

Верификация параллельных программных и аппаратных систем



Курс лекций

Шошмина Ирина Владимировна
Карпов Юрий Глебович



План курса

1. Введение
2. Метод Флойда-Хоара доказательства корректности программ
3. Исчисление взаимодействующих систем (CCS) Р.Милнера
4. Темпоральные логики
5. Автоматный подход к проверке выполнения формул LTL
6. Система верификации Spin и язык Promela. Примеры верификации
7. Алгоритм model checking для проверки формул CTL
8. BDD и их применение
9. Символьная проверка моделей
10. Структура Крипке как модель реагирующих систем
11. Темпоральные свойства систем
12. Применения метода верификации model checking
13. Количественный анализ дискретных систем при их верификации
14. Верификация систем реального времени
15. Консультации по курсовой работе

Верификация реактивных систем:

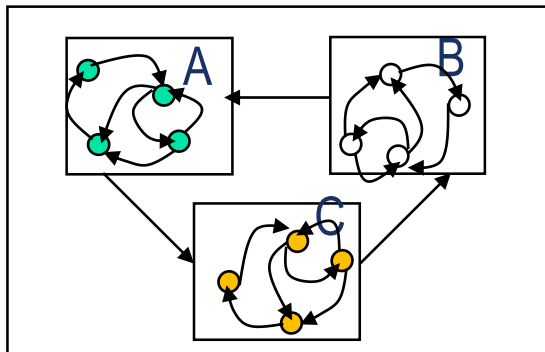
State explosion problem

$$2^{10} \sim 10^3$$

Классические алгоритмы верификации реактивных систем могут работать только с игрушечными системами – число состояний реальных систем очень велико

Обычная техника Model checking работает с системами $\sim 10^6$ - 10^7 состояний – представление каждого состояния требует ~ 50 - 100 байт

Пример:
три процесса,
три канала связи



Простая система, а
число состояний -
миллионы миллиардов!!

Каждая подсистема – 4 состояния + 1 целая переменная (short, 1 байт) + каналы для передачи целых (байт) значений.

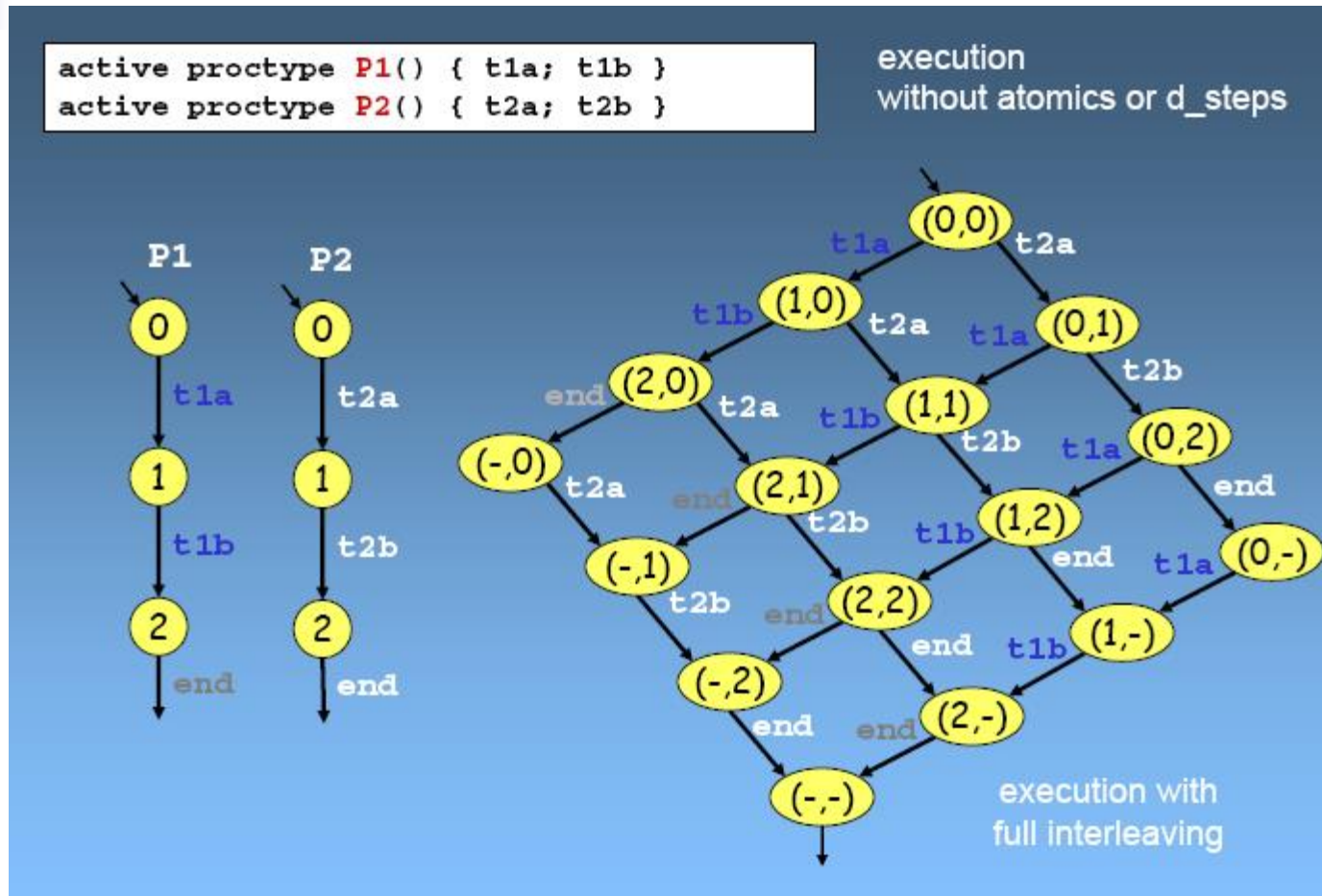
Пусть для простоты очередь каждом из 3-х каналов не больше 1 сообщения

Всего $(4 \times 2^8)^3 \times (2^8)^3 = 2^{54} \approx 10^{16}$ глобальных состояний системы - **State**

Explosion Problem. Система в своей жизни проходит ничтожную долю своих возможных состояний, но мы не знаем, какие именно!! Число частиц во

Вселенной $\sim 10^{78}$

State explosion problem: интерливинг



- Два параллельных процесса с тремя состояниями каждый: экспоненциальный рост числа глобальных состояний для анализа.



Верификация методом Model checking

- Алгоритмы Model checking весьма эффективны: сложность проверки выполнимости формулы CTL Φ на структуре Крипке K пропорциональна числу подформул Φ и сложности K
- Какой сложности структуру Крипке можно разместить в памяти современного компьютера при явном представлении состояний?
- ~ 50 байт на состояние
- Каждое отношение – множество пар, ~ 100 байт на пару
- Число пар в отношении – порядка квадрата числа состояний
- Вся память пусть 1 терабайт = 10^{12} байт
- Вывод: явное представление состояний структуры Крипке позволяет работать с такими структурами, содержащими $\sim 10^6$ состояний

Структуры Крипке реальных систем содержат $10^{100} - 10^{200}$ состояний

Что делать?

Методы борьбы со “state explosion problem”:

1. Символьная верификация
2. Редукция частичных порядков
3. Композициональная верификация



Что делать?

Использовать специальные
эффективные методы представления
подмножеств конечных множеств и
операций над ними

Этот подход называется
“СИМВОЛЬНОЙ ВЕРИФИКАЦИЕЙ”



Символьный подход и BDD: борьба с проблемой взрыва числа состояний

Model Checking – эффективная техника верификации, но есть недостатки. Число состояний при введении дополнительных параметров и/или компонент исследуемой системы растет экспоненциально: “State explosion problem”

Конечные математические структуры (множества, отношения, ...) и операции над ними могут быть представлены логическими функциями и булевыми операциями над этими функциями.

В свою очередь, БФ можно экономно представить в BDD.

Поэтому BDD можно использовать **в любых алгоритмах над конечными структурами** и алгоритмы эти получаются очень эффективными

В 1992 г. все это было осознано в Carnegi Mellon University – были представлены методы задания структуры Крипке и алгоритмы (Model checking) - анализа темпоральных свойств структур Крипке с помощью BDD. Так был разработан метод “СИМВОЛЬНОЙ ВЕРИФИКАЦИИ”

J.Burch, E.Clarke, K.McMillan et.al. [Symbolic model checking: 10²⁰ states and beyond.](#) [Information and Computation](#), v.98, N2,1992

BDD – Бинарные решающие диаграммы (Binary Decision Diagrams)

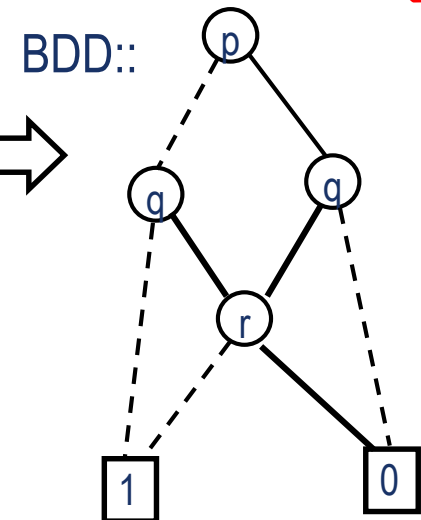
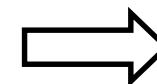
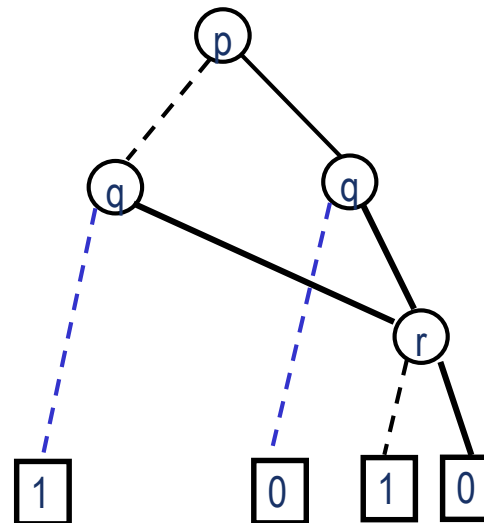
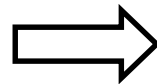
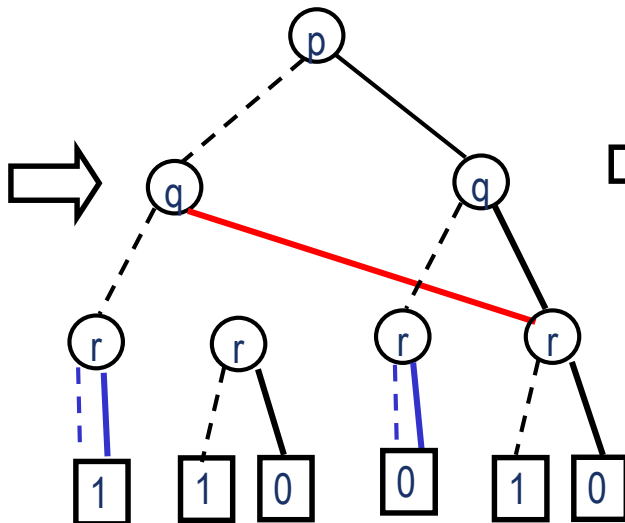
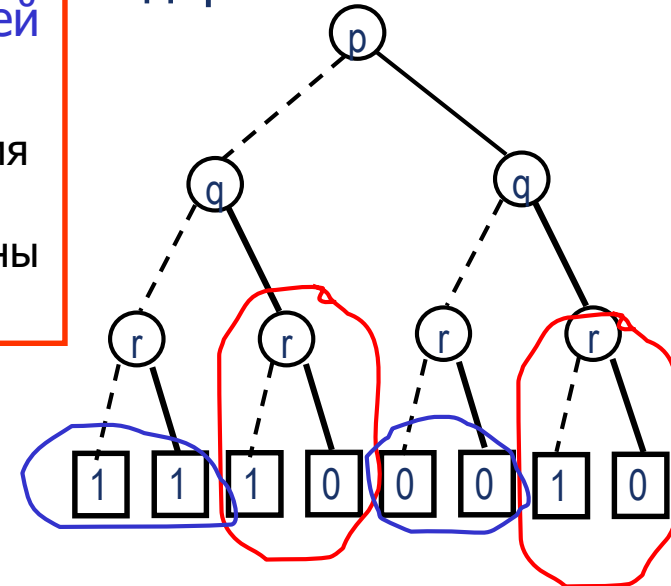
BDD-это семантическое дерево БФ без избыточностей

BDD – ациклический орграф, в котором отсутствуют повторения в структуре, с одной корневой вершиной, двумя листьями, помеченными 0 и 1, и промежуточными вершинами. Корневые и промежуточные вершины помечены переменными, из них выходят ровно два ребра

Binary – потому что двоичные переменные

Decision – потому что решения принимаются при направленном движении по графу (диаграмме)

Семантическое
дерево:





Свойства BDD

1. BDD – **каноническое** представление – **любая** БФ имеет *единственное BDD-представление* при заданном порядке переменных

2. Сложность BDD зависит от порядка переменных.

Пример: функция $(a_1 \oplus b_1) \& \dots \& (a_n \oplus b_n)$

при порядке $a_1 \dots a_n b_1 \dots b_n$ имеет сложность $3 \times 2^{n-1}$

при порядке $a_1 b_1 a_2 b_2 \dots a_n b_n$ имеет сложность $3 \times n + 2$

3. Проблема нахождения оптимального порядка NP – трудна. Но есть эвристики

4. BDD для многих функций имеет линейную сложность.

Некоторые классы функций имеют экспоненциальную сложность BDD *при любом* порядке переменных (пример: функция, выдающая средний бит результата произведения $A \times B$ n -разрядных переменных A и B)

5. Булевы операции над БФ, представленными в BDD, просты.

Теорема. Сложность выполнения булевых операций над двумя функциями f и g , представленными в BDD, полиномиальна: $O(|f| \times |g|)$



С BDD введен новый класс алгоритмов в МС

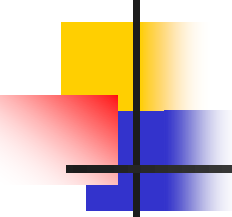
- **Явные:** классические алгоритмы обычно оперируют с явным представлением дискретных объектов, **перебирая их последовательно**, “один за другим” (**explicit representation**)
- Основанные на BDD алгоритмы называют символьными (“**symbolic**”) или неявными (“**implicit**”). ПОЧЕМУ?
- **Символьные**
 - для манипулирования объектами и отношениями вводятся дополнительные булевы переменные, которые можно считать “дополнительными” параметрами, или “**символами**”
- **Неявные:**
 - алгоритмы, основанные на BDD, работают с **неявным** представлением (**implicit representation**) конечных множеств и отношений объектов с помощью БФ, представленных в форме BDD
 - размер представления МНОЖЕСТВ растет не линейно, а логарифмически, операции для всех объектов множества выполняются каждая за один шаг для всего множества, а не последовательно для каждого элемента множества



Верификация систем с числом состояний 10^{20}

Явные алгоритмы model checking: до 10^6 состояний, ~ 100 состояний в секунду

- Пусть для запоминания одного состояния нужно только 10 байт
- Тогда запоминание 10^{20} состояний требует *тысячу миллионов терабайт*
- Пусть скорость перебора при выполнении алгоритмов верификации обычная \sim сотня состояний в секунду
- Тогда перебор 10^{20} состояний с помощью явных (обычных) алгоритмов *model checking* потребует *сотен миллиардов лет*
- **ВЫВОД:** без использования новых (СИМВОЛЬНЫХ) методов хранения данных и манипулирования ими с помощью BDD верификация реальных систем была бы невозможна
- **MODEL CHECKING** позволяет оперировать множествами с числом элементов $\sim 10^{400}$ и более!!! (Baier, Katoen. Principles of Model checking, 2008)

- 
-
- Как BDD используются для задания булевых функций, используемых в МС?
 - Вспомним, что такое характеристические функции множеств и отношений

Характеристические функции множества

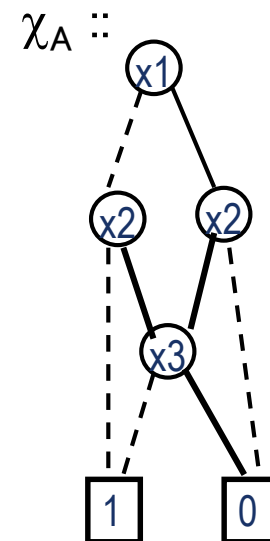
Пусть $S = \{a_0, \dots, a_8\}$ и $A = \{a_2, a_4, a_5\}$ – подмножество S . Закодируем элементы S двоичными кодами. Тогда $A = \{010, 100, 101\}$

Обозначим χ_A **характеристическую функцию** множества A . Она равна 1 на наборах, кодирующих элементы из A , т.е. на $\{010, 100, 101\}$, и 0 на всех остальных

Пусть x_1, x_2, x_3 – разряды кодировки,

Тогда $\chi_A = \neg x_1 x_2 \neg x_3 \vee x_1 \neg x_2 \neg x_3 \vee x_1 \neg x_2 x_3$
задает $A = \{010, 100, 101\}$

Итак, подмножества конечного множества можно задавать логической формулой, представляющей характеристическую функцию => Для задания характеристической функции BDD



Будем писать $A = \{010, 100, 101\} (x_1, x_2, x_3)$, чтобы показать переменные кодировки и их порядок



Операции над множествами с помощью BDD

Нульарные операции (константы):

- полное множество: $\chi_S = \text{True}$
- пустое множество: $\chi_\emptyset = \text{False}$

Унарная операция:

- дополнение множества: $\chi_{S-Q} = \neg \chi_Q$

Бинарные операции:

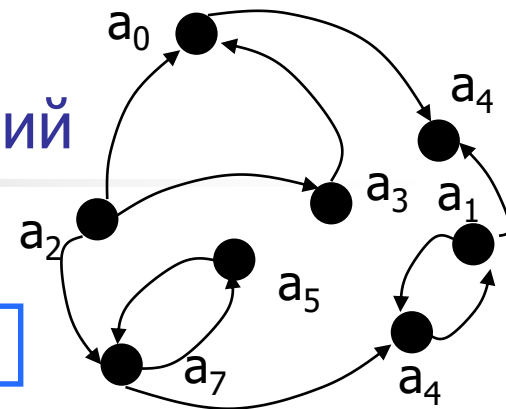
- пересечение множеств: $\chi_{P \cap Q} = \chi_P \wedge \chi_Q$
- объединение множеств: $\chi_{P \cup Q} = \chi_P \vee \chi_Q$
- разность множеств: $\chi_{P-Q} = \chi_P \wedge \neg \chi_Q$

Все операции над множествами можно выразить через булевы операции над характеристическими булевыми функциями:

Символьное представление отношений

Пусть $S = \{a_0, \dots, a_{n-1}\}$ – множество

Введем кодирование: $a_0 \Leftrightarrow 000, a_1 \Leftrightarrow 001, \dots, a_7 \Leftrightarrow 111$



Бинарное отношение на S – подмножество пар из S , $R = \{ (a_2, a_3), (a_4, a_7), (a_5, a_7), \dots \}$

Первый элемент отношения R – это текущая вершина (pre-state), разряды ее кода обозначим $v = (x_1, x_2, x_3, \dots)$.

