \mathbf{EX}\varphi \Leftrightarrow$ существует вершина v', достижимая из v в дереве по одной дуге, такая что $\mathcal{I}, v' \models \varphi$
- $\mathcal{I}, v \models \mathbf{AX}\varphi \Leftrightarrow$ для каждой вершины v', достижимой из v в дереве по одной дуге, верно $\mathcal{I}, v' \models \varphi$

Логика деревьев вычислений

В логике деревьев вычислений модальности $\Box$, $\Diamond$ обозначаются записями **AG** и **EF**

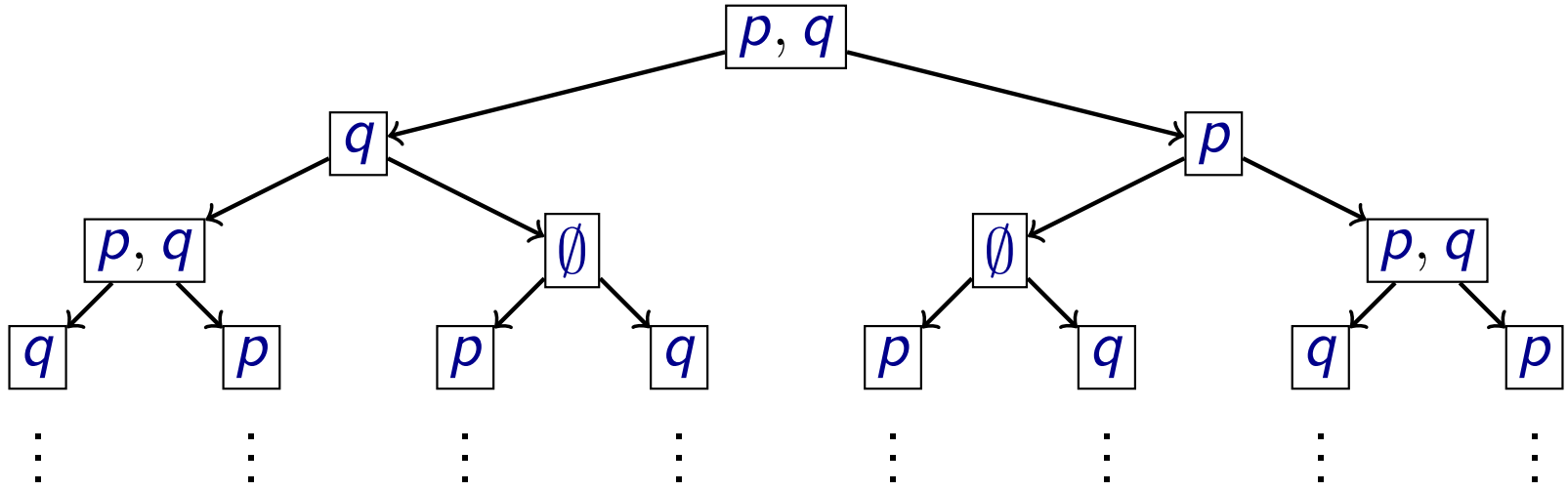
CTL содержит и другие модальности: **EG**, **AF**, **EX**, **AX**, **EU**, **AU**

Значение этих модальностей определяется так:

- $\mathcal{I}, v \models \varphi \mathbf{EU} \psi \Leftrightarrow$ существует ветвь дерева, исходящая из v, содержащая вершину v', такую что
  - $\mathcal{I}, v' \models \psi$ и
  - для каждого предка v'' вершины v', начиная с v, верно $\mathcal{I}, v'' \models \varphi$
- $\mathcal{I}, v \models \varphi \mathbf{AU} \psi \Leftrightarrow$ для каждой ветви дерева, исходящей из v, существует вершина v', такая что
  - $\mathcal{I}, v' \models \psi$ и
  - для каждого предка v'' вершины v', начиная с v, верно $\mathcal{I}, v'' \models \varphi$

Логика деревьев вычислений

Пример: рассмотрим такую CTL-интерпретацию $\mathcal{I}$:



Справедливы следующие соотношения:

$\mathcal{I}, \square \models p \,\&\, \mathbf{EX}p$

$\mathcal{I}, \square \models \mathbf{AXEX}(p \,\&\, q)$

$\mathcal{I}, \square \models \mathbf{EF}\neg p$

$\mathcal{I}, \square \models \mathbf{EG}p$

$\mathcal{I}, \square \models (q \to p)\mathbf{EU}(\neg p \,\&\, \neg q)$

$\mathcal{I}, \square \models \mathbf{AGEF}(p \,\&\, q)$

$\mathcal{I}, \square \not\models \mathbf{AX}p$

$\mathcal{I}, \square \not\models \mathbf{EXAX}(p \,\&\, q)$

$\mathcal{I}, \square \not\models \mathbf{AF}\neg p$

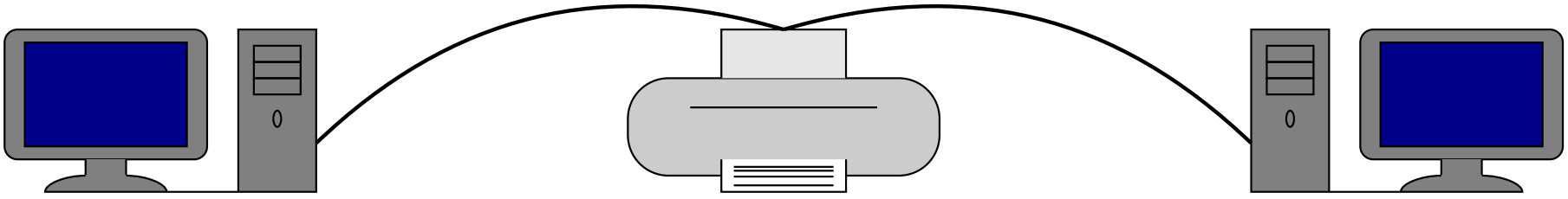
$\mathcal{I}, \square \not\models \mathbf{AG}p$