Второй элемент отношения – (post-state) и ее код – $v' = (x_1', x_2', x_3', \dots)$

χ_R – характеристическая функция R равна 1 на кодировках $\{(a_0, a_4), (a_2, a_0), (a_2, a_7) \dots\}$, т.е. на наборах (v, v') 6 булевых переменных $\{ 000 \ 100, 010 \ 000, 010 \ 111, \dots \}$

$$\begin{aligned} \chi_R = & \neg x_1 \neg x_2 \neg x_3 \neg x_1' x_2' x_3' \vee \\ & \neg x_1 x_2 \neg x_3 \neg x_1' \neg x_2' \neg x_3' \vee \\ & \neg x_1 x_2 \neg x_3 x_1' x_2' x_3' \vee \dots \end{aligned}$$

Характеристическая функция отношения обычно определяется как логическая формула над штрихованными и нештрихованными двоичными переменными

Ее можно представить с помощью BDD

Синтаксис: CTL (грамматика):

$$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid AX\phi \mid EX\phi \mid AG\phi \mid \\ EG\phi \mid AF\phi \mid EF\phi \mid A[\phi_1 U \phi_2] \mid E[\phi_1 U \phi_2]$$

State formulas: Каждая CTL формула – это формула состояний, которая в каждом состоянии либо истинна, либо ложна

Поэтому можно считать, что любая CTL формула ϕ описывает множество состояний, в которых она истинна:

$$Q_\phi = \{ s \in S \mid s \models \phi \},$$

т.е. Q_ϕ можно определить как множество таких состояний s из S , на которых эта формула истинна

Для CTL формулы ϕ будем ОПРЕДЕЛЯТЬ характеристическую функцию множества, на котором ϕ истинно



Символьная верификация

Задача верификации: Даны структура Крипке K и CTL-формула ϕ . Найти множество всех таких состояний s , для которых верно: $K, s \models \phi$, т.е. те, для которых выполняется ϕ . Если начальное состояние принадлежит s , то $K \models \phi$,

Алгоритм Model checking сводится к нахождению *подмножеств* состояний K , для которых выполняются *подформулы* формулы ϕ

Строим булевы функции в форме BDD для:

- (а) множества начальных состояний K ;
- (б) множества переходов K
- (в) множеств состояний K , в которых истинен каждый атомарный предикат

Символьная верификация: Все подмножества состояний будем представлять булевыми функциями в форме BDD, т.е. для каждой CTL- формулы ϕ будем строить в форме BDD характеристическую булеву функцию f_ϕ , задающую то множество состояний структуры Крипке K , на которых ϕ выполняется



Как построить структуру Крипке в BDD

- Структуры Крипке реальных систем содержат 10^{100} состояний
- Для представления подмножеств состояний нужны БФ от ~ 300 двоичных переменных, а для представления отношений нужны БФ от ~ 600 переменных, что при представлении БФ в форме BDD вполне выполнимо
- Но как их построить?

КАК представить структуру Крипке в форме BDD без построения самих состояний и переходов явно?

НУЖНО булевыми функциями задать только начальное состояние (множество начальных состояний) и переходы

Как задавать характеристические функции: Проблема волк, коза и капуста

Состояние – вектор значений переменных

Булевы переменные – f, w, g, c.

Начальное состояние $\langle 0,0,0,0 \rangle$; $\chi_{s_0} = \sim f \sim w \sim g \sim c$

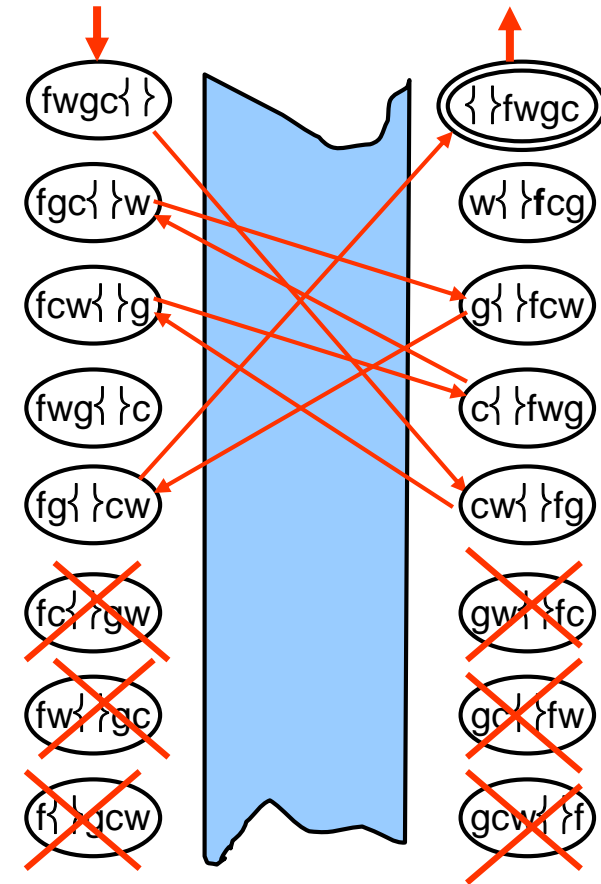
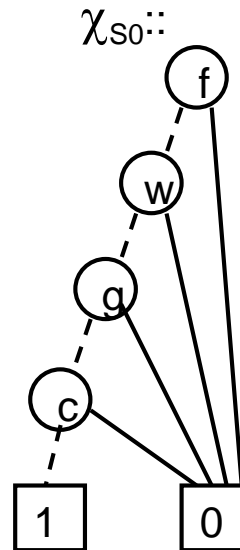
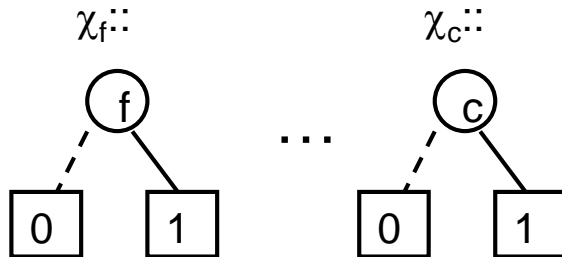
Атомарные предикаты:

$\chi_f = f$;

$\chi_w = w$;

$\chi_g = g$;

$\chi_c = c$



Переходы:

а) сам фермер может поехать

б) может взять один предмет (волка, козу или капусту)

Проблема волк, коза и капуста: Представление переходов в BDD

Переходы:

- сам может переправиться или взять с собой не более одного объекта

R1 : сам едет $\chi_{R1}(v, v') =$
 $(f \equiv \sim f')(w \equiv w')(g \equiv g')(c \equiv c')$

R2 : везет волка $\chi_{R2}(v, v') =$
 $(f \equiv w)(f \equiv \sim f')(w \equiv \sim w')(g \equiv g')(c \equiv c')$

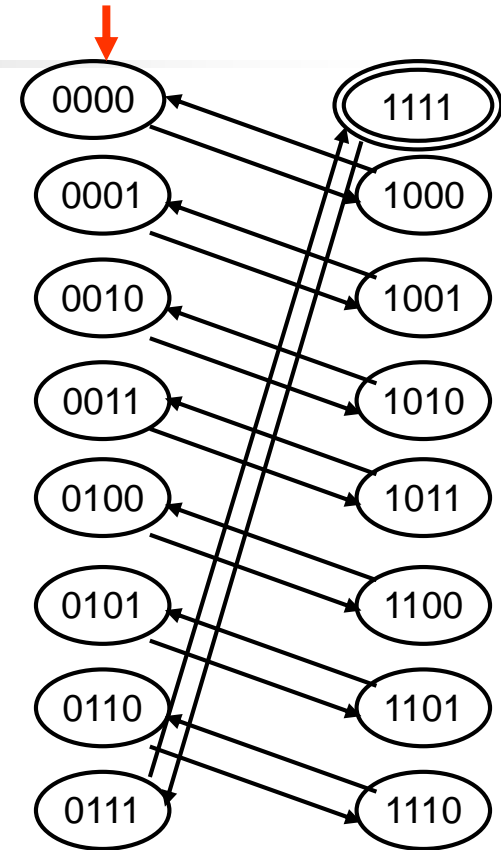
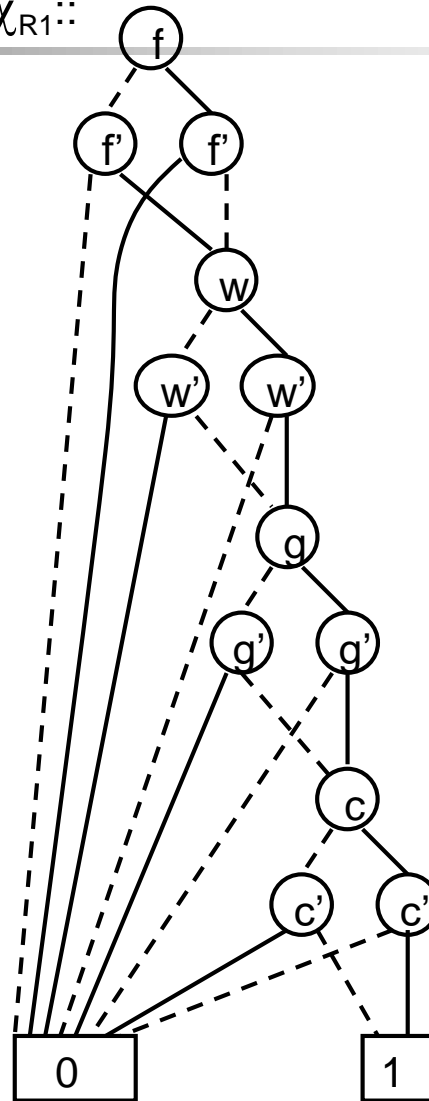
R3 : везет козу $\chi_{R3}(v, v') =$
 $(f \equiv g)(f \equiv \sim f')(w \equiv w')(g \equiv \sim g')(c \equiv c')$

R4 : везет капусту $\chi_{R4}(v, v') =$
 $(f \equiv c)(f \equiv \sim f')(w \equiv w')(g \equiv g')(c \equiv \sim c')$

$$R = R1 \vee R2 \vee R3 \vee R4$$

Но явно проводить ребра НЕ НАДО!
 Они все описываются характеристической функцией

$\chi_{R1}::$

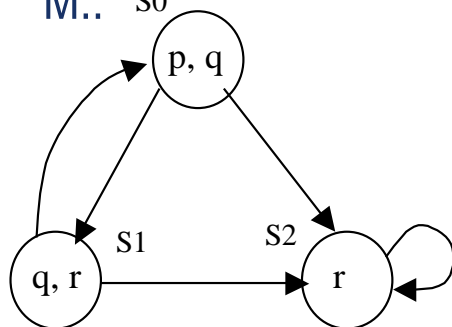


Все 16 переходов структуры Крипке, соответствующие R1, задаются булевой формулой

χ_R

Пример. Представление структуры Крипке в BDD

M:: s0



$M=(S, s_0, R, L)$ Переменные: $v=\langle x_1, x_2 \rangle$, $v'=\langle x_1', x_2' \rangle$

Кодирование состояний: $s_0 \Leftrightarrow 00$, $s_1 \Leftrightarrow 01$, $s_2 \Leftrightarrow 10$ (v)

Множество $S = \{00, 01, 10\}$ (v); $s_0 = \{00\}$ – подмножество S

Множество $R = \{0001, 0010, 0100, 0110, 1010\}$ (v, v')



Функция пометок $L: S \rightarrow 2^{AP}$

- Свойство $p = \{00\}$ (v) – в состоянии s_0
- Свойство $q = \{00, 01\}$ (v) – в состояниях s_0, s_1
- Свойство $r = \{01, 10\}$ (v) – в состояниях s_1, s_2

Характеристические функции :

Все эти функции представляем в BDD

$$\chi_{s_0} = \neg x_1 \neg x_2$$

$$\chi_{R(v,v')} = \neg x_1 \neg x_2 (x_1' \oplus x_2') \vee \neg x_1 x_2 \neg x_2' \vee x_1 \neg x_2 x_1' \neg x_2'$$

$$\chi_p(v) = \neg x_1 \neg x_2$$

$$\chi_q(v) = \neg x_1$$

$$\chi_r(v) = x_1 \oplus x_2$$

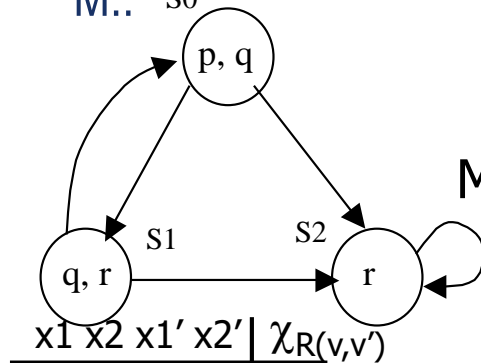
Структура Крипке → характеристическая функция отношения R

M:: s₀

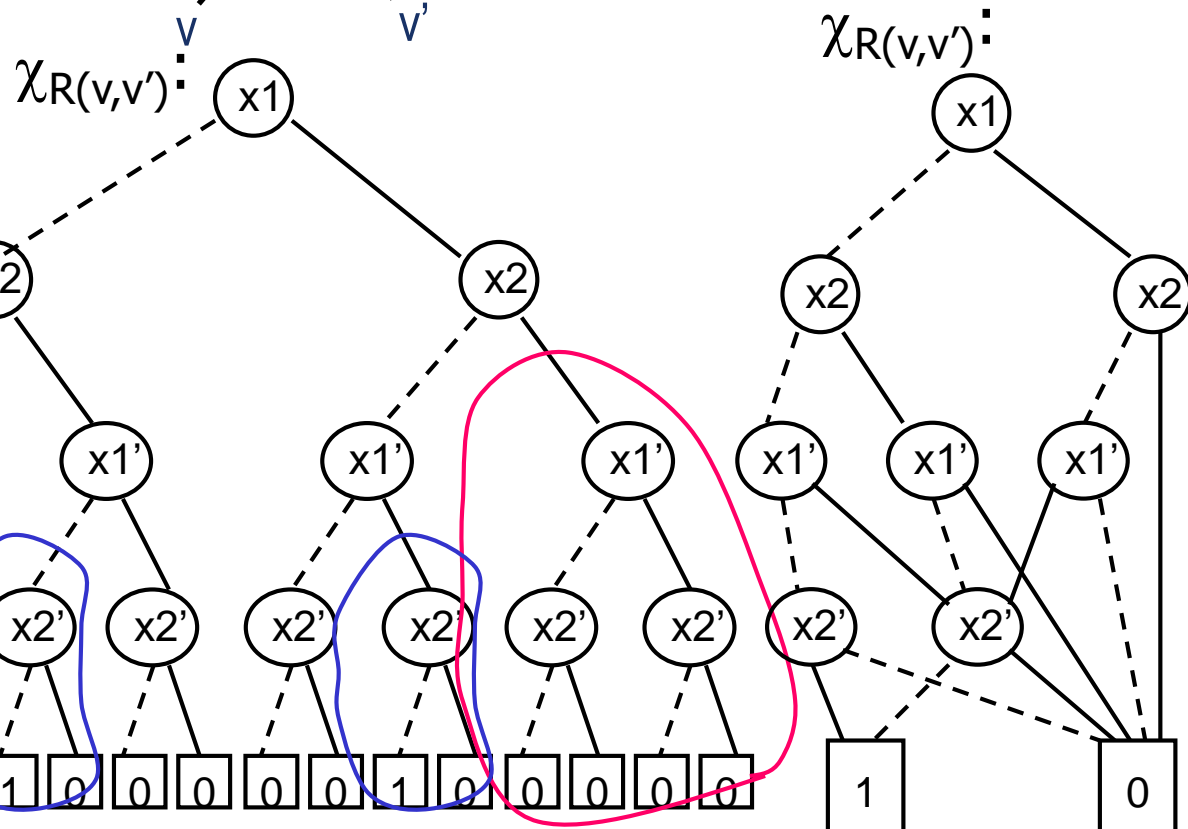
$$v = \langle x_1, x_2 \rangle, v' = \langle x_1', x_2' \rangle$$

Кодирование состояний: $s_0 \Leftrightarrow 00, s_1 \Leftrightarrow 01, s_2 \Leftrightarrow 10$ (v)

Множество R = {0001, 0010, 0100, 0110, 1010} (v, v')



x ₁	x ₂	x ₁ '	x ₂ '	$\chi_{R(v,v')}$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0





Вычисление операторов базиса: p , $\neg\phi$, $\phi_1 \vee \phi_2$

Один из темпоральных базисов:

$EX \phi$ – из данного состояния есть ребро в состояние, помеченное ϕ

$EG \phi$ – из данного состояния есть путь, все состояния которого помечены ϕ

$E[\phi U \psi]$ – из данного состояния существует путь, на котором есть состояние, помеченное ψ , а все состояния до него помечены ϕ

С и н т а к с и с CTL:

$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid EX\phi \mid EG\phi \mid E[\phi_1 U \phi_2]$

Рассмотрим формулы базиса p , $\neg\phi$, $\phi_1 \vee \phi_2$:

p – для каждого атомарного предиката p функция $\chi_{Q,p}$, задающая множество состояний, в которых p истинен, определяется в задании структуры Крипке

$\neg\phi$ - операция отрицания очевидным образом выполняется над характеристической функцией $\chi_{Q,\phi}$, задающей то множество состояний структуры Крипке, на котором выполняется формула ϕ

$\psi \vee \phi$ - операция дизъюнкции очевидным образом выполняется над соответствующими характеристическими функциями $\chi_{Q,\psi}$ и $\chi_{Q,\phi}$

Алгоритм Model Checking: базис {EX, AF, EU}

```
for all  $0 < i \leq |\Phi|$  do          /* для всех уровней синтаксического дерева */
  for all  $\Psi \in \text{Sub}(\Phi)$  with  $|\Psi| = i$  do /* для всех подформул уровня i */
    switch ( $\Psi$ ):                  /* для каждого типа подформул – свой алгоритм */
      p      :       $\text{Sat}_\Psi := \{ s \in S \mid p \in L(s) \}$ ;
       $\neg p$    :       $\text{Sat}_\Psi := \{ s \in S \mid p \notin L(s) \}$ ;
       $p \wedge q$  :       $\text{Sat}_\Psi := \{ s \in S \mid p, q \in L(s) \}$ ;
      EXp     :       $\text{Sat}_\Psi := \text{Sat\_EX}(p)$ ;
      AFp     :       $\text{Sat}_\Psi := \text{Sat\_AF}(p)$ ;
      E(pUq)  :       $\text{Sat}_\Psi := \text{Sat\_EU}(p, q)$ ;
    end switch /*  $\text{Sat\_EX}, \text{Sat\_AF}$  и т.п. – функции,
                                   определенные далее */
    for all  $s \in \text{Sat}_\Psi$  do  $L(s) := L(s) \cup \{p_\Psi\}$  od
      /* Все состояния из  $\text{Sat}_\Psi$  помечаем новым атомарным предикатом  $p_\Psi$  */
    od;
  od;
od;
```

Алгоритм обрабатывает его все узлы синтаксического дерева формулы Φ .

Sat_Ψ - множество состояний, в которых выполняется формула Ψ

$L(s)$ - множество (атомарных) формул, которые выполняются в состоянии s

Если ϕ выполняется в НАЧАЛЬНЫХ состояниях M , то она выполняется на M

A directed graph with 10 nodes and 12 edges. Four nodes are highlighted in cyan and labeled 'p'. The graph shows a complex network of dependencies or relationships.

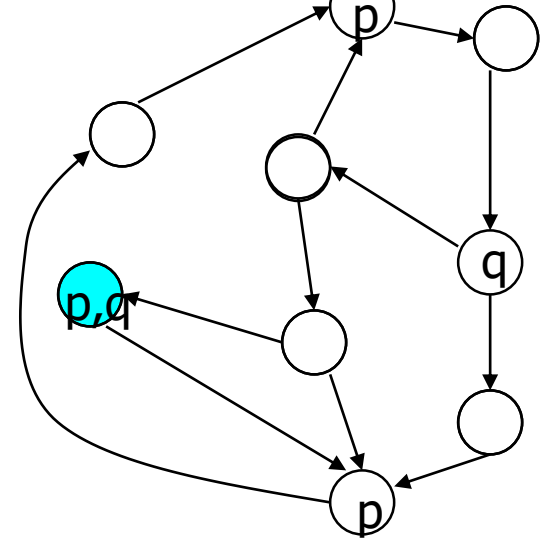
Если p введено в символьную разметку, то характеристическая функция здесь будет проста: $SAT_p = p$

Алгоритм Model Checking для CTL ($p \wedge q$)

Sat _{$p \wedge q$} := { $s \in S \mid p \in L(s) \wedge q \in L(s)$ } -

характеристическая функция множества состояний, помеченных p и q

```
function SAT_( $p \wedge q$ ) /*дает все s,  
                        в которых истинна p и q */  
  local var X,Y,Z;  
  begin  
    Y := {  $s \in S \mid p \in L(s)$  };  
    Z := {  $s \in S \mid q \in L(s)$  };  
    X := Y  $\wedge$  Z;  
  return X  
end
```

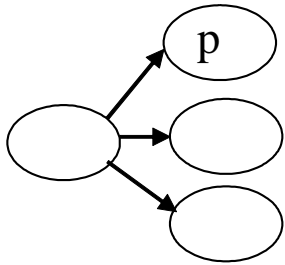


Все закрашенные
состояния попадают в
множество SAT_($p \wedge q$)

Используем операции над множествами для поиска
характеристической функции пересечения множеств \wedge

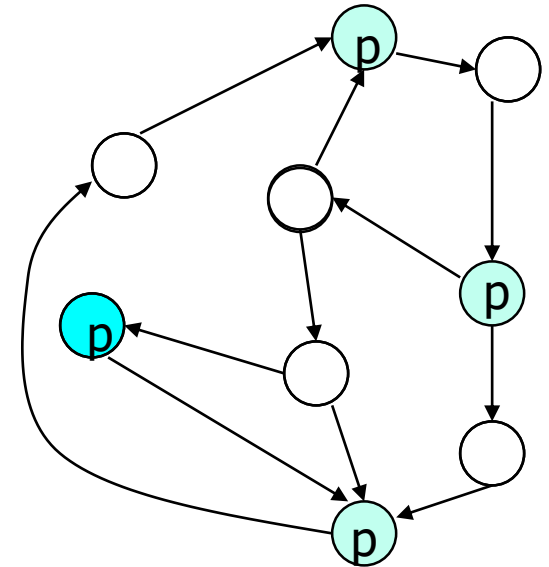
Алгоритм Model Checking для CTL (EX). Итерация 0

EXp: сначала находим состояния, помеченные p



было

```
function SAT_EX(p) /*дает все s,  
                    в которых истинна EXp */  
  local var Y;  
  begin  
    Y := { s | (∃s1 ∈ SAT (p)) s → s1 };  
  return Y  
end
```

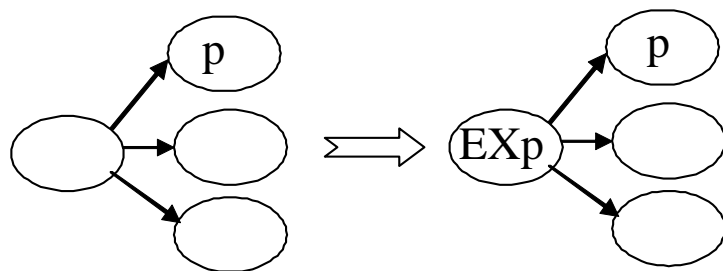


Все закрашенные
состояния попадают в
множество SAT_(p)

Перебираем все состояния структуры Крипке и смотрим, есть ли состояние, помеченное p; а потом надо рассмотреть предыдущее состояние.

Алгоритм Model Checking для CTL (EX). Итерация 1

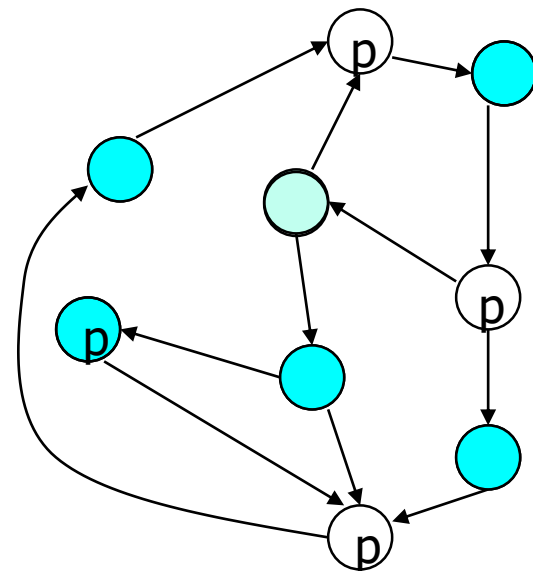
EXp: помечаем с меткой p_{EXp} если хотя бы один преемник с помечен p



было

стало

```
function SAT_EX(p) /*дает все s,  
                    в которых истинна EXp */  
  local var Y;  
  begin  
    Y := { s | (∃s1 ∈ SAT (p)) s → s1 };  
  return Y  
end
```

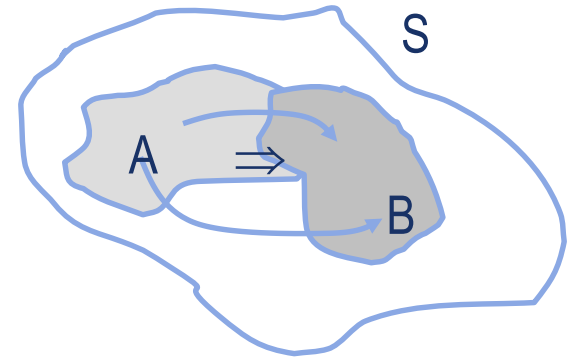


Все закрашенные
состояния попадают в
множество SAT_EX(p)

**Как найти такие состояния с помощью операций над
двоичными функциями?**

Прямой образ для бинарных отношений с помощью двоичных функций

Пусть $A \subseteq S$ – подмножество S ,
 R – бинарное отношение на S
В какие элементы S можно перейти из A ?



Ограничение отношения R на тех начальных элементах R из S , которые принадлежат A :

$$\chi_{A(v)} \& \chi_{R(v, v')}$$

$B = \text{Forward Image } (A, R) = \text{FI}(A, R)$ - Прямой Образ A относительно R :

$$\chi_{\text{FI}(A, R)(v)} = \exists v'. [\underbrace{\chi_{A(v)} \& \chi_{R(v, v')}}_{\text{Переходы из элементов } \in A}] \langle v / v' \rangle$$

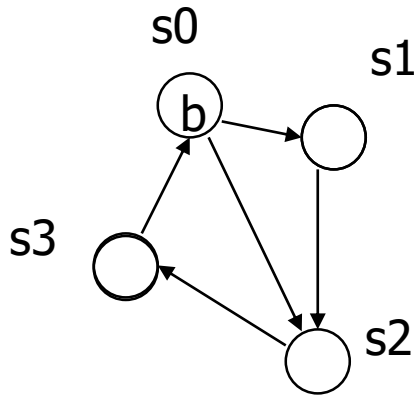
$\langle v / v' \rangle$ - это замена переменными v штрихованных значений v'

Чтобы найти множество B всех тех элементов S , которые достижимы за один шаг отношения R из элементов множества A , нужно вычислить прямой образ: несколько операций с булевыми характеристическими функциями $\chi_{A(v)}$ и $\chi_{R(v, v')}$

Пример: найти состояния, достижимые из s0 за 1 шаг?

Пусть $S = \{s_0, s_1, s_2, s_3\}$ – множество

Кодируем: $s_0 \Leftrightarrow 00, s_1 \Leftrightarrow 01, s_2 \Leftrightarrow 10, s_3 \Leftrightarrow 11$



$R = \{ (s_0, s_1), (s_0, s_2), (s_1, s_2), (s_2, s_3), (s_3, s_1) \}$

$$\chi_R = \neg x_1 \neg x_2 \neg x_1' x_2' \vee \neg x_1 \neg x_2 x_1' \neg x_2' \vee \neg x_1 x_2 x_1' \neg x_2' \vee x_1 \neg x_2 x_1' x_2' \vee x_1 x_2 \neg x_1' \neg x_2'$$

$\chi_A = \neg x_1 \neg x_2$ Характеристическая функция s0

Вычислим прямой образ

$$\chi_{FI(A,R)}(v) = \exists v'. [\chi_A(v) \& \chi_R(v, v')] \langle v/v' \rangle$$

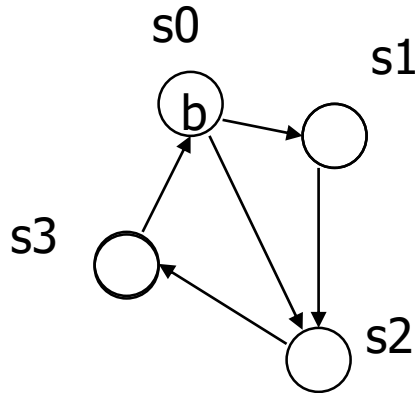
$$1. \chi_A(v) \& \chi_R(v, v') = \neg x_1 \neg x_2 \neg x_1' x_2' \vee \neg x_1 \neg x_2 x_1' \neg x_2'$$

$$\begin{aligned}
 2. \exists v. (\chi_A(v) \& \chi_R(v, v')) &= \\
 \exists x_1, \exists x_2. (\neg x_1 \neg x_2 \neg x_1' x_2' \vee \neg x_1 \neg x_2 x_1' \neg x_2') &= \\
 \exists x_1. (\neg x_1 \neg x_1' x_2' \vee \neg x_1 x_1' \neg x_2') &= \\
 = \neg x_1' x_2' \vee x_1' \neg x_2' &
 \end{aligned}$$

$$3. \exists v. [\chi_A(v) \& \chi_R(v, v')] \langle v/v' \rangle = \neg x_1 x_2 \vee x_1 \neg x_2$$

Пример: найти состояния, достижимые из s0 за 1 шаг?

Вычислим прямой образ



$$\chi_{FI(A,R)}(v) = \exists v'. [\chi_A(v) \& \chi_R(v, v')] \langle v/v' \rangle$$

$$1. \quad \chi_A(v) \& \chi_R(v, v') = \\ \neg x_1 \neg x_2 \neg x_1' x_2' \vee \neg x_1 \neg x_2 x_1' \neg x_2'$$

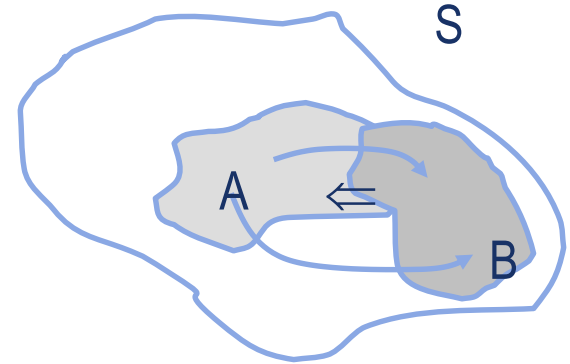
$$2. \quad \exists v. (\chi_A(v) \& \chi_R(v, v')) = \\ \exists x_1, \exists x_2. (\neg x_1 \neg x_2 \neg x_1' x_2' \vee \neg x_1 \neg x_2 x_1' \neg x_2') = \\ \exists x_1. (\neg x_1 \neg x_1' x_2' \vee \neg x_1 x_1' \neg x_2') = \\ = \neg x_1' x_2' \vee x_1' \neg x_2'$$

$$3. \quad \exists v. [\chi_A(v) \& \chi_R(v, v')] \langle v/v' \rangle = \neg x_1 x_2 \vee x_1 \neg x_2$$

Если х.ф. представлены в ДНФ, то прямой образ равносильен «отбрасыванию» нештрихованных переменных и замене штрихованных переменных на нештрихованные

Обратный образ для бинарных отношений с помощью двоичных функций

Пусть $B \subseteq S$ – подмножество S ,
 R – бинарное отношение на S
Из каких элементов S можно перейти в B ?



Ограничение отношения R на тех вторых элементах R из S , которые принадлежат B :

$$\chi_{B(v')} \& \chi_{R(v,v')}$$

$A = \text{Reverse Image}(B, R) = \text{RI}(B, R) =$
обратный образ B относительно R :

$\chi_{\text{RI}(B, R)}(v) = \exists v'. (\chi_{B(v')} \& \chi_{R(v, v')})$ – выбрасываем все вторые элементы в ДНФ

Чтобы найти множество A всех тех элементов S , из которых за один шаг отношения R достижимы элементы заданного множества B , нужно вычислить обратный образ: несколько операций с булевыми характеристическими функциями $\chi_{B(v')}$ и $\chi_{R(v, v')}$

Вычисление операторов базиса: $EX\phi$

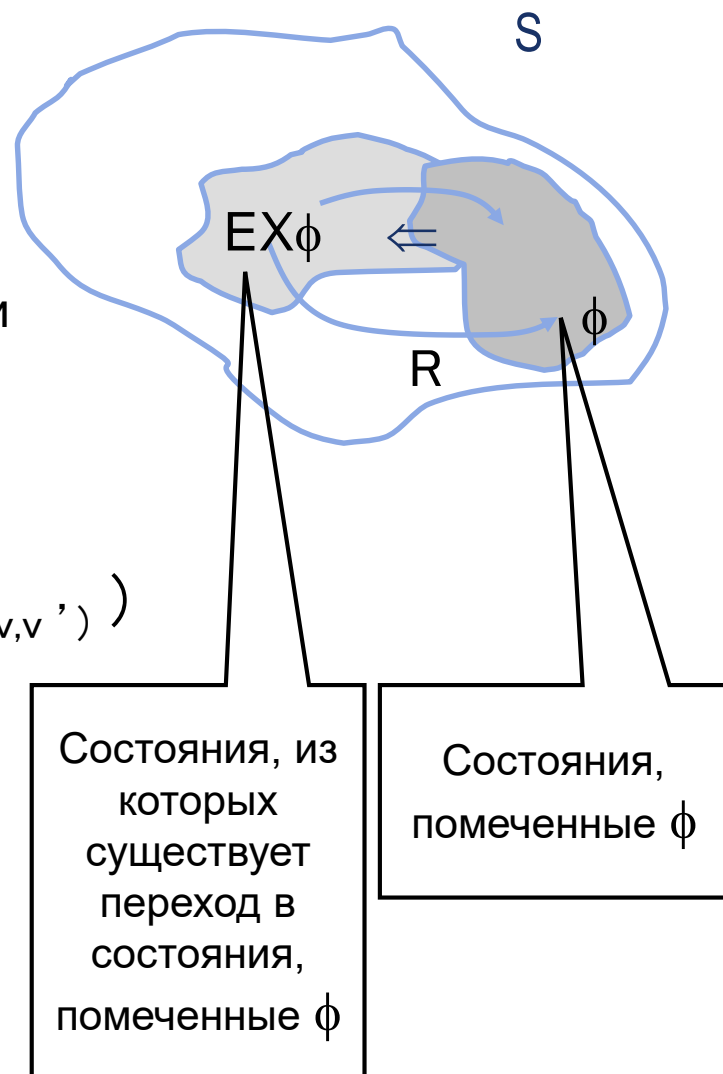
Синтаксис: CTL (для одного из базисов):

$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid EX_{\phi} \mid EG\phi \mid E[\phi_1 U \phi_2]$

Характеристическая булева функция множества $Q_{EX\phi}$ строится с помощью операции “Обратный Образ” над характеристическими БФ $\chi_{Q\phi}$ и χ_R , представляющими Q_{ϕ} и R

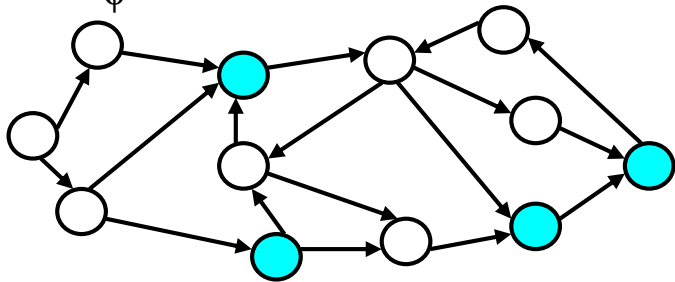
$$\chi_{RevImage(Q\phi, R)}(v) = \exists v'. (\chi_{Q\phi}(v) \langle v' / v \rangle \& \chi_R(v, v'))$$

Строим ‘обратный образ’ ограничения отношения R на множестве Q_{ϕ} с

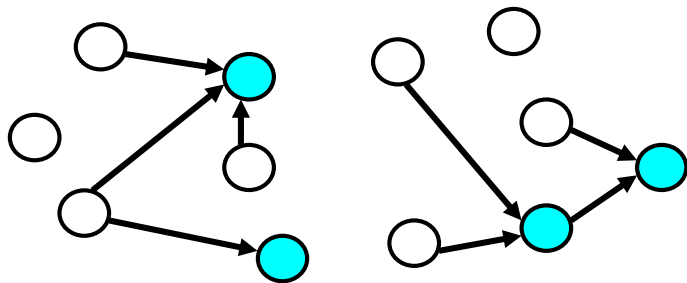


Пример вычисления оператора $EX\phi$

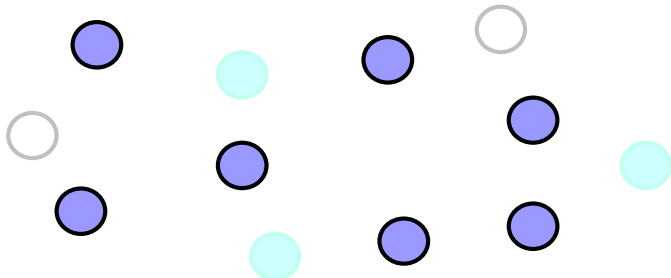
Q_ϕ и $R ::$



$$\chi = (\chi_{Q_\phi}(v) \langle v' / v \rangle) \& \chi_{R(v, v')}$$



$$Q_{EX\phi} = \exists v'. (\chi)$$



Состояния, удовлетворяющие ϕ , помечены (множество Q_ϕ определено). Отношение R задано

Строим ограничение отношения R на множестве Q_ϕ (оставляем только такие переходы, у которых вторая компонента - из множества Q_ϕ)

У оставшихся переходов берем только первую компоненту с помощью операции квантификации по переменным второй компоненты. Если отношение перехода представлено ДНФ, то операция квантификации существования равносильна «отбрасыванию» переменных в соответствующих слагаемых.

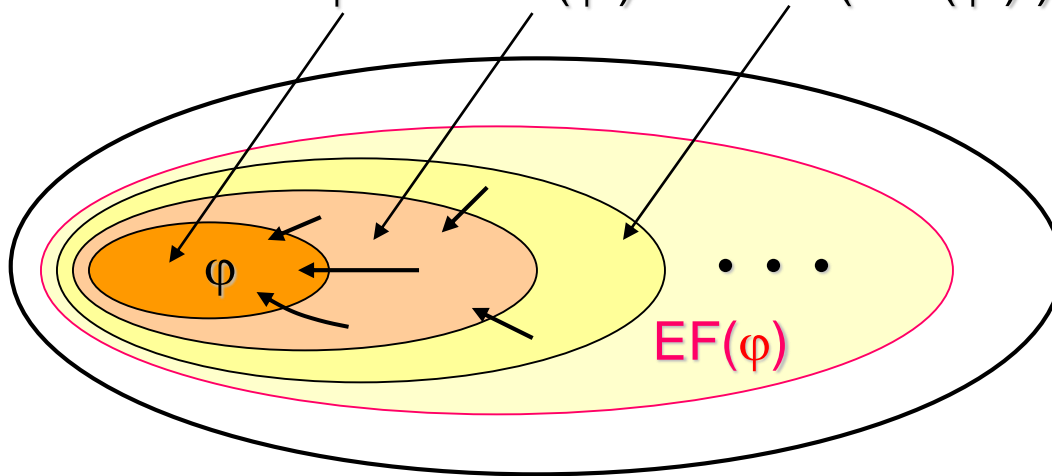
Так получаем множество $Q_{EX\phi}$

Всего несколько операций с BDD, представляющими Q_ϕ и R

Определение множества состояний, удовлетворяющих формуле $EF\varphi$

$$EF\varphi = \varphi \vee EX\varphi \vee EX EX\varphi \vee EX EX EX\varphi \vee \dots$$

$EF(\varphi) \equiv$ состояния из которых можно достичь φ - состояния,
помеченные: $\varphi \cup EX(\varphi) \cup EX(EX(\varphi)) \cup \dots$



$$S_0 = S_\varphi$$

$$S_1 = S_0 \cup EX S_0$$

$$S_2 = S_1 \cup EX S_1$$

...

$$S_{i+1} = S_i \cup EX S_i$$

Вычисление искоемых множеств состояний структуры Крипке состоит в повторяющихся преобразованиях этих множеств до тех пор, пока они не перестанут изменяться. Они называются НЕПОДВИЖНЫМИ точками

Вопросы:

всегда ли придем к пределу, или иногда будет циклический перебор подмножеств?
если пришли к пределу, будет ли это предельное множество тем, что мы хотели?

- Вопросы:
- всегда ли придем к пределу, или будет циклический перебор подмножеств, или всегда будем получать разные результаты?
- если пришли к пределу, будет ли это предельное множество тем, что мы хотели?