$\mathcal{I}, \square \not\models (q \to p)\mathbf{AU}(\neg p \,\&\, \neg q)$

$\mathcal{I}, \square \models \mathbf{AGAF}(p \,\&\, q \vee \neg p \vee \neg q)$

Выразительные возможности CTL

Вернёмся к примеру с сетевым принтером, для простоты оставив в нём только два компьютера:



Можно ли сформулировать какие-либо разумные требования к этой системе на языке CTL?

Начнём формализацию требований с описания элементарных событий:

- $busy_i$: принтер занят i-м компьютером
- try_i: i-му компьютеру требуется доступ к принтеру
- $print_i$: i-й компьютер передаёт данные на принтер

Выразительные возможности CTL

Теперь можно формализовать, например, такие требования:

- принтер не может быть занят
  двумя компьютерами одновременно
  $$\neg \mathbf{EF}(busy_1 \,\&\, busy_2)$$
- (*первый*) компьютер передаёт данные на принтер только тогда,
  когда принтер занят этим компьютером
  $$\mathbf{AG}(print_1 \to busy_1)$$
- если компьютеру требуется доступ к принтеру,
  то рано или поздно доступ будет получен
  $$\mathbf{AG}(try_1 \to \mathbf{AF}\, busy_1)$$
- принтер, занятый компьютером,
  начнёт печать с этого компьютера в тот же сеанс связи
  $$\mathbf{AG}(\neg busy_1 \to \mathbf{AX}(busy_1 \to (busy_1 \mathbf{AU} print_1)))$$
- как бы ни работала система,
  всегда есть возможность освободить принтер
  $$\mathbf{AG}(busy_1 \vee busy_2 \to \mathbf{EF}(\neg busy_1 \,\&\, \neg busy_2))$$

Выразительные возможности CTL

CTL — это формальный язык описания **требований**

Чтобы уметь формально верифицировать систему, необходимо

- иметь адекватную **формальную модель** системы и
- уметь **проверять** выполнимость CTL-формул в модели

Попробуем предложить такой формализм описания систем, чтобы

- поведение модели системы представляло собой темпоральную модель Крипке
- поведение реальной системы достаточно точно соответствовало поведению модели системы
- проверка выполнимости CTL-формул в модели была разрешима

Самая известный такой формализм —

размеченные системы переходов

LTSs (**L**abelled **T**ransition **S**ystems)

Размеченные системы переходов

Размеченная система переходов (LTS) над множеством элементарных событий $\mathcal{P}$ — это система $(S, S_0, \to, \rho)$, где:

- S — непустое множество состояний вычисления
- $S_0 \subseteq S$ — непустое множество начальных состояний
- $\to$: $S \times S$ — **тотальное** отношение переходов
- $\rho : S \to 2^{\mathcal{P}}$ — функция разметки

Тотальность отношения переходов означает, что из любого состояния вычисления s можно совершить переход: $\exists s'(s \to s')$

Функция разметки сопоставляет каждому состоянию вычисления s события $\rho(s)$, происходящие в этом состоянии

Если быть точным, то это особый вид системы переходов: структура Крипке (*Kripke structure*)

Размеченные системы переходов

Развёртка системы M, порождаемая состоянием s системы M — это CTL-интерпретация $U(s) = (W, R, \xi)$ следующего вида:

- состояния интерпретации имеют вид (s, t), где $s \in S$ и $t \in \mathbb{N}$
- $\xi(s, t) = \rho(s)$
- дерево T, основывающее интерпретацию $\mathcal{I}$, имеет следующий вид:
  - корень T — это состояние $(s_0, 0)$
  - дуги, исходящие из вершины (s, t), ведут во все вершины $(s', t + 1)$, такие что $s \to s'$

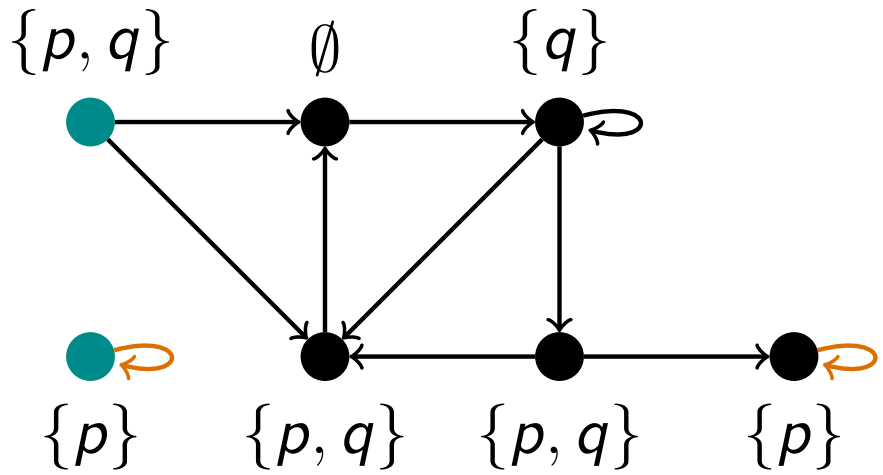
CTL-формула φ выполняется на LTS M ($M \models \varphi$), если для каждого начального состояния s_0 системы M справедливо соотношение $U(s_0), (s_0, 0) \models \varphi$