Ответ на эти вопросы дает теория неподвижной точки операторов на множествах

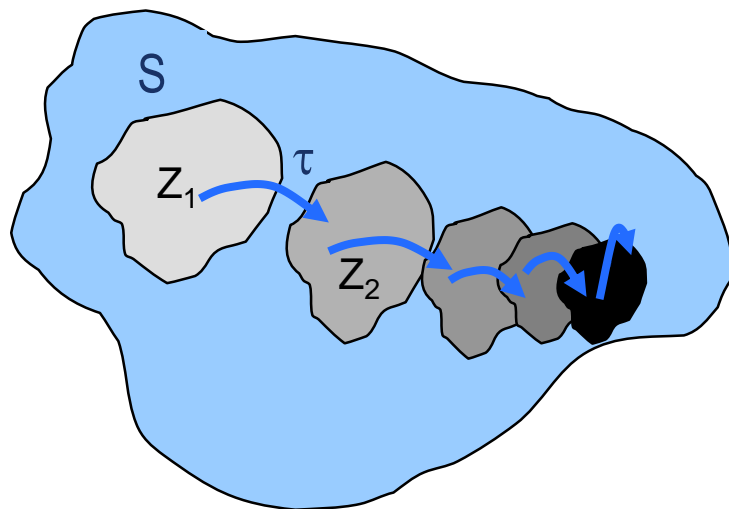
Теория фиксированной (неподвижной) точки нужна нам для обоснования алгоритмов для темпоральных операторов - $EF\phi$, $EG\phi$ и других

Теория неподвижной точки (Fixpoint, FP)

Пусть S – конечное множество, 2^S – совокупность его подмножеств

Оператор на S в S – это отображение $\tau: 2^S \rightarrow 2^S$, т.е. τ переводит подмножество в подмножество

$$\tau(Z_1) = Z_2$$



Что будет, если оператор τ на 2^S применить несколько раз?

$$\tau^0(Z) = Z$$

$$\tau^k(Z) = \tau \left(\underbrace{\tau \left(\dots \tau(Z) \right)}_{k \text{ раз}} \right)$$

Поскольку S – конечно, результат будет либо сходиться к одному и тому же множеству, либо к циклическому повторению множеств

Неподвижной точкой отображения τ называется такое $Z \subset S$, что $\tau(Z) = Z$

Операторы на конечных множествах

$$\tau_1(Q) = \{a, b\} \cup Q$$

$$\tau_1(\{a, b\}) = \tau_1^2(\{a, b\}) = \dots = \{a, b\} \text{ FP}$$

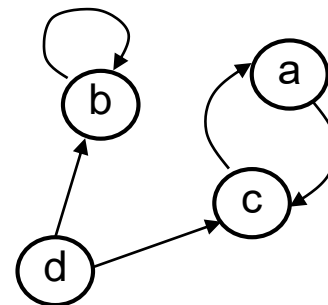
$$\tau_1(\{a, d\}) = \{a, b, d\}$$

$$\tau_1(\{a, b, c\}) = \{a, b, c\} \text{ FP}$$

$$\tau_1(S) = S = \{a, b, c, d\} \text{ FP}$$

$$\tau_1(\emptyset) = \{a, b\}; \tau_1(\tau_1(\emptyset)) = \tau_1^k(\emptyset) = \{a, b\}$$

$$S = \{a, b, c, d\}$$



У τ_1 *несколько* неподвижных точек

Операторы на конечных множествах

$$\tau_2(Q) = \{c\} \cup \{a\} \cap Q$$

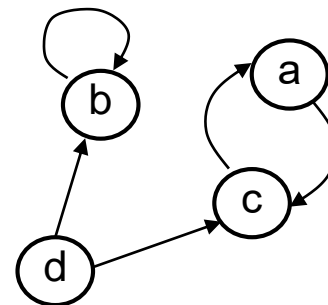
$$\tau_2(\{a\}) = \{a, c\}.$$

$$\tau_2(\{a, c\}) = \{a, c\} \text{ FP}$$

$$\tau_2(S) = \{a, c\}$$

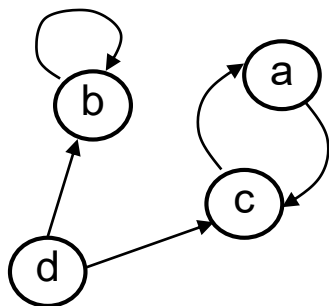
$$\tau_2(c) = \{c\} \text{ FP}$$

$$S = \{a, b, c, d\}$$



У τ_2 две неподвижные точки

Операторы на конечных множествах



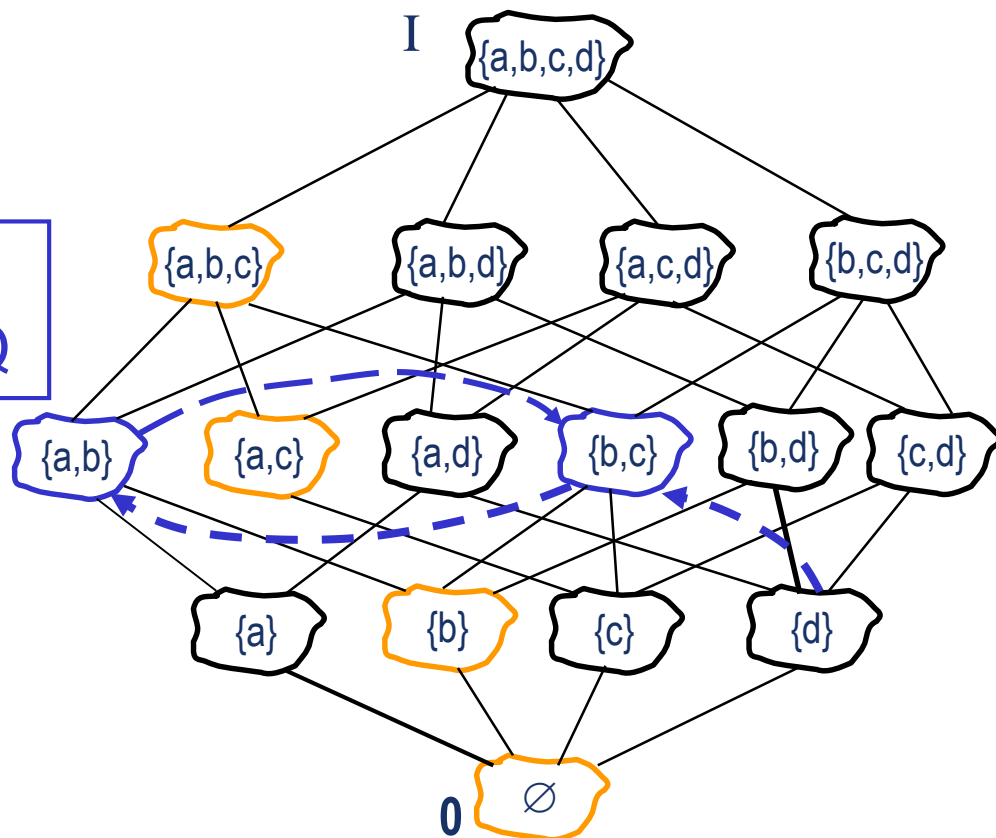
$S = \{a, b, c, d\}$

$\tau_3 (Q) = Q'$, где Q' –
преемники Q

$\tau_3 (\{d\}) = \{b, c\}$
 $\tau_3 (\{b, c\}) = \{a, b\}$
 $\tau_3 (\{a, b\}) = \{b, c\}$
 $\tau_3 (\{a, c\}) = \{a, c\}$ FP

Существуют множества, применение τ_3 к
которым приводит к циклическому повторению

Решетка всех
подмножеств S

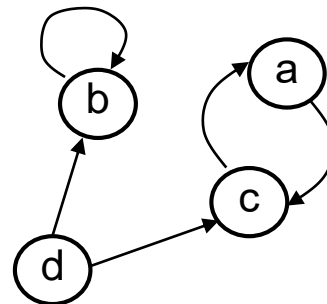


Операторы на конечных множествах

$$\tau_4 (Q) = Q - Q'$$

$$\begin{aligned}\tau_4 (\{d\}) &= \{d\} \text{ FP} \\ \tau_4 (\{c,d\}) &= \{c, d\} \text{ FP} \\ \tau_4 (\{a,c\}) &= \emptyset \\ \tau_4 (\emptyset) &= \emptyset \text{ FP} \\ \tau_4 (\{a\}) &= \{a\} \text{ FP}\end{aligned}$$

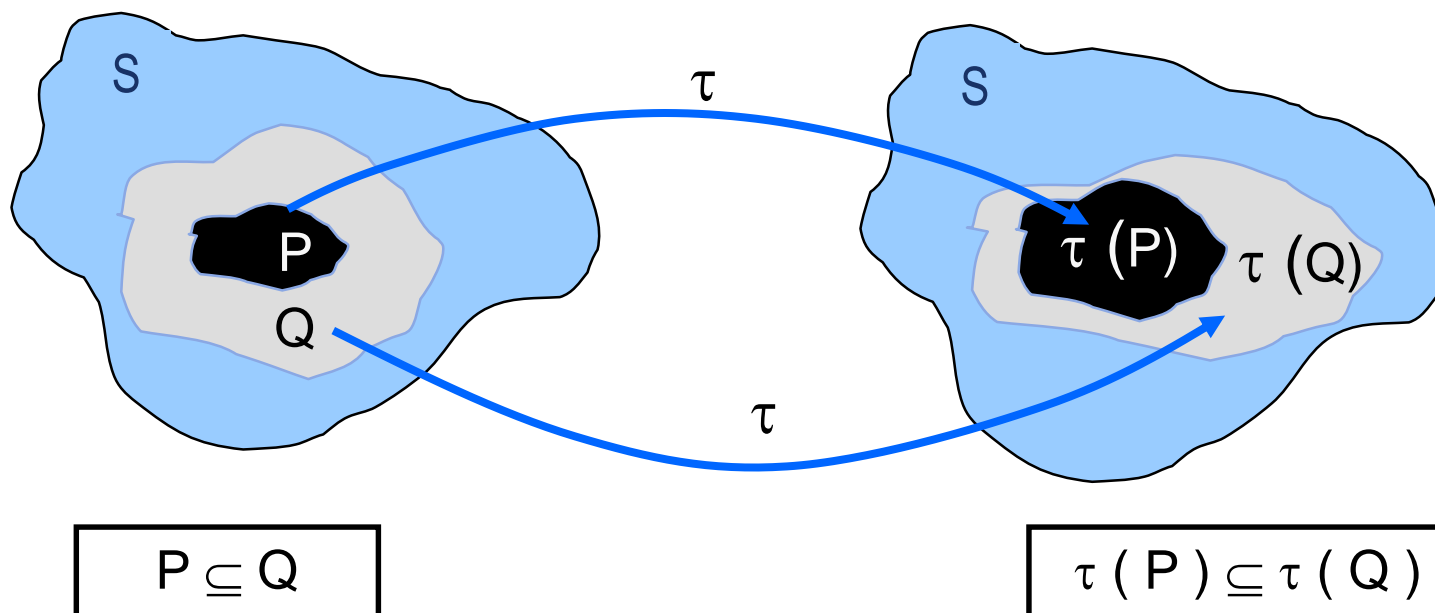
$$S = \{a, b, c, d\}$$



Как разобраться в неподвижных точках??

Монотонные операторы на конечных множествах

Оператор τ на 2^S **МОНОТОННЫЙ**, если $P \subseteq Q \Rightarrow \tau(P) \subseteq \tau(Q)$

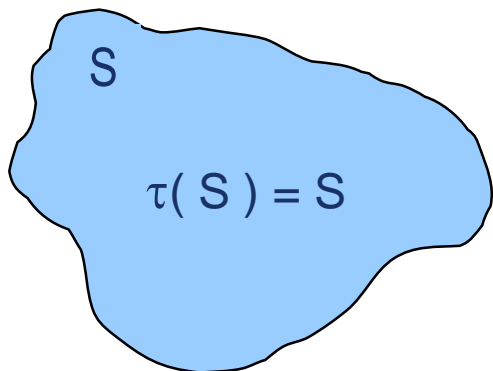


Композиция монотонных операторов -- монотонный оператор

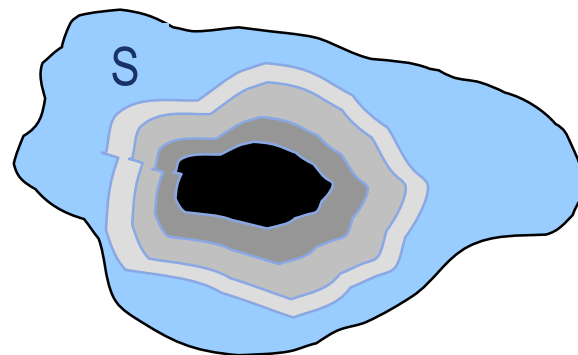
Теорема Тарского о неподвижной точке

Что, если **МОНОТОННЫЙ** оператор τ на 2^S применить к множеству S ?

Очевидно, что $\tau(S) \subseteq S$, $\tau(\tau(S)) \subseteq \tau(S)$, и т.д.: $\tau^{k+1}(S) \subseteq \tau^k(S)$



Если $\tau(S) = S$,
то S – неподвижная точка τ ,
 $\tau^k(S) = S$



Если $\tau(S) \subset S$,
то $\tau^k(S) \subset S$, $\tau(S)$ – “сжимает” S :
 $\tau^{k+1}(S) \subseteq \tau^k(S)$

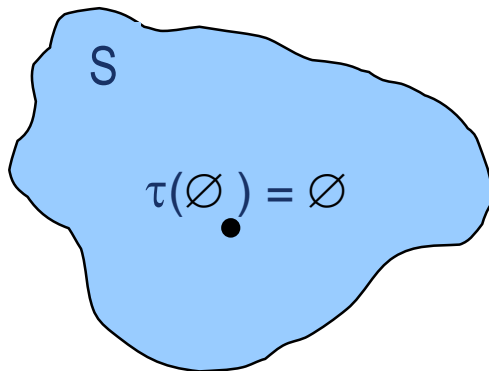
До каких пор?

Поскольку множество S конечное, то $S, \tau(S), \tau^2(S) \dots \tau^k(S)$ придет к FP

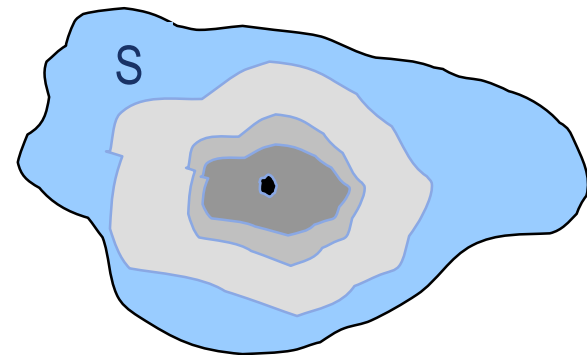
Теорема Тарского о неподвижной точке

Что, если монотонный оператор τ на 2^S применить к пустому множеству \emptyset ?

Очевидно, что $\emptyset \subseteq \tau(\emptyset)$, $\tau(\emptyset) \subseteq \tau(\tau(\emptyset))$ и т.д.: $\tau^k(\emptyset) \subseteq \tau^{k+1}(\emptyset)$



Если $\emptyset = \tau(\emptyset)$,
то \emptyset – неподвижная точка τ ,
 $\tau^k(\emptyset) = \emptyset$



Если $\emptyset \subset \tau(\emptyset)$,
то $\emptyset \subset \tau^k(\emptyset)$, $\tau(\emptyset)$ – “расширяющий”
оператор: $\tau^k(\emptyset) \subseteq \tau^{k+1}(\emptyset)$

До каких пор?

Поскольку множество S конечное, то $\emptyset, \tau(\emptyset), \tau^2(\emptyset) \dots \tau^k(\emptyset)$ придет к FP

Решетка множества всех подмножеств

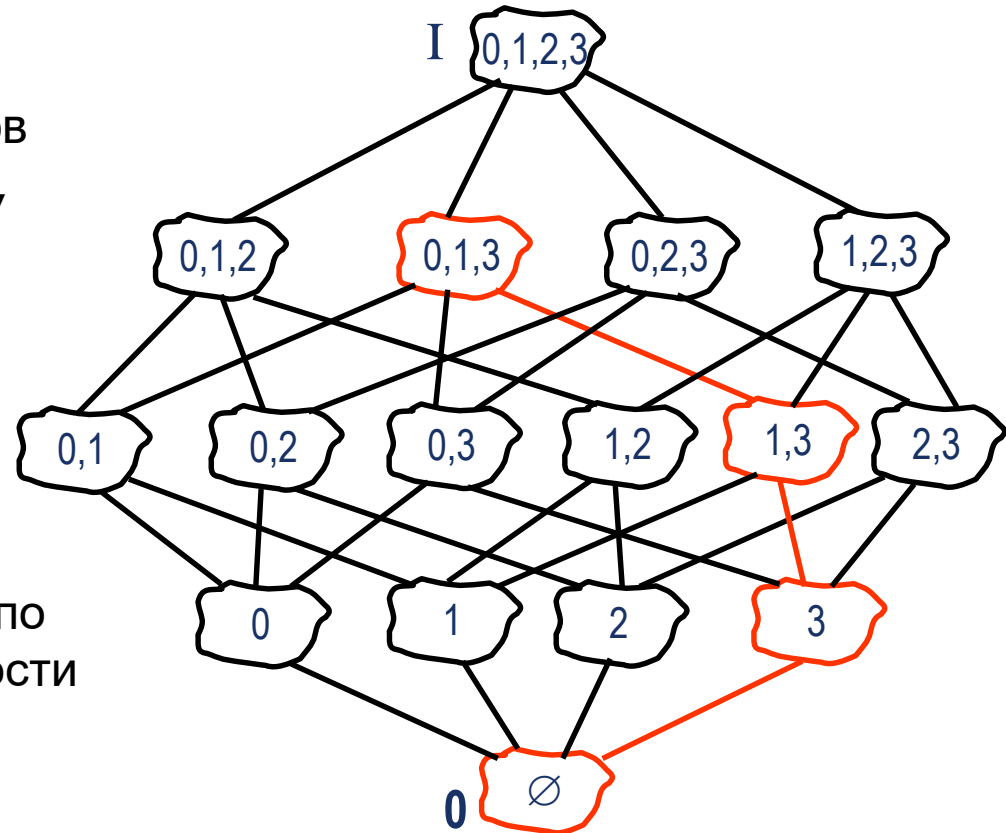
Пусть $S = \{0, 1, 2, 3\}$ $2^S = \{\emptyset, \{0\}, \{1\}, \{2\}, \{3\}, \{0, 1\}, \dots \{0, 1, 2, 3\}\}$

Решетка – частичный порядок, в который для каждой пары элементов входит и наибольшая нижняя грань, и наименьшая верхняя грань.

Решетка имеет минимальный и максимальный элементы

Все элементы 2^S образуют решетку по отношению частичной упорядоченности

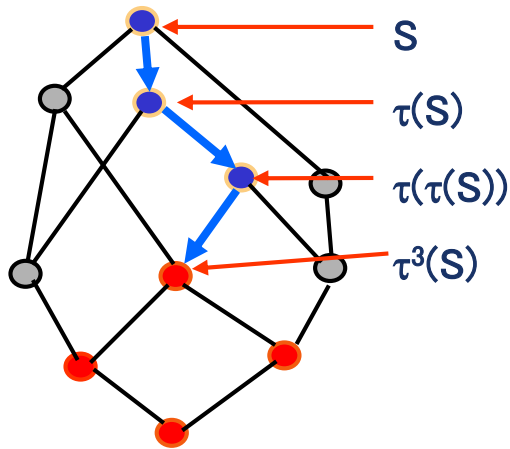
$$A \leq B \text{ iff } A \subseteq B$$



Теорема Тарского о неподвижной точке для конечных множеств

Теорема Тарского: Все неподвижные точки монотонного оператора на конечном множестве образуют решетку. Любой монотонный оператор имеет и максимальную, и минимальную FP, которые находятся алгоритмически

● неподвижная точка



$\tau(S) \subseteq S$ всегда для монотонного оператора

$$\tau(S) \subseteq S \Rightarrow \tau(\tau(S)) \subseteq \tau(S)$$

Максимальная неподвижная точка
 $\bigvee S. \tau(S) = \tau^\infty(S)$

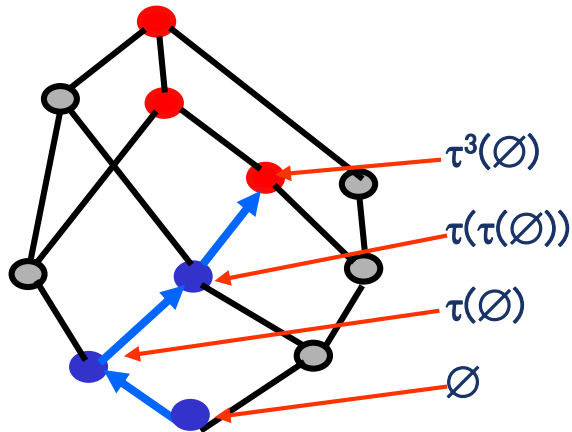
В некоторых приложениях важны **максимальные** FP

Теорема Тарского о неподвижной точке для конечных множеств

Теорема Тарского: Все неподвижные точки *монотонного* оператора на *конечном* множестве образуют решетку. Любой монотонный оператор имеет и максимальную, и минимальную FP, которые находятся алгоритмически

● неподвижная точка

$\emptyset \subseteq \tau(\emptyset)$ всегда для монотонного оператора



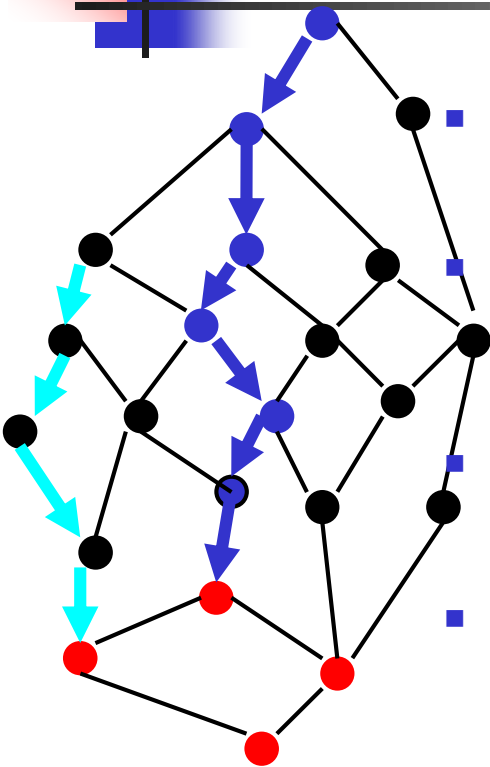
Расширяющий оператор

$\emptyset \subseteq \tau(\emptyset) \Rightarrow \tau(\emptyset) \subseteq \tau(\tau(\emptyset))$

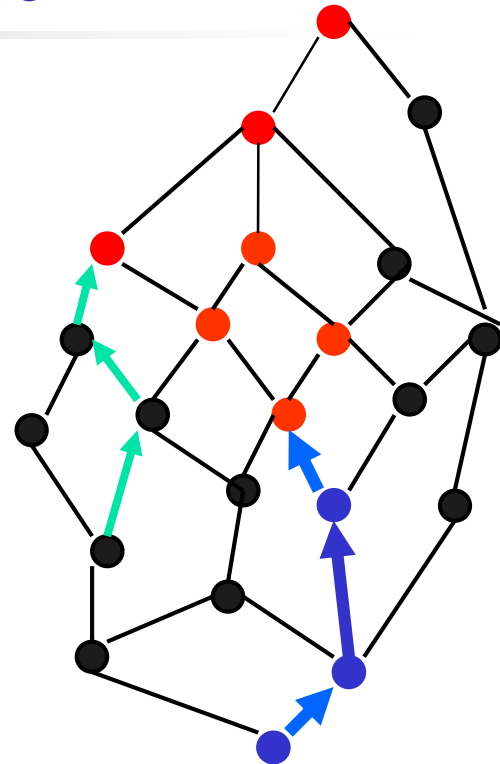
Минимальная неподвижная точка
 $\mu S. \tau(S) = \tau^\infty(\emptyset)$

В некоторых приложениях важны **минимальные** FP

Теорема Тарского о неподвижной точке на конечных множествах - доказательство



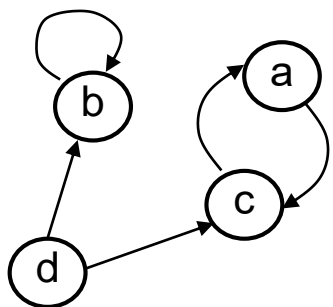
- Любой монотонный оператор на конечном множестве имеет неподвижные точки
- Все эти неподвижные точки образуют решетку, поэтому среди них есть минимальный и максимальный элемент
- Максимальная неподвижная точка монотонного оператора может быть найдена как предел $\tau^k(S)$
- Минимальная неподвижная точка монотонного оператора может быть найдена как предел $\tau^k(\emptyset)$



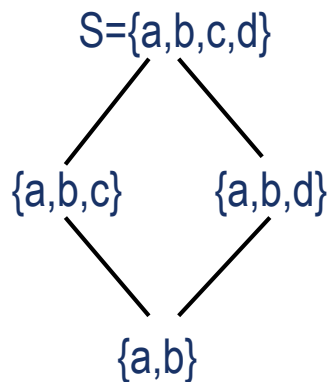
Доказательство. Для любого k , $\tau^k(\emptyset) \subseteq \tau^{k+1}(\emptyset)$ (база: $\emptyset \subseteq \tau(\emptyset)$, шаг индукции – по монотонности τ). Пусть $M = \tau^\infty(\emptyset)$. Поскольку S – конечное множество, то $\tau^k(\emptyset) = \tau^{k+1}(\emptyset)$ с какого-то k . Но тогда $\tau(M) = \tau^{k+1}(\emptyset) = M$ (след, M – FP)

Докажем, что $M \subseteq$ любой другой FP M^* . Применим k раз к обеим частям очевидного неравенства $\emptyset \subseteq M^*$ оператор τ : $\tau^k(\emptyset) = M \subseteq \tau^k(M^*) = M^*$ \square

Решетки неподвижных точек операторов



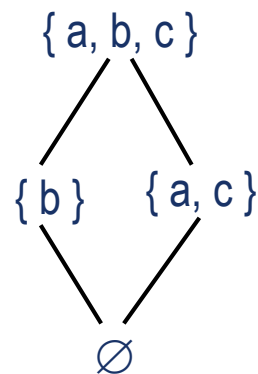
$$\tau_1 (Q) = \{a, b\} \cup Q$$



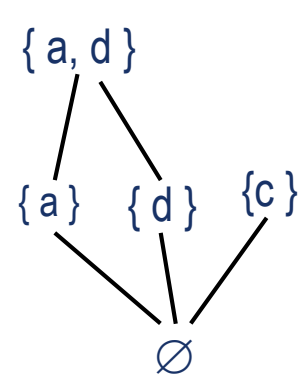
$$\tau_2 (Q) = \{c\} \cup \{a\} \cap Q$$



$$\tau_3 (Q) = Q'$$



$$\tau_4 (Q) = Q \setminus Q'$$



τ_1, τ_2, τ_3 монотонные операторы

Их неподвижные точки образуют решетку, а наибольшая и наименьшая неподвижные точки находятся простыми алгоритмами

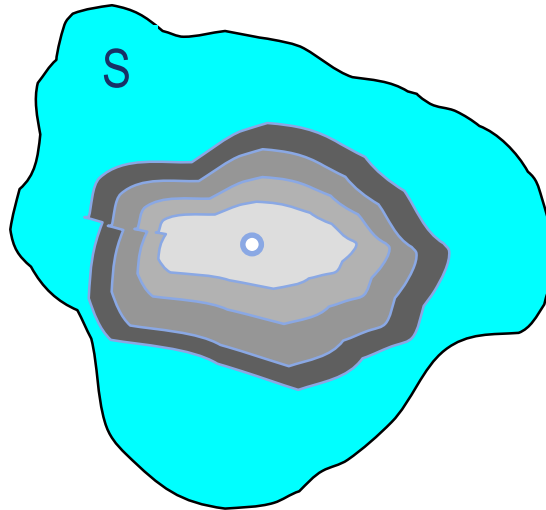
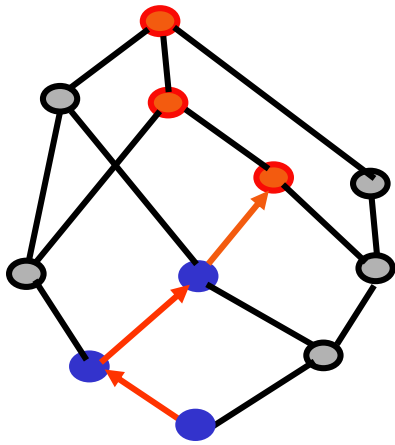
τ_4 — не монотонный оператор

У этого оператора есть неподвижные точки, но они не образуют решетку. У него нет **наибольшей** неподвижной точки.

Вычисление минимальной неподвижной точки Least Fixed Point (LFP) – расширяющий оператор

$$\mu S. \tau (S) = \bigcup_{k=0}^{\infty} \tau^k (\emptyset)$$

$$\tau^k(\emptyset) \subseteq \tau^{k+1}(\emptyset)$$



Вычисление LFP(Tau)

```
A := ∅;  
do  
  B := A;  
  A := Tau ( B );  
while ( A ≠ B );  
return ( A );
```

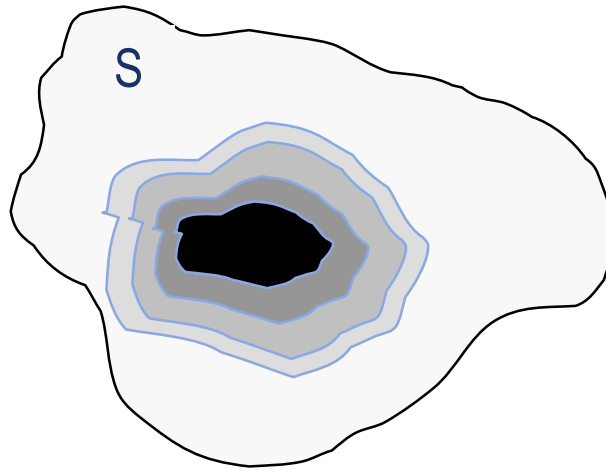
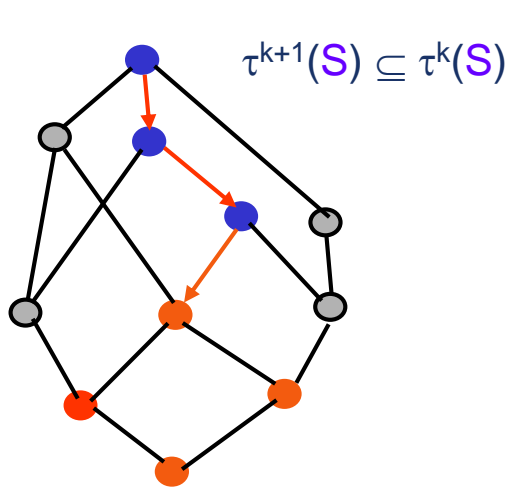
Минимальная неподвижная точка монотонного оператора τ
вычисляется как предел последовательности:

$$\emptyset \subseteq \tau (\emptyset) \subseteq \tau (\tau (\emptyset)) \dots$$

Эта последовательность всегда имеет предел – **Least Fixed Point (LFP)**

Вычисление максимальной неподвижной точки Greatest Fixed Point (GFP) – сжимающий оператор

$$\nu S. \tau(S) = \bigcap_{k=0}^{\infty} \tau^k(S)$$



Вычисление GFP (Tau)

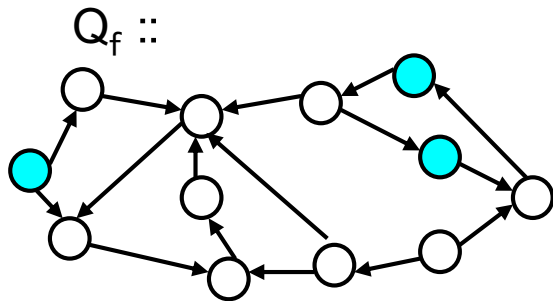
```
A := S;  
do  
  B := A;  
  A := Tau ( B );  
while ( A ≠ B );  
return ( A );
```

Максимальная неподвижная точка монотонного оператора τ
вычисляется как предел последовательности:
 $S \supseteq \tau(S) \supseteq \tau(\tau(S)) \supseteq \dots$

Эта последовательность всегда имеет предел – **Greatest Fixed Point (GFP)**

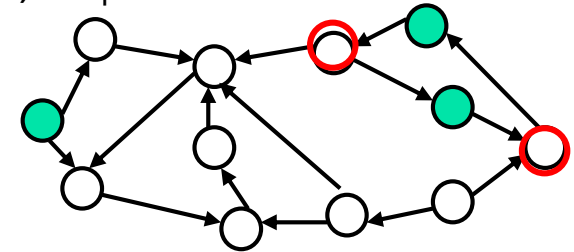
Пример: Множество Q_{Eff} как минимальная неподвижная точка оператора

Состояния, которые помечены Eff - это состояния, помеченные f , + все те, из которых за один шаг можно достичь состояния, помеченного Eff , причем нам нужно построить МИНИМАЛЬНОЕ множество таких состояний



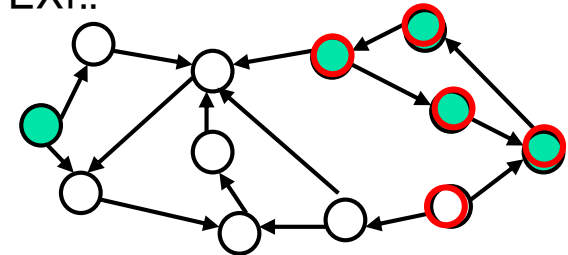
$$\tau(Q) = Q_f \cup \text{EX}(Q)$$

$$\tau^1(\emptyset) = Q_f ::$$

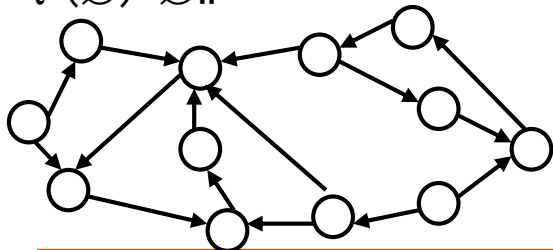


Минимальная FP определяется как предел $\emptyset \subseteq \tau(\emptyset) \subseteq \tau^2(\emptyset) \subseteq \tau^3(\emptyset) \subseteq \dots$

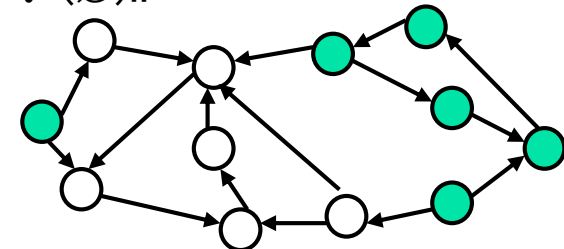
$$\tau^2(\emptyset) = Q_f \cup \text{EX}f ::$$



$$\tau^0(\emptyset) = \emptyset ::$$



$$\tau^3(\emptyset) = Q_f \cup \text{EX}\tau^2(\emptyset) = \tau^\infty(\emptyset) ::$$

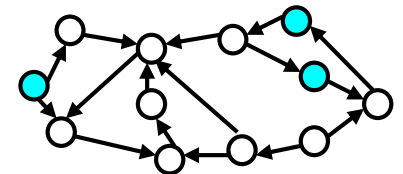
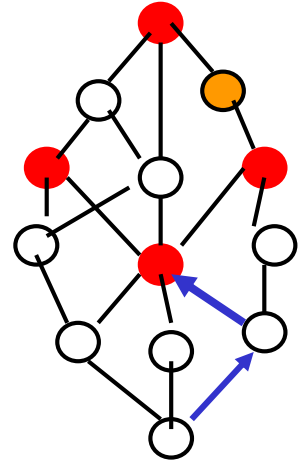
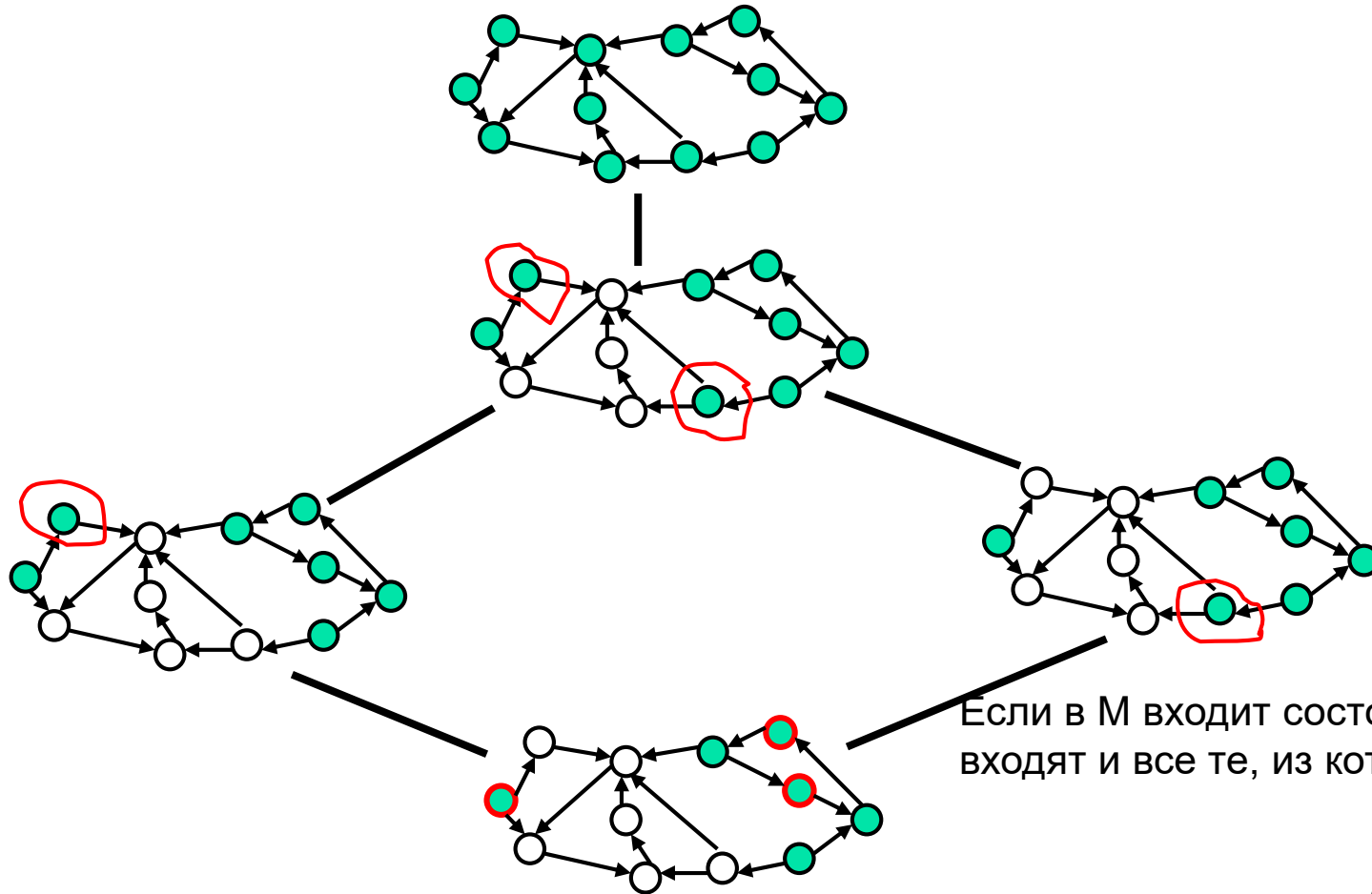


Красным обведены состояния $\text{EX}(\text{помеч})$, т.е. те, из которых можно достичь помеченные за 1 шаг

Решетка неподвижных точек оператора

$\tau(Q) = Q_f \cup EX(Q)$ – нахождение множества Q_{Eff}

Максимальная неподвижная точка – все множество S

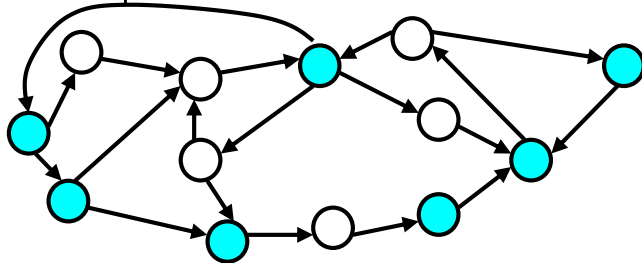


Минимальная неподвижная точка – искомое множество EFf

Пример: максимальная неподвижная точка оператора

Множество состояний, из которых существует путь, на котором Φ выполняется на каждом расстоянии четной длины от этих состояний