Размеченные системы переходов

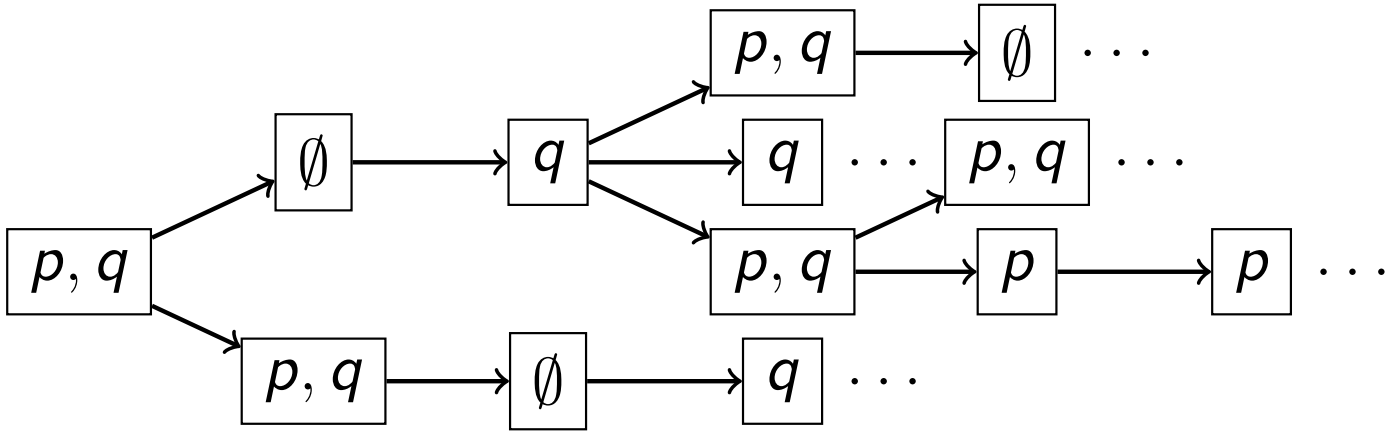
Пример

$\mathcal{P} = \{p, q\}$

LTS $(\{\ldots, \bullet, \ldots\}, \{\ldots, \bullet, \ldots\}, \to, \rho)$:



Развёртка системы, порождённая верхним $\bullet$:



Размеченные системы переходов

Как построить LTS, описывающую распределённую систему?

Начнём с простого: покажем, как можно построить LTS для последовательной программы π

- состояния LTS — это состояния вычисления программы π
  - (состояние вычисления включает в себя состояние управления и состояние данных)
- начальные состояния LTS — это всевозможные состояния, с которых может начинаться вычисление π
- переход в LTS соответствует шагу вычисления программы π
  - (например, $\langle \pi,\ \theta \rangle \to_I \langle \pi',\ \theta' \rangle$)
- элементарное событие соответствует отношению над значениями переменных и состояния управления программы
- событие включается в метку состояния LTS $\Leftrightarrow$ состояние вычисления удовлетворяет соответствующему отношению

Размеченные системы переходов

Пример

Вернёмся ещё раз к примеру с сетевым принтером

Предположим, что в контроллере принтера есть однобитовый регистр **R**, доступный на чтение и запись всем компьютерам в сети:

$$\mathbf{R} = \mathbf{true} \quad \Leftrightarrow \quad \text{принтер свободен}$$

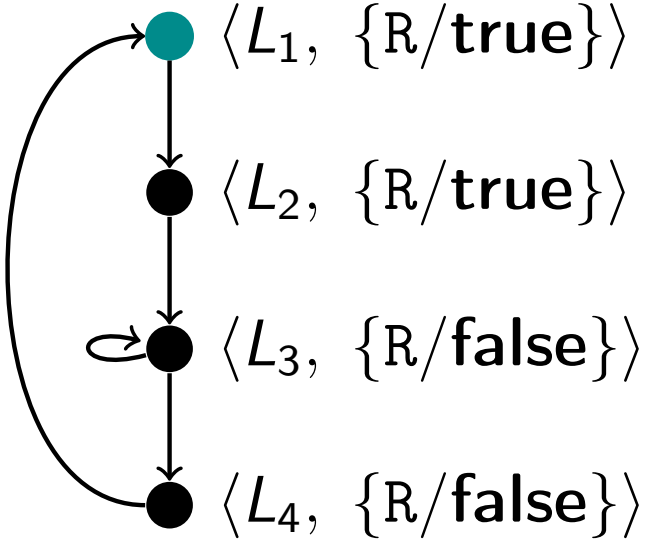
Тогда программа π взаимодействия компьютера с принтером может выглядеть так:

$$\begin{array}{l}
\mathbf{while}\ (\mathbf{true})\ \{ \\
\quad L_1 : \mathbf{while}\ (!R); \\
\quad L_2 : R = \mathbf{false}; \\
\quad L_3 : SEND_DATA \\
\quad L_4 : R = \mathbf{true}; \\
\}
\end{array}$$