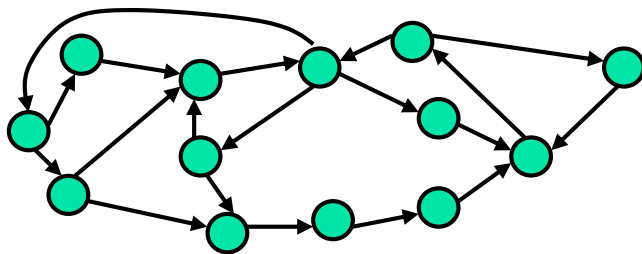
$\Phi ::$



$$\tau(Q) = \Phi \cap \text{EXEX } Q$$

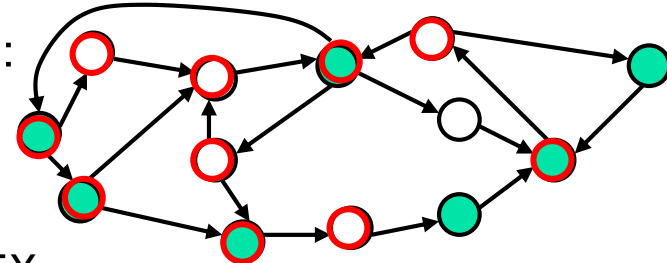
Максимальная FP определяется как предел $S \supseteq \tau(S) \supseteq \tau^2(S) \supseteq \tau^3(S) \supseteq \dots$

$$\tau^0(S) = S ::$$

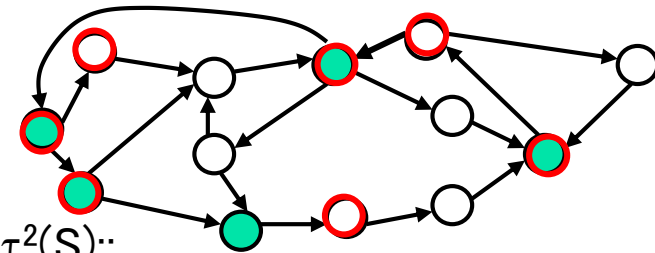


Красным обведены состояния EXEX Φ , т.е. те, из которых можно достичь помеченные Φ за 2 шага

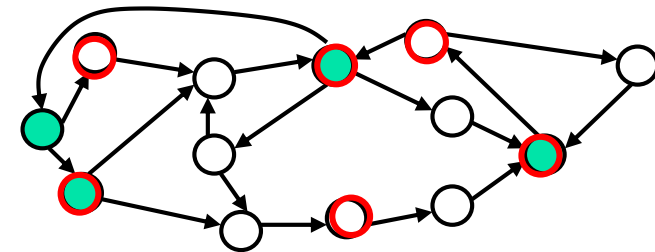
$$\tau^1(S) = \Phi ::$$



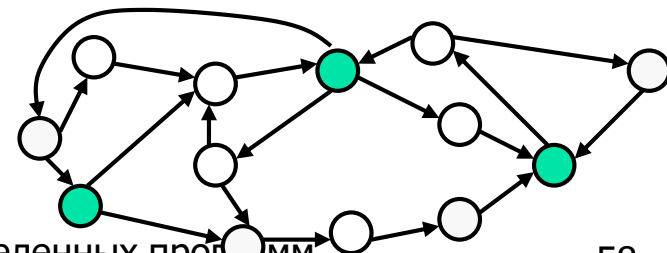
$$\tau^2(S) = \Phi \cap \text{EXEX} \Phi ::$$



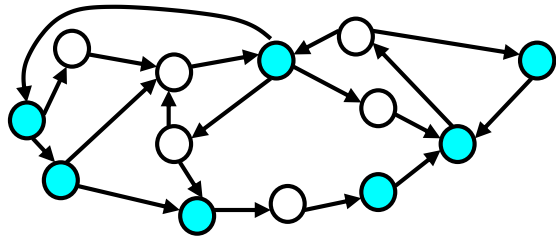
$$\tau^3(S) = \Phi \cap \text{EXEX} \tau^2(S) ::$$



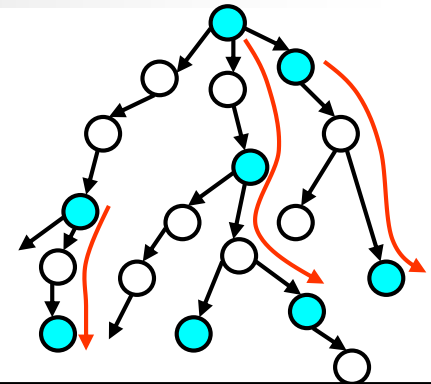
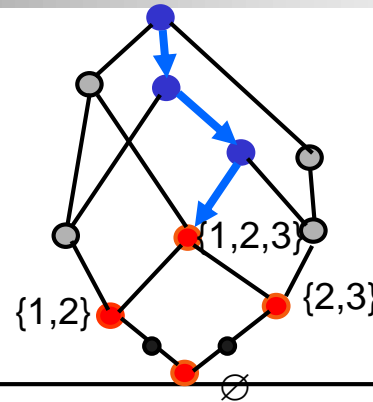
$$\tau^4(S) = \Phi \cap \text{EXEX} \tau^3(S) = \tau^\infty(S) ::$$



Решетка неподвижных точек

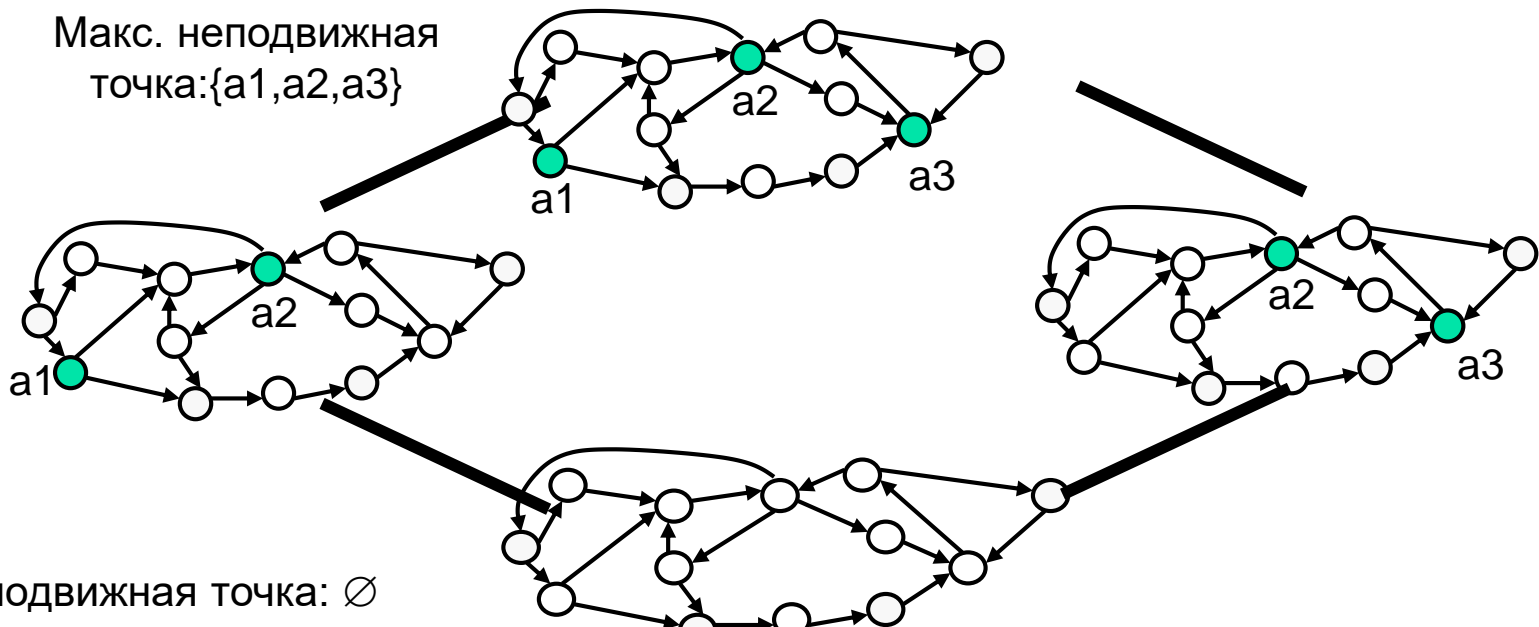


Четыре
решения –
неподвижные
точки



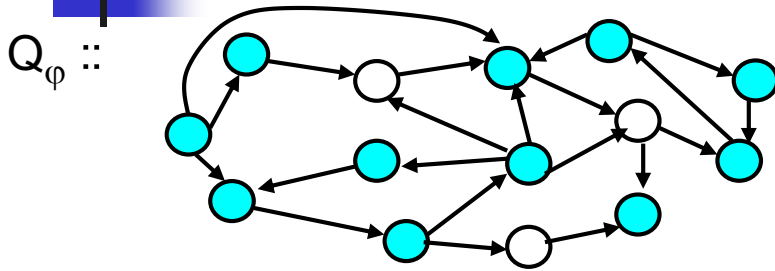
Множества состояний, из которых существует путь, на котором Φ выполняется на каждом расстоянии четной длины от этих состояний, образуют решетку

Макс. неподвижная
точка: $\{a1, a2, a3\}$

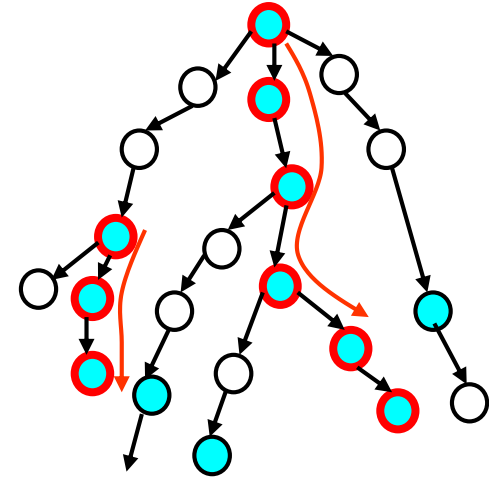


Мин. неподвижная точка: \emptyset

Пример: множество $Q_{EG\varphi}$ как максимальная неподвижная точка оператора



На структуре Крипке задано множество состояний, в которых истинна формула φ . Найти множество M таких состояний, на которых истинна формула $EG\varphi$



Множество M можно определить так: Это множество таких состояний, помеченных φ , что в них формула $EX M$ истинна (из каждого из них существует переход в состояние из M). Очевидно, $M = Q_\varphi \cap EX M$

Это значит, что M является неподвижной точкой оператора $\tau(Q) = Q_\varphi \cap EXQ$

Оператор $\tau(Q)$ - монотонный, он имеет минимальную и максимальную ФР

Определению удовлетворяет любая неподвижная точка. Например, минимальная ФР – пустое множество. Но нам, очевидно (?), нужна **МАКСИМАЛЬНАЯ** неподвижная точка оператора $\tau(Q) = Q_\varphi \cap EXQ$

Пример: максимальная неподвижная точка оператора $\tau(Q) = Q_\varphi \cap EX Q$

Q_φ

$$\tau^1(S) = Q_\varphi \cap EX S ::$$

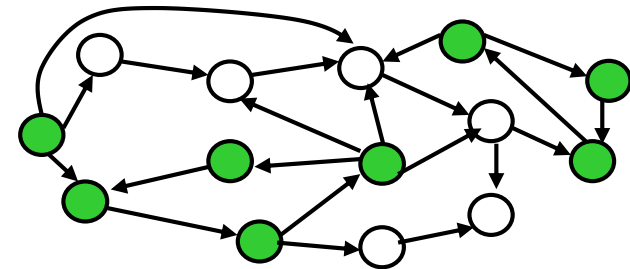
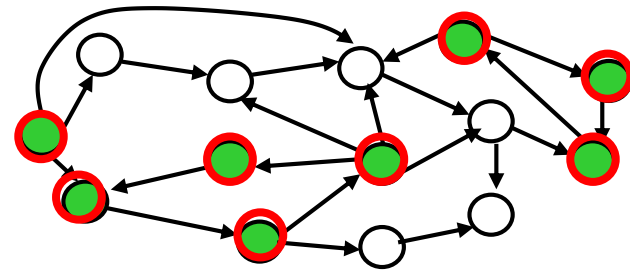
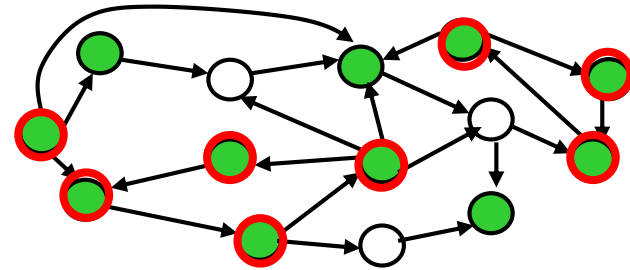
$$\tau^2(S) = Q_\varphi \cap EX Q_\varphi ::$$

$$\tau(Q) = Q_\varphi \cap EX Q$$

Максимальная FP определяется
как предел $S \supseteq \tau(S) \supseteq \tau^2(S) \supseteq \tau^3(S) \supseteq \dots$

$$\tau^0(S) = S ::$$

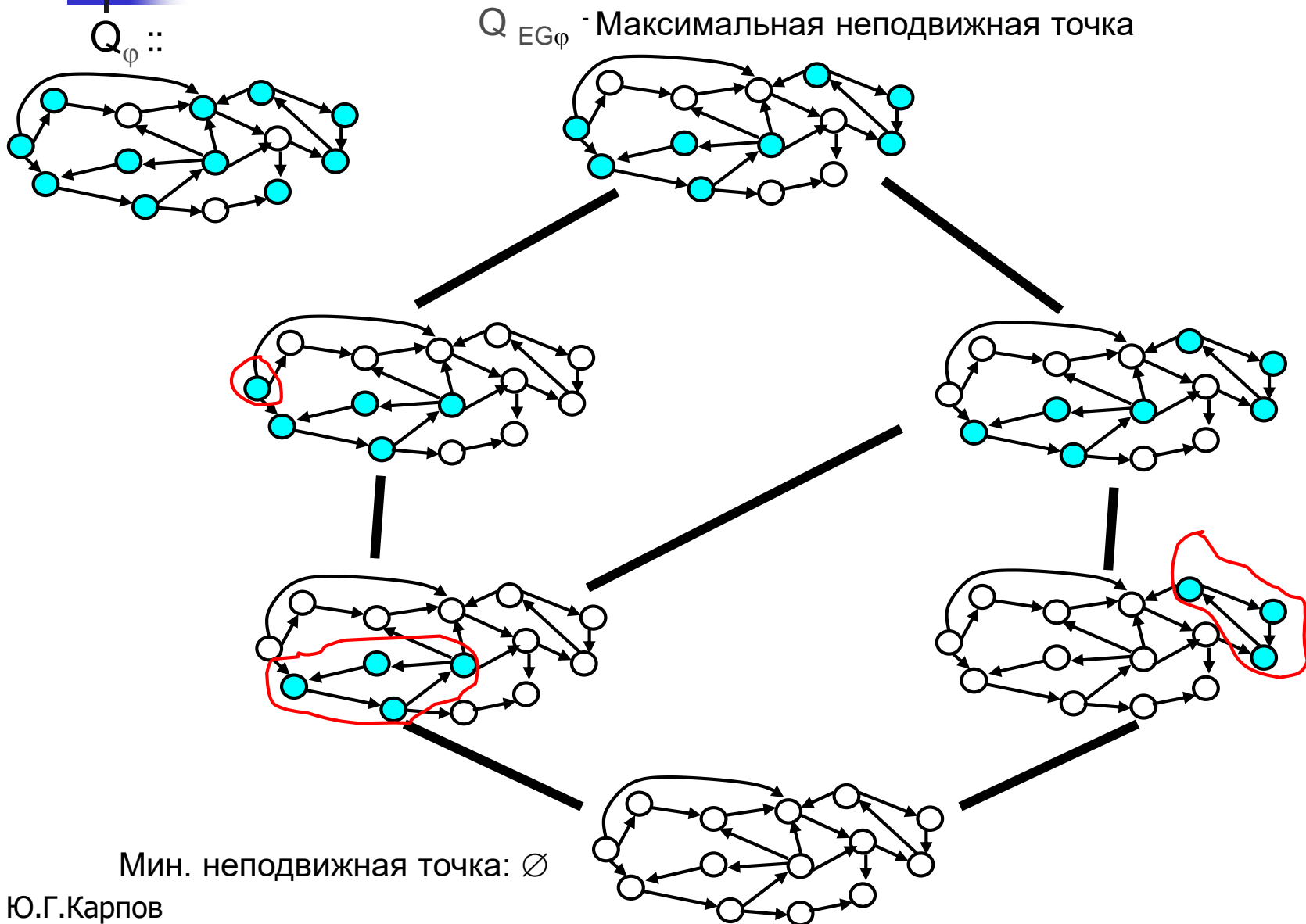
$$\tau^3(S) = \varphi \cap EX \tau^2(S) = \tau^\infty(S) ::$$



Красным обведены состояния $EX\Phi$, (из которых можно достичь помеченные Φ за 1 шаг)

Решетка неподвижных точек оператора

$$\tau(Q) = Q_{\varphi} \cap EXQ$$



Символьная верификация: все операции

$$EF f = \mu S . f \cup EX S$$

$$AF f = \mu S . f \cup AX S$$

$$EG f = \nu S . f \cap EX S$$

$$AG f = \nu S . f \cap AX S$$

$$E[f_1 \cup f_2] = \mu S . f_2 \cup (f_1 \cap EX S)$$

$$A[f_1 \cup f_2] = \mu S . f_2 \cup (f_1 \cap AX S)$$

LFP $\mu S . \tau(S)$ - свойства eventuality

GFP $\nu S . \tau(S)$ - свойства forever

Операции \cup , \cap , EX и AX

Операции выполняются над характеристическими булевыми функциями для множеств состояний

- \cup , \cap реализуются булевыми операциями \vee , \wedge над ф-циями в BDD
- EX f реализуется с помощью Backward Image над булевыми ф-циями в BDD, представляющими подмножество Q_f и отношение R структуры Крипке
- $AX f = \neg EX \neg f$



Как построить структуру Крипке в BDD

- Структуры Крипке реальных систем содержат $> 10^{100}$ состояний
- Для представления подмножеств состояний нужны БФ от ~ 300 двоичных переменных, а для представления отношений нужны БФ от ~ 600 переменных
- Но как их построить? Не перебирать же все состояния программы!

КАК представить структуру Крипке в форме BDD без предварительного явного построения состояний и переходов?

НУЖНО множество начальных состояний, переходы и функцию пометок для атомарных предикатов задавать только булевыми функциями (предикатами), которые потом представим в форме BDD!

Представление системы переходов предикатами

- Пример программы:

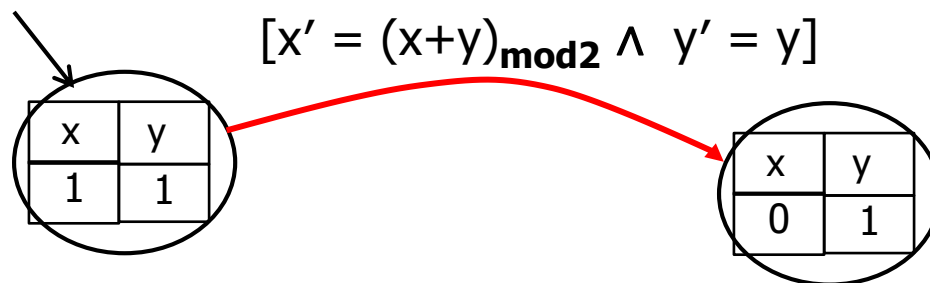
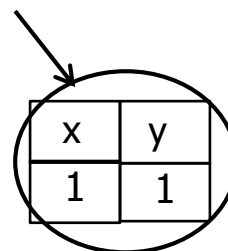
$x, y \in \{0, 1\};$
 $\text{initial_state: } x = 1, y = 1;$
 $x := (x + y) \bmod 2$

Представление множества начальных состояний:

- $S_0(x, y) \equiv (x = 1) \wedge (y = 1)$

Представление перехода:

- $R(x, y, x', y') \equiv x' = (x + y)_{\bmod 2} \wedge y' = y$



Компиляция: программа → система переходов

как предикат

Программа

```

begin
{ x, y ∈ {0, 1, 2}, x = y = 1 }
m0: x := 0;
m1: while x < 2
do
m2:  x := x + y (mod 3)
od;
m3: end
    
```

Начальный шаг: перед
каждым оператором
ставим метку

Выполнение операции – переход.
В результате компиляции
бинарное отношение R
представляется **предикатом** над
нештрихованными и
штрихованными переменными

Проверяем требование:
EF EG x > y

состояние

M	x	y
...

$s = \langle \text{Метка}, x, y \rangle$

оператор присваивания можно представить предикатом:

$R1 = C(m0; [x := 0]; m1) = \{ M = m0 \wedge M' = m1 \wedge x' = 0 \wedge y' = y \}$

начальное состояние:

M	x	y
m0	1	1

$[x := 0]$

$g \in R1$

M	x	y
m1	0	1

оператор цикла: g – переход, удовлетворяющий предикату R1

$R2 = C(m1; [\text{while } x < 2 \text{ do } m2: x := x + y (\text{mod } 3) \text{ od }]; m3) = R2.1$

$\{ (M = m1 \wedge x < 2 \wedge M' = m2 \wedge x' = x \wedge y' = y) \}$ R2.2

$\vee C(m2; [x := x + y (\text{mod } 3)]; m1)$ R2.3

$\vee (M = m1 \wedge \neg(x < 2) \wedge M' = m3 \wedge x' = x \wedge y' = y) \}$

оператор присваивания:

$C(m2; [x := x + y (\text{mod } 3)]; m1) = \{ M = m2 \wedge M' = m1 \wedge x' = x + y (\text{mod } 3) \wedge y' = y \}$

$C(M, [\text{ОП}], M')$ дает результат компиляции оператора ОП с меткой M в предикат, описывающий все возможные переходы, вызванные этим оператором. Предикат определяет все переходы, которые выражается через нештрихованные значения (первые состояния) и штрихованные значения (вторые состояния переходов).

Компиляция: программа → система переходов как предикат

Программа

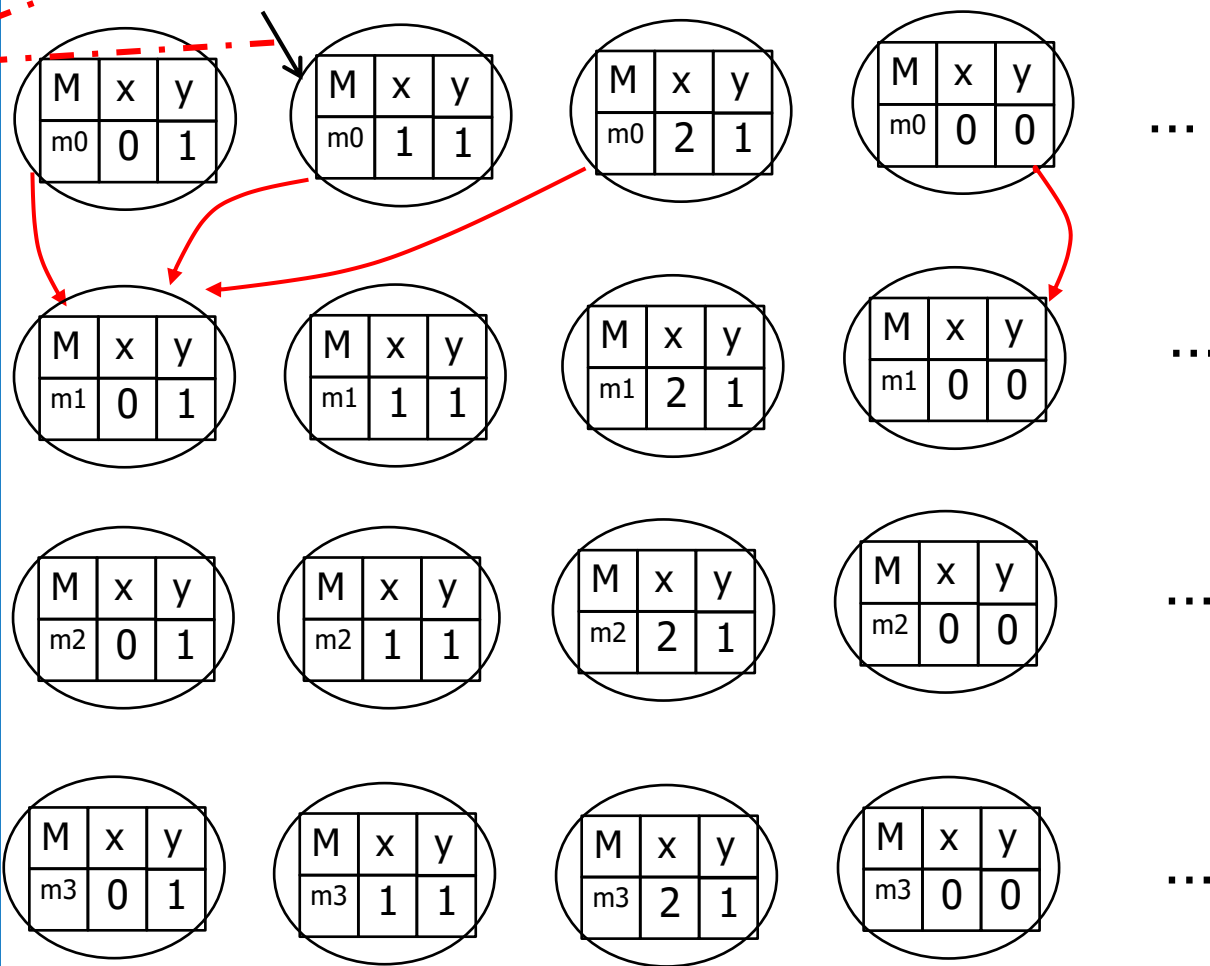
```

begin
{x,y ∈ {0,1,2}, x=y=1}
m0: x:=0;
m1: while x<2
do
m2:   x:=x+y (mod3)
od;
m3: end
    
```

В результате
компиляции
бинарное отношение R
представляется
предикатом над
нештрихованными и
штрихованными
переменными

Проверяем
требование:
EF EG $x > y$
Атом $p = x > y$

$$R1 = C(m0; [x:=0]; m1) = \{ M=m0 \wedge M'=m1 \wedge x'=0 \wedge y'=y \}$$



Всего $4 \times 3 \times 3 = 36$ состояний

Компиляция: программа → система переходов как предикат

Программа

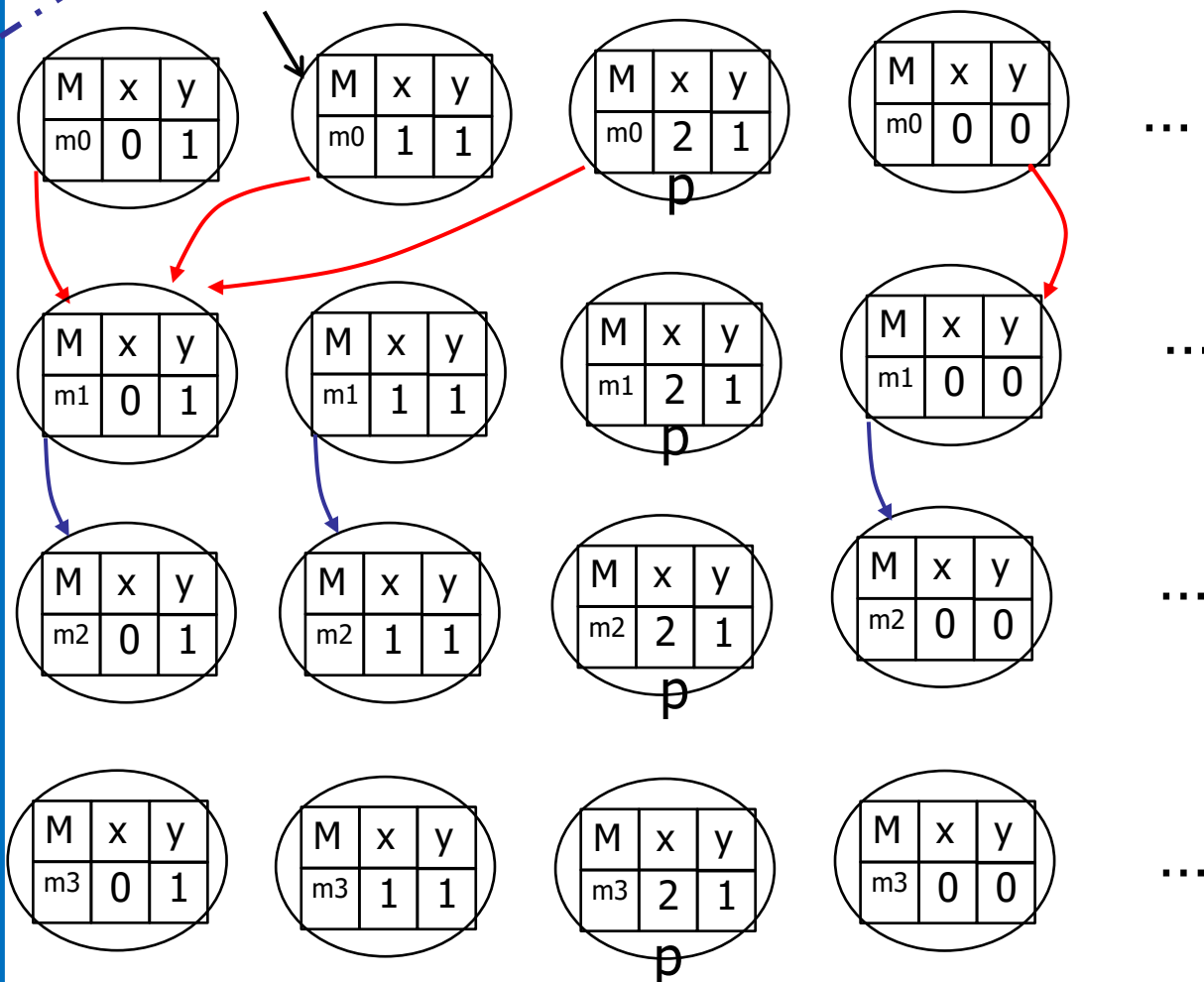
```

begin
{x,y ∈ {0,1,2}, x=y=1}
m0: x:=0;
m1: while x<2
do
m2:   x:=x+y (mod3)
od;
m3: end
    
```

В результате
компиляции
бинарное отношение R
представляется
предикатом над
нестрихованными и
стрихованными
переменными

Проверяем
требование:
EF EG $x > y$
Атом $p = x > y$

$$R2.1 = (M = m1 \wedge x < 2 \wedge M' = m2 \wedge x' = x \wedge y' = y)$$



Компиляция: программа \rightarrow система переходов как предикат

Программа

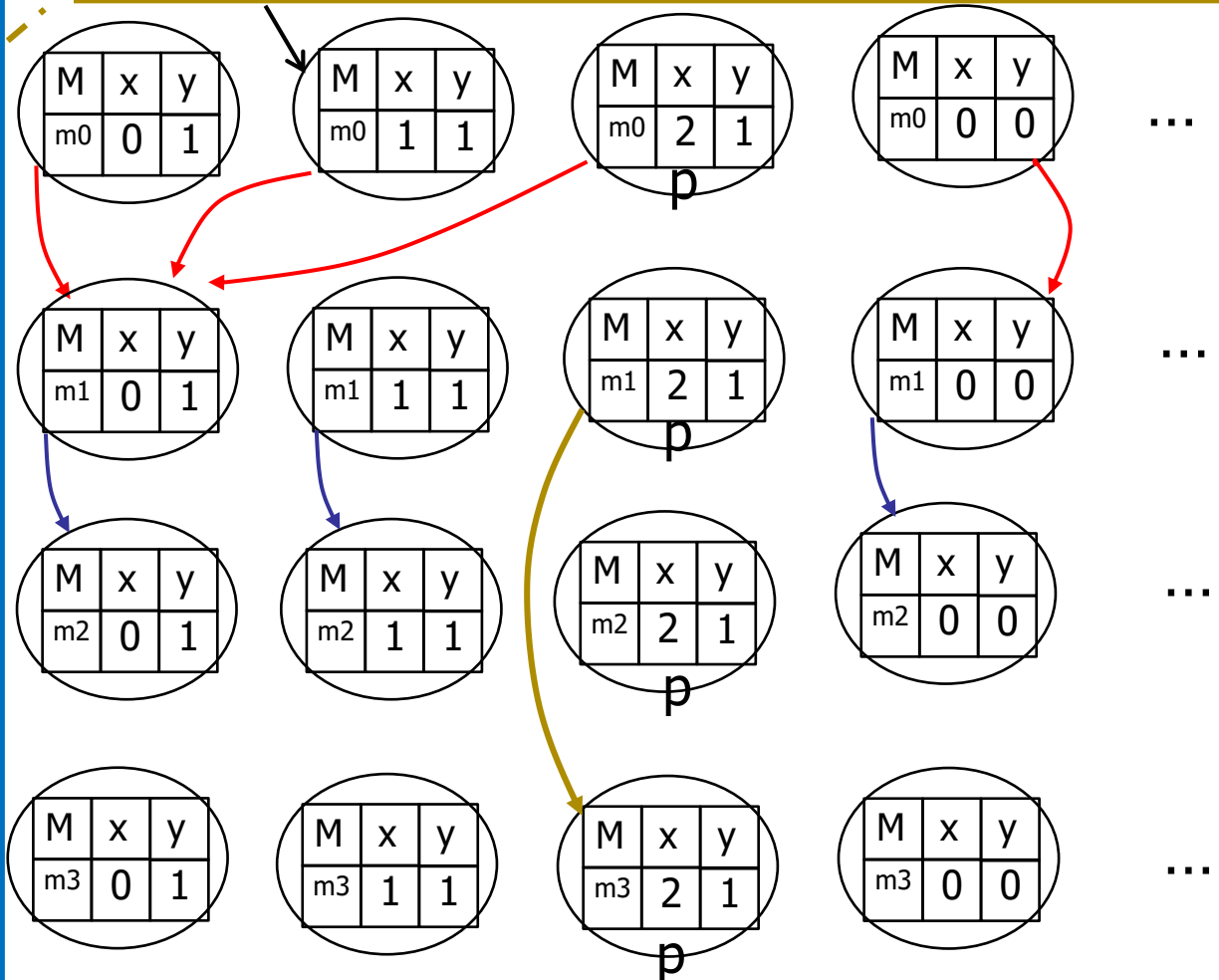
```

begin
{x,y ∈ {0,1,2}, x=y=1}
m0: x:=0;
m1: while x<2
do
m2:   x:=x+y (mod3)
od;
m3: end
    
```

В результате
компиляции
бинарное отношение R
представляется
предикатом над
нештрихованными и
штрихованными
переменными

Проверяем
требование:
EF EG $x > y$
Атом $p = x > y$

$$R2.3 = (M = m1 \wedge M' = m3 \wedge \neg(x < 2) \wedge x' = x \wedge y' = y)$$



Компиляция: программа → система переходов как предикат

Программа

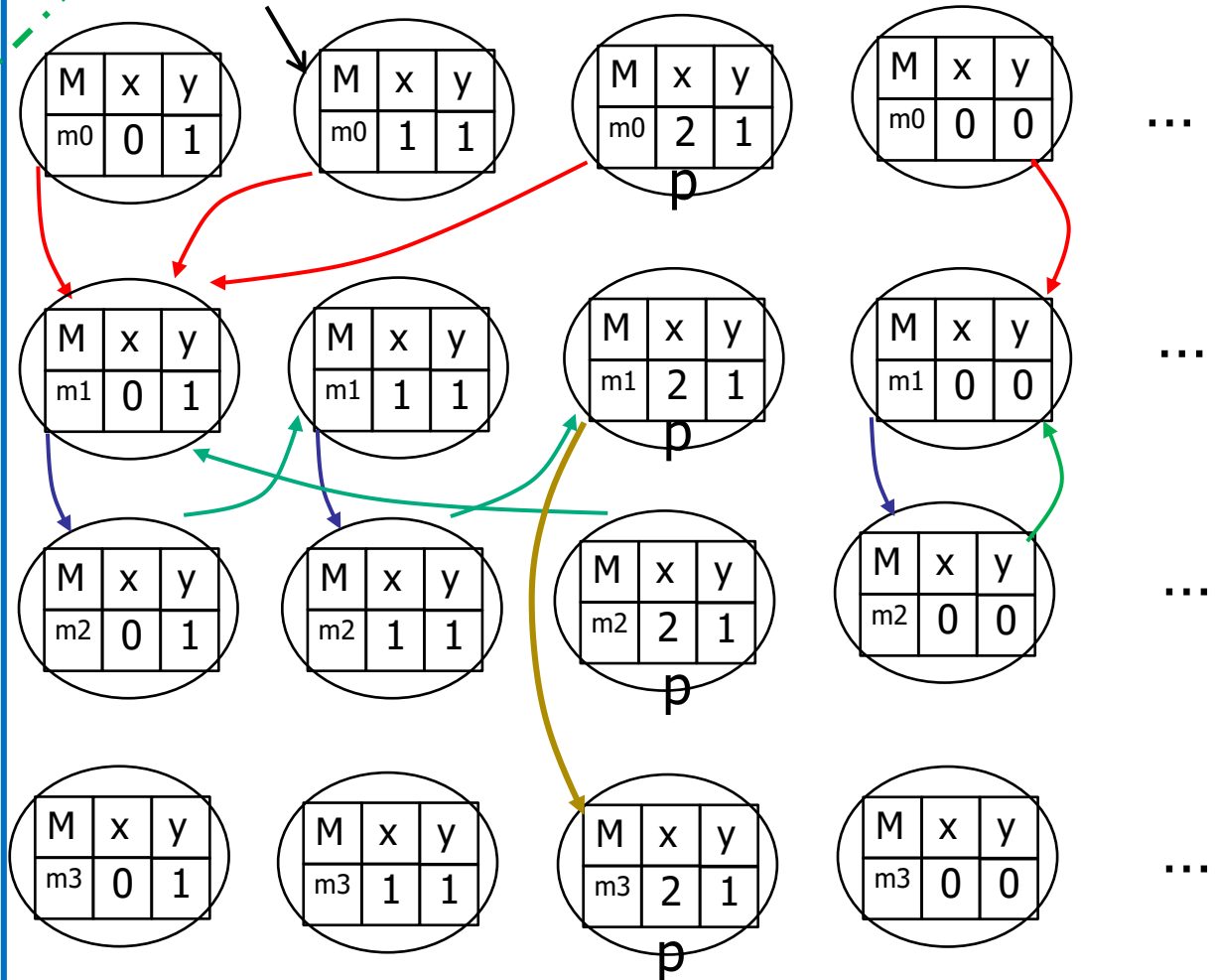
```

begin
{x,y ∈ {0,1,2}, x=y=1}
m0: x:=0;
m1: while x<2
do
m2:   x:=x+y (mod3)
od;
m3: end
    
```

В результате
компиляции
бинарное отношение R
представляется
предикатом над
нестрихованными и
штрихованными
переменными

Проверяем
требование:
EF EG $x > y$
Атом $p = x > y$

$R2.2 = (M = m2 \wedge M' = m1 \wedge x' = x + y \pmod{3} \wedge y' = y)$



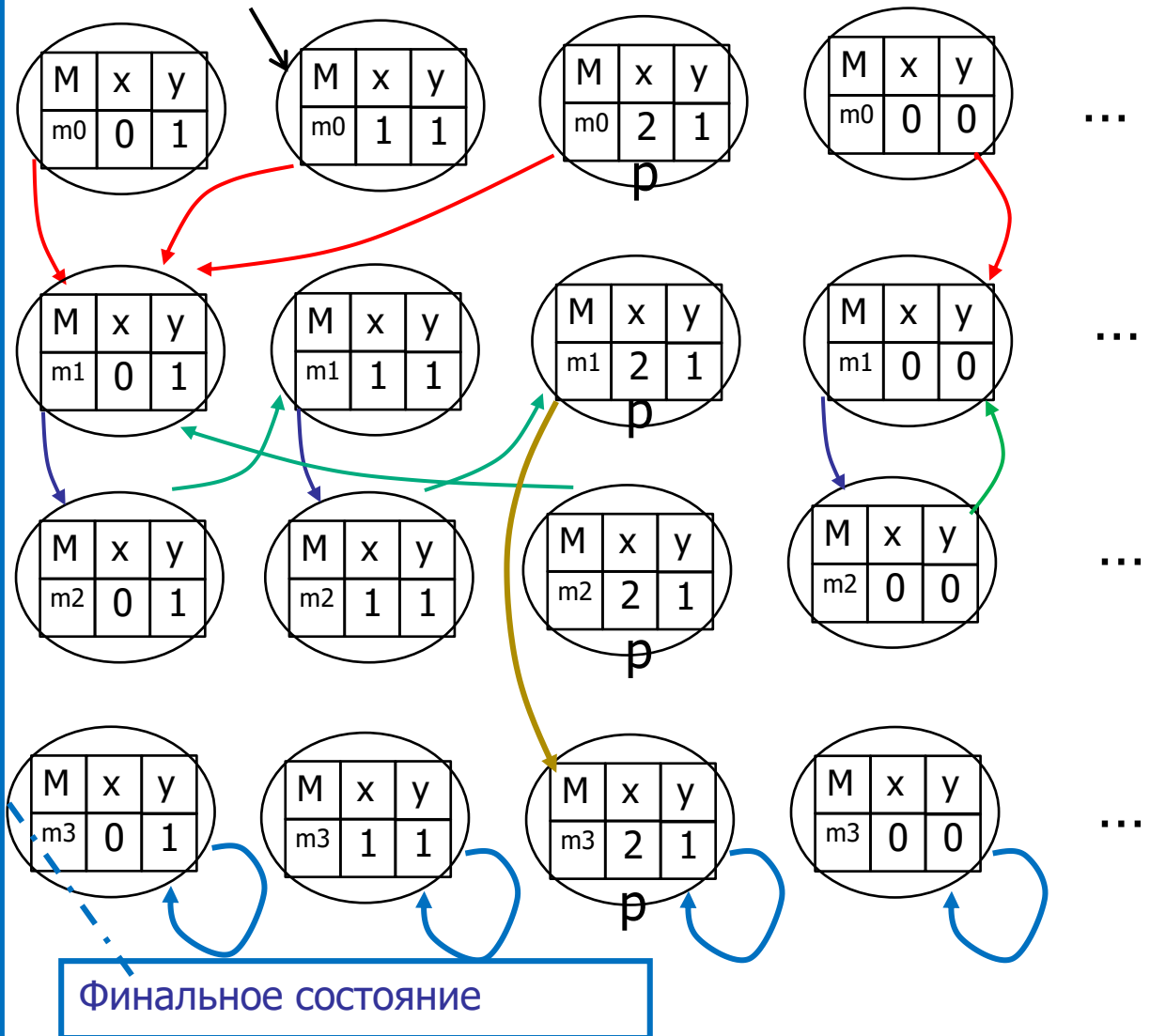
Компиляция: программа \rightarrow система переходов как предикат