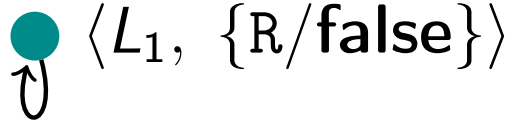
Размеченные системы переходов

Пример

LTS для программы π будет выглядеть так:



(функция разметки опущена)



Размеченные системы переходов

LTS и окружение программы

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

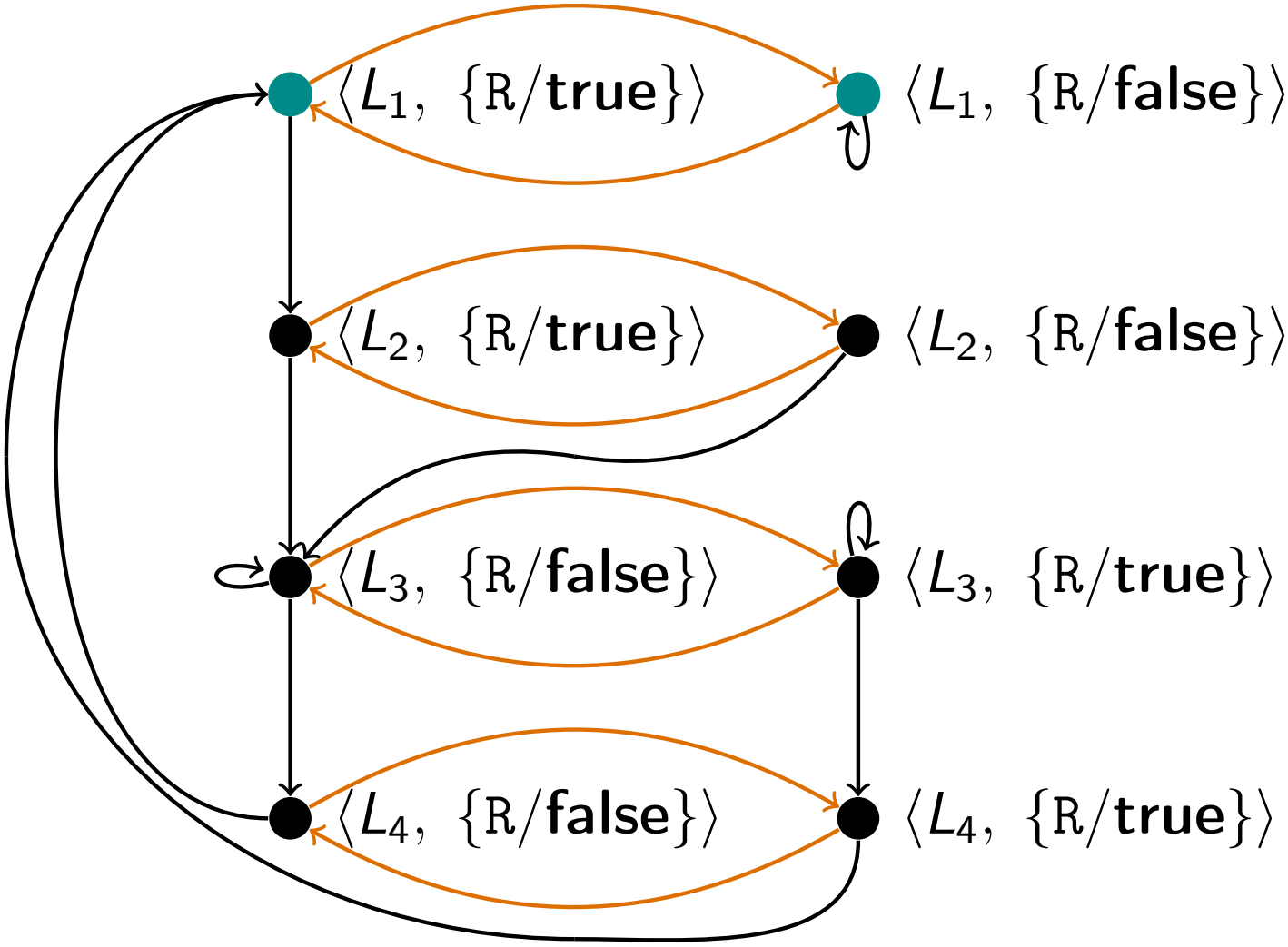
Такое взаимодействие выражается в том, что состояние вычисления программы может измениться (под воздействием её **окружения**)

Например, регистр **R** в последнем примере может быть изменён любым компьютером сети

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

Размеченные системы переходов

Пример: LTS для программы π с окружением, способным произвольно переключать значение регистра **R**



Размеченные системы переходов

LTS и взаимодействие программ

Предположим, что для программ π_1, π_2 построены LTSs и что эти программы взаимодействуют между собой

Как построить LTS, описывающую взаимодействие π_1, π_2?

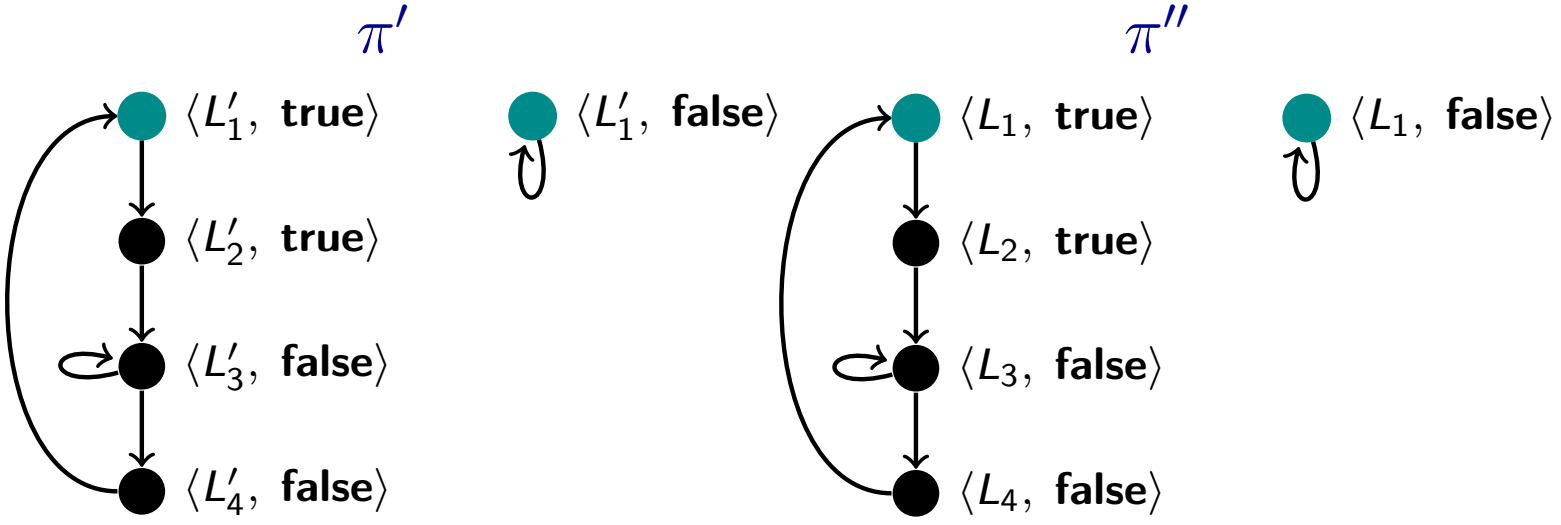
Один из наиболее популярных способов описания взаимодействия программ — это
семантика чередующихся вычислений: (*interleaving semantics*)

- совокупное состояние вычисления — это состояния π_1, π_2, в которых общие переменные изменяются синхронно (и учтены другие взаимосвязи состояний, если они есть)
- переход в LTS, описывающей взаимодействие программ, состоит в (произвольном) выборе одной из программ π_1, π_2 и выполнении следующего оператора этой программы

Размеченные системы переходов

Пример

Рассмотрим LTSs, описывающие две (одинаковые) программы взаимодействия с сетевым принтером:

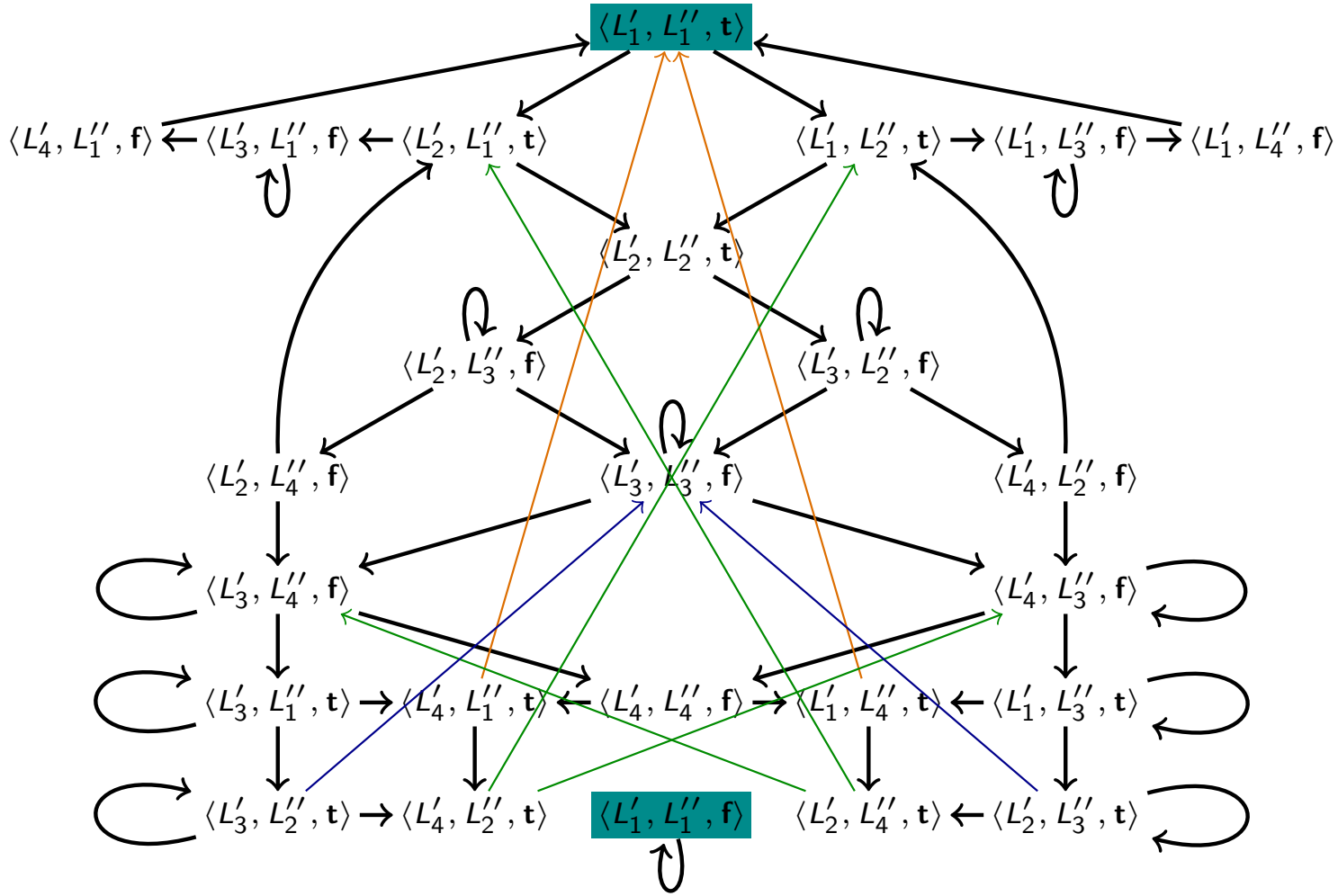


Считаем регистр **R общей переменной** программ π', π''

Размеченные системы переходов

Пример

LTS, описывающая взаимодействие π', π'' согласно **семантике чередующихся вычислений**, выглядит так:



Постановка задачи проверки моделей[1] для CTL

для заданных LTS M и CTL-формулы φ
проверить справедливость соотношения $M \models \varphi$

Теперь мы знаем,

- как строить LTSs, адекватно описывающие поведение распределённых систем
- как описывать требования к распределённой системе на языке CTL

Осталось показать, как **проверять** выполнимость CTL-формул на размеченных системах переходов

---

[1] Model checking

Законы логики ветвящегося времени

Утверждение об основных законах CTL

Для любой LTS M и любых формул φ, ψ справедливы следующие равносильности:

- $M \models \mathbf{AX}\varphi \Leftrightarrow M \models \neg\mathbf{EX}\neg\varphi$
- $M \models \mathbf{AG}\varphi \Leftrightarrow M \models \neg\mathbf{EF}\neg\varphi$
- $M \models \mathbf{AF}\varphi \Leftrightarrow M \models \neg\mathbf{EG}\neg\varphi$
- $M \models \mathbf{EF}\varphi \Leftrightarrow M \models \mathbf{true}\mathbf{EU}\varphi$
- $M \models \varphi\mathbf{AU}\psi \Leftrightarrow M \models \mathbf{AF}\psi \,\&\, \neg((\neg\psi)\mathbf{EU}(\neg\varphi \,\&\, \neg\psi))$

Доказательство. Самостоятельно

Значит, достаточно уметь проверять выполнимость CTL-формул, содержащих логические связки и операторы

EX, **EG**, **EU**

Алгоритм проверки моделей для CTL

Вход: LTS $M = (S, S_0, \to, \rho)$, CTL-формула φ

Выход: $M \overset{?}{\models} \varphi$

$M|_s$, где $s \in S$ — это LTS, получающаяся из M заменой множества S_0 на $\{s\}$

Этапы алгоритма:

1. преобразовать формулу φ в формулу ψ, не содержащую операторов **AX**, **AG**, **AF**, **EF**, **AU** и связок $\to$, &, при помощи утверждения об основных законах CTL и законов булевой алгебры
2. свести задачу к ряду проверок для LTS с одним начальным состоянием:

$$M \models \psi \Leftrightarrow \text{для любого } s_0 \in S_0 \text{ верно } M|_{s_0} \models \psi$$

Алгоритм проверки моделей для CTL

Вход: LTS $M = (S, S_0, \to, \rho)$, CTL-формула φ

Выход: $M \overset{?}{\models} \varphi$

Этапы алгоритма:

3. Проверить каждое соотношение $M|_{s_0} \models \psi$ следующей процедурой:

```
bool Check(M, s0, ψ) {
  if(ψ ∈ P) return φ ∈ ρ(s0);
  if(ψ is ¬χ) return !Check(M, s0, χ);
  if(ψ is χ1 ∨ χ2) return Check(M, s0, χ1)
                       || Check(M, s0, χ2);
  if(ψ is EXχ) return CheckEX(M, s0, χ);
  if(ψ is EGχ) return CheckEG(M, s0, χ);
  if(ψ is χ1EUχ2) return CheckEU(M, s0, χ1, χ2);
}
```

Алгоритм проверки моделей для CTL

Вход: LTS $M = (S, S_0, \to, \rho)$, CTL-формула φ

Выход: $M \overset{?}{\models} \varphi$

Утверждение. $M|_s \models \mathbf{EX}\psi \Leftrightarrow$ **существует состояние** s', **такое что** $s \to s'$ **и** $M|_{s'} \models \psi$

```
bool CheckEX(M, s, ψ) {
  V_ψ = {s' | s' ∈ S && Check(M, s', ψ)};
  return ∃s': s → s' && s' ∈ V_ψ;
}
```

Алгоритм проверки моделей для CTL

Вход: LTS $M = (S, S_0, \to, \rho)$, CTL-формула φ

Выход: $M \stackrel{?}{\models} \varphi$

Утверждение. $M|_s \models \psi_1 \mathbf{EU} \psi_2 \Leftrightarrow$ **из** s **исходит конечный путь, такой что**

- **он оканчивается в вершине** s'**, такой что** $M|_{s'} \models \psi_2$
- **для всех вершин** s'' **пути, кроме последней, верно** $M|_{s''} \models \psi_1$

```
bool CheckEU(M, s, ψ, χ) {
  V_ψ = {s' | s' ∈ S && Check(M, s', ψ)};
  V_χ = {s' | s' ∈ S && Check(M, s', χ)};
  return ∃s_0 → s_1 → ··· → s_k :
    s_0 = s && s_k ∈ V_χ && {s_0, ..., s_{k-1}} ⊆ V_ψ;
}
```

Алгоритм проверки моделей для CTL

Вход: LTS $M = (S, S_0, \to, \rho)$, CTL-формула φ

Выход: $M \stackrel{?}{\models} \varphi$

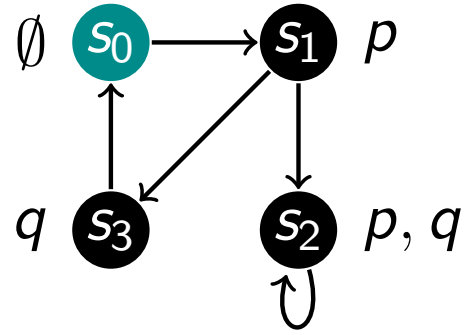
Утверждение. $M|_s \models \mathbf{EG}\psi \Leftrightarrow$ **в системе** M **существует конечный путь** $s \to \cdots \to s' \to \cdots \to s'$**, такой что для каждой вершины** s'' **этого пути верно** $M|_{s''} \models \psi$

```
bool CheckEG(M, s, ψ) {
  V_ψ = {s' | s' ∈ S && Check(M, s', ψ)};
  return ∃s_0 → s_1 → ··· → s_k :
    s_0 = s && {s_0, ..., s_k} ⊆ V_ψ
    && ∃i: (0 ≤ i < k && s_i = s_k);
}
```

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

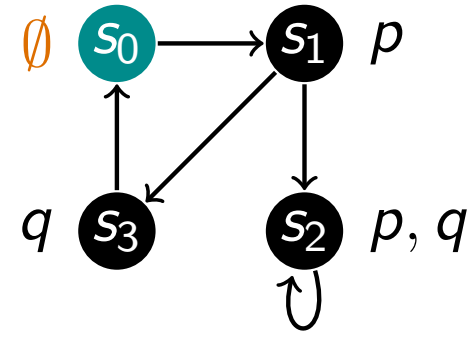
M:



Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:

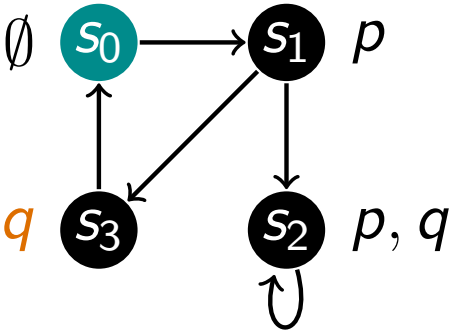


$M|_{s_0} \models \neg p \,\&\, \neg q$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:



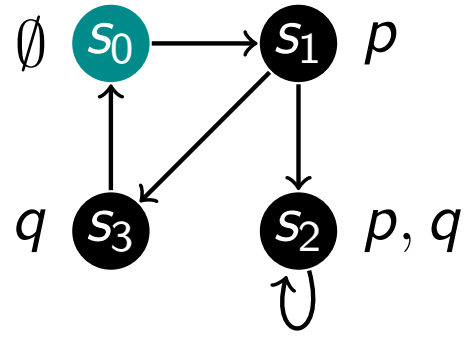
$M|_{s_0} \models \neg p \,\&\, \neg q$

$M|_{s_3} \models \neg p$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:



$M|_{s_0} \models \neg p \,\&\, \neg q$

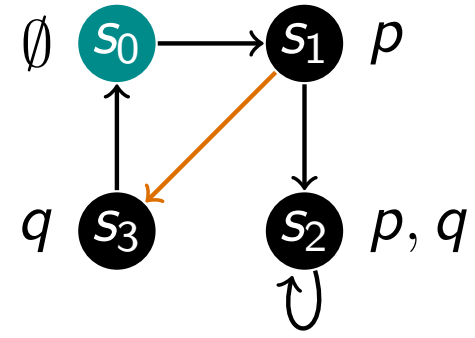
$M|_{s_3} \models \neg p$

$M|_{s_3} \models \neg p \vee \mathbf{EX}\neg p$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:



$M|_{s_0} \models \neg p \,\&\, \neg q$

$M|_{s_3} \models \neg p$

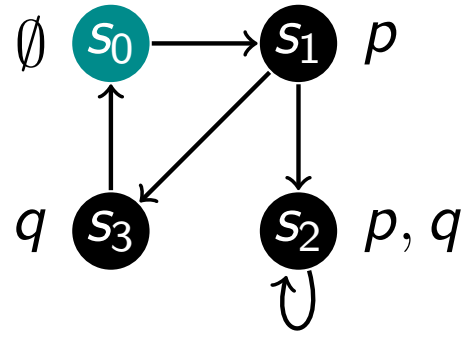
$M|_{s_3} \models \neg p \vee \mathbf{EX}\neg p$

$M|_{s_1} \models \mathbf{EX}\neg p$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:



$M|_{s_0} \models \neg p \,\&\, \neg q$

$M|_{s_3} \models \neg p$

$M|_{s_3} \models \neg p \vee \mathbf{EX}\neg p$

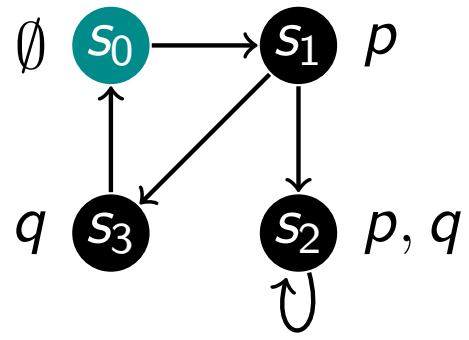
$M|_{s_1} \models \mathbf{EX}\neg p$

$M|_{s_1} \models \neg p \vee \mathbf{EX}\neg p$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:



$M|_{s_0} \models \neg p \,\&\, \neg q$

$M|_{s_3} \models \neg p$

$M|_{s_3} \models \neg p \vee \mathbf{EX}\neg p$

$M|_{s_1} \models \mathbf{EX}\neg p$

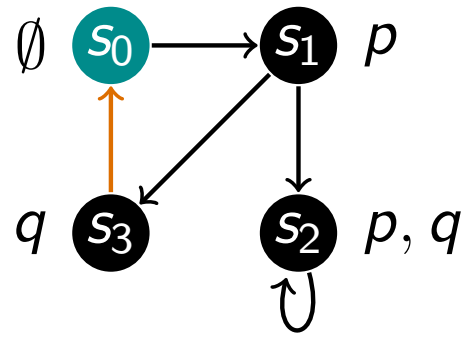
$M|_{s_1} \models \neg p \vee \mathbf{EX}\neg p$

$M|_{s_0} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:



$M|_{s_0} \models \neg p \,\&\, \neg q$

$M|_{s_3} \models \neg p$

$M|_{s_3} \models \neg p \vee \mathbf{EX}\neg p$

$M|_{s_1} \models \mathbf{EX}\neg p$

$M|_{s_1} \models \neg p \vee \mathbf{EX}\neg p$

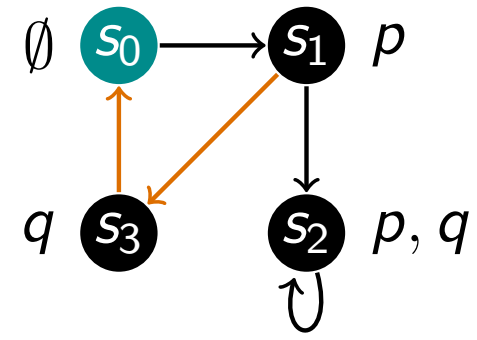
$M|_{s_0} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$

$M|_{s_3} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:



$M|_{s_0} \models \neg p \,\&\, \neg q$

$M|_{s_3} \models \neg p$

$M|_{s_3} \models \neg p \vee \mathbf{EX}\neg p$

$M|_{s_1} \models \mathbf{EX}\neg p$

$M|_{s_1} \models \neg p \vee \mathbf{EX}\neg p$

$M|_{s_0} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$

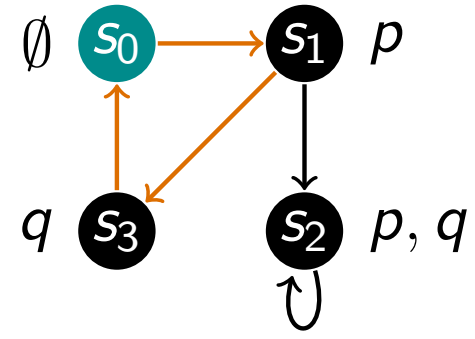
$M|_{s_3} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$

$M|_{s_1} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:

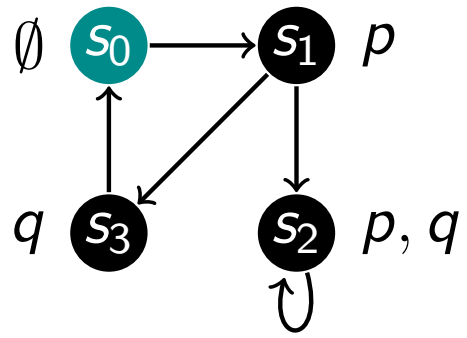


$$M|_{s_0} \models \neg p \,\&\, \neg q$$
$$M|_{s_3} \models \neg p$$
$$M|_{s_3} \models \neg p \vee \mathbf{EX}\neg p$$
$$M|_{s_1} \models \mathbf{EX}\neg p$$
$$M|_{s_1} \models \neg p \vee \mathbf{EX}\neg p$$
$$M|_{s_0} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$$
$$M|_{s_3} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$$
$$M|_{s_1} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$$
$$M|_{s_0} \models \varphi$$

Алгоритм проверки моделей для CTL

Пример: $\varphi = \mathbf{EG}((\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q))$

M:



$$M|_{s_0} \models \neg p \,\&\, \neg q$$
$$M|_{s_3} \models \neg p$$
$$M|_{s_3} \models \neg p \vee \mathbf{EX}\neg p$$
$$M|_{s_1} \models \mathbf{EX}\neg p$$
$$M|_{s_1} \models \neg p \vee \mathbf{EX}\neg p$$
$$M|_{s_0} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$$
$$M|_{s_3} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$$
$$M|_{s_1} \models (\neg p \vee \mathbf{EX}\neg p)\mathbf{EU}(\neg p \,\&\, \neg q)$$
$$M|_{s_0} \models \varphi$$

Вопрос на понимание:

а какова сложность предложенного алгоритма?

Конец лекции 16–17