Программа

```
begin
{x,y $\in$ {0,1,2}, x=y=1}
m0: x:=0;
m1: while x<2
do
m2:   x:=x+y (mod3)
od;
m3: end
```

В результате
компиляции
бинарное отношение R
представляется
предикатом над
нестрихованными и
стрихованными
переменными

Проверяем
требование:
EF EG x>y
Атом p = x>y



Компиляция: программа \rightarrow система переходов как предикат

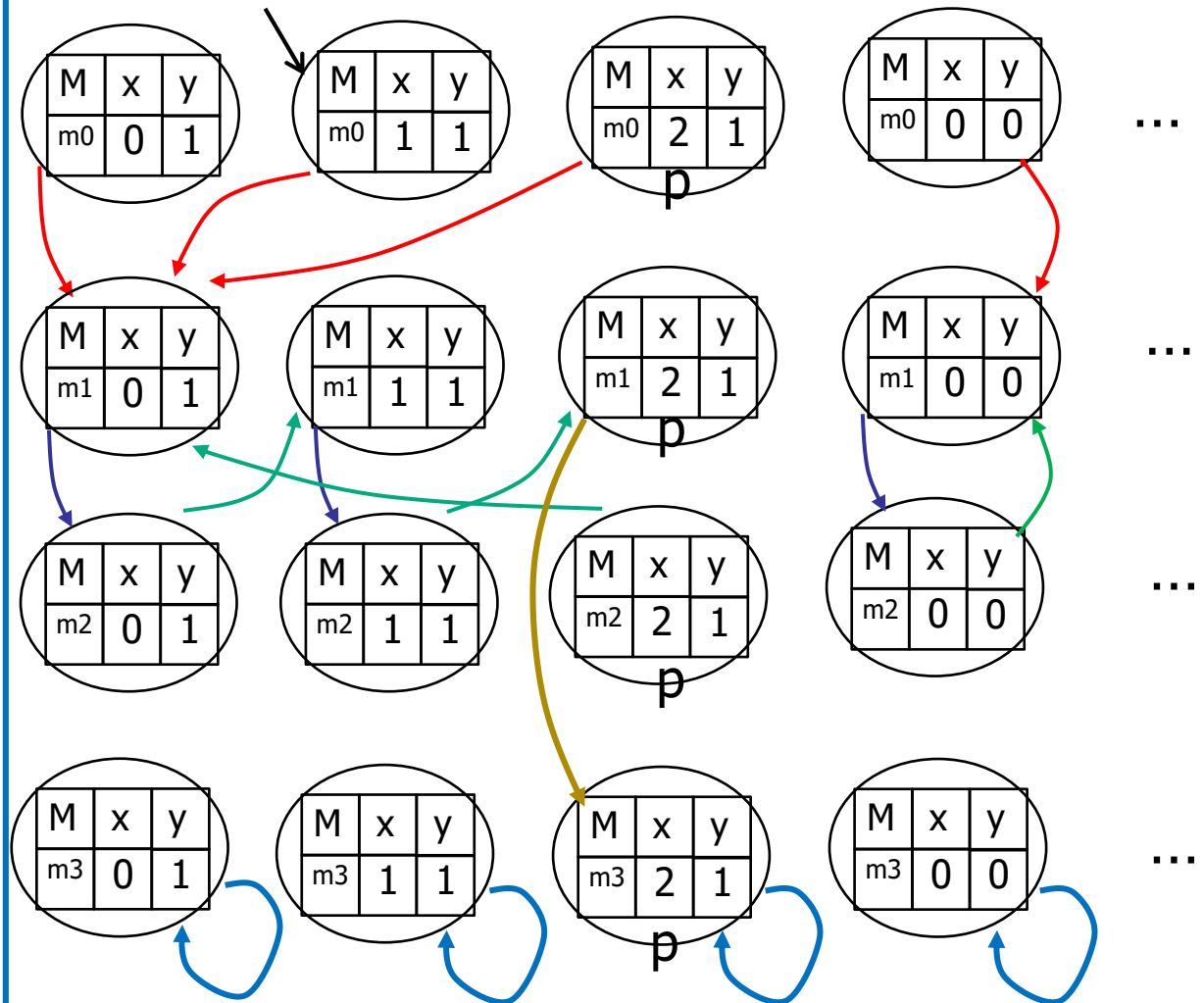
Программа

```
begin
{x,y ∈ {0,1,2}, x=y=1}
m0: x:=0;
m1: while x<2
do
m2:   x:=x+y (mod3)
od;
m3: end
```

В результате
компиляции
бинарное отношение R
представляется
предикатом над
нестрихованными и
штрихованными
переменными

Проверяем
требование:
EF EG $x > y$
Атом $p = x > y$

Достижимые состояния



Компиляция: программа \rightarrow система переходов как предикат

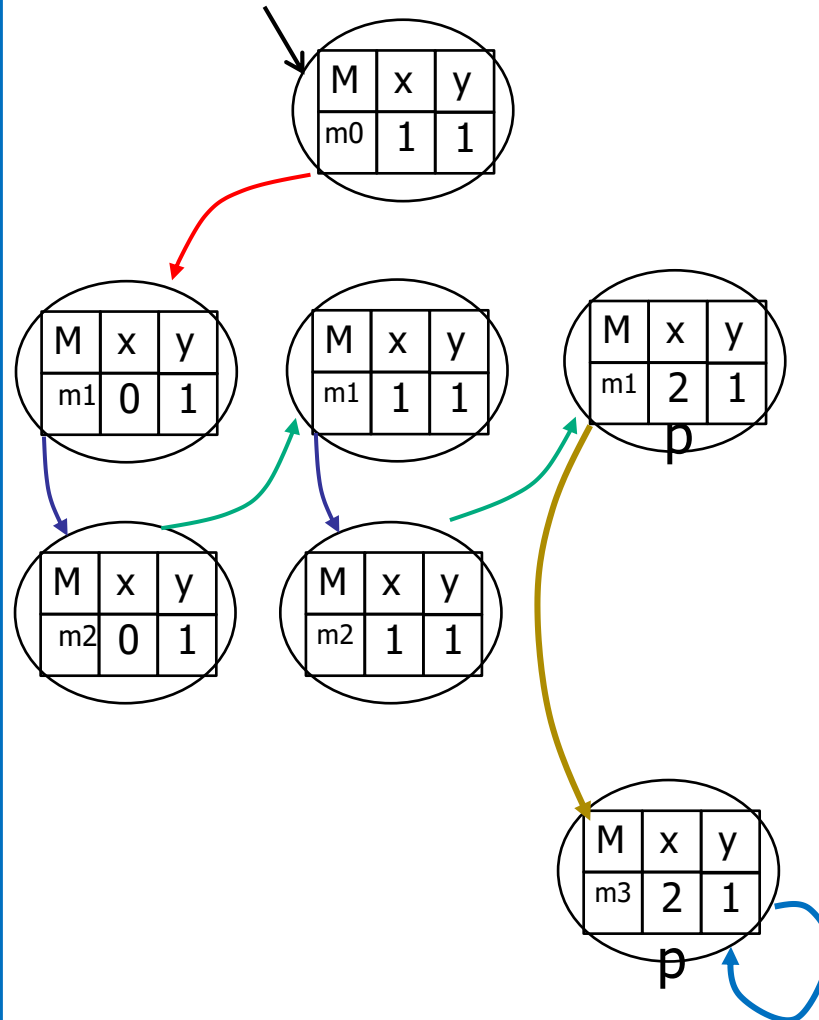
Программа

```
begin
{x,y $\in$ {0,1,2}, x=y=1}
m0: x:=0;
m1: while x<2
do
m2:   x:=x+y (mod3)
od;
m3: end
```

В результате
компиляции
бинарное отношение R
представляется
предикатом над
нестрихованными и
штрихованными
переменными

Проверяем
требование:
EF EG $x > y$
Атом $p = x > y$

Можно сразу строить только достижимые состояния



Только достижимые состояния

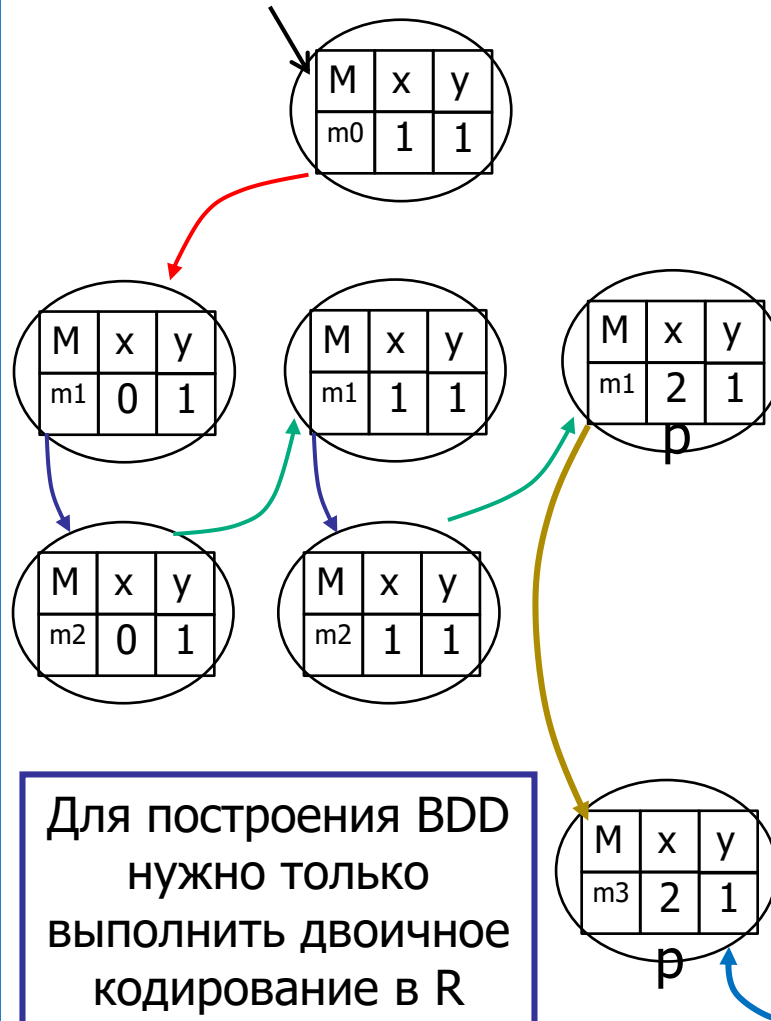
Программа

```

begin
{x,y ∈ {0,1,2}, x=y=1}
m0: x:=0;
m1: while x<2
do
m2:   x:=x+y (mod3)
od;
m3: end
    
```

В результате компиляции бинарное отношение R представляется **предикатом** над нештрихованными и штрихованными переменными

Проверяем требование:
EF EG $x > y$
Атом $p = x > y$



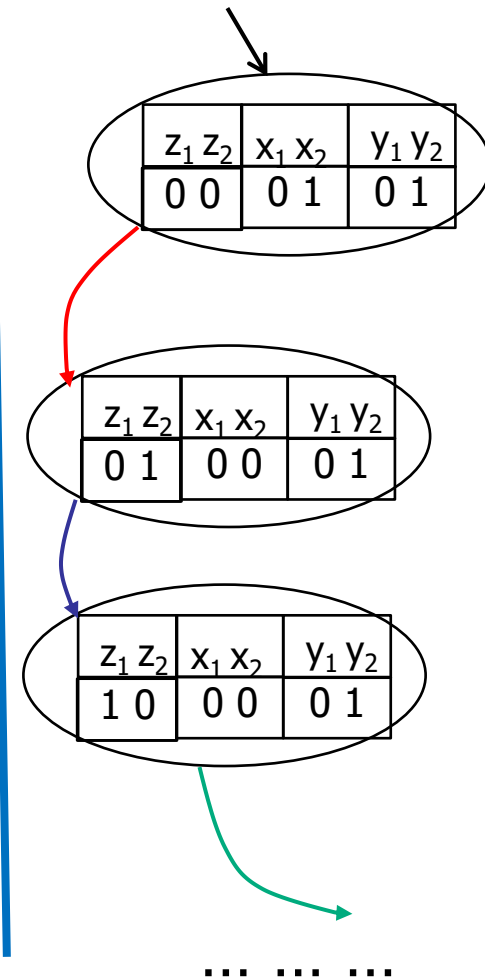
Для построения BDD нужно только выполнить двоичное кодирование в R

Кодируем:

$M: z_1, z_2;$

$x: x_1, x_2,$

$y: y_1, y_2$

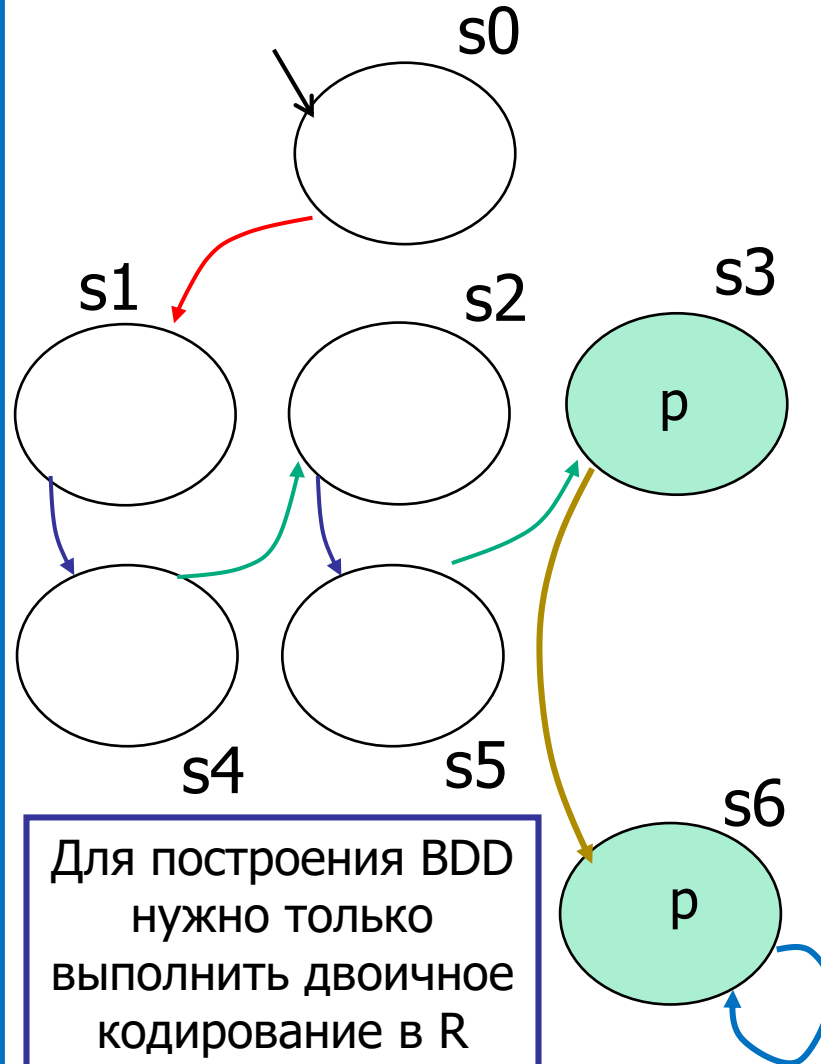


Метод проверки модели

Проверяем
требование:
 $EF\ EG\ x > y$
Атом $p = x > y$

1. $EG\ p$
 $\tau(Q) = Qp \cap EX\ Q$

- строим
характеристическую
функцию состояний,
помеченных p ,
- характеристическую
функцию R



- начинаем с $Q=S$
- строим прямой
образ Q (по R)
- находим
пересечение
- получаем новое Q
- повторяем пока Q
не перестанет
изменяться

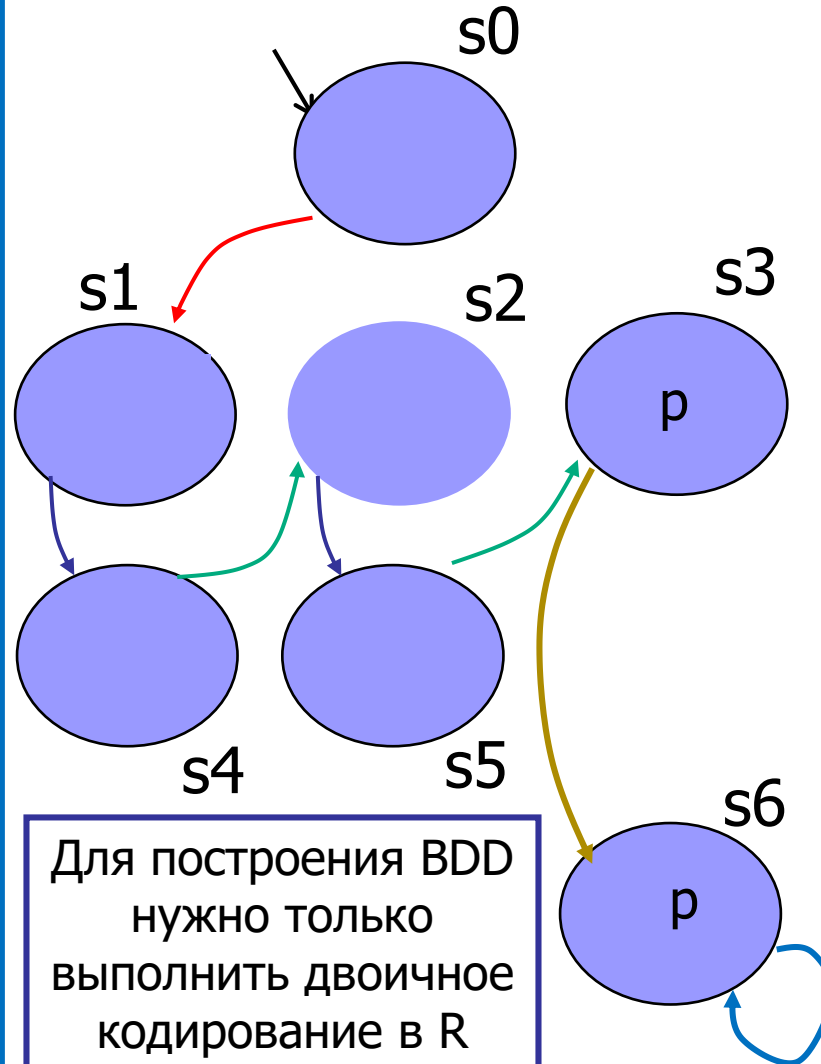
Метод проверки модели

Проверяем
требование:
 $EF EG x > y$
Атом $p = x > y$

2. $EF \psi$

$$\tau(Q) = Q \cup EX(Q)$$

- начинаем с $Q = \emptyset$
- строим прямой образ Q (по R)
- находим объединение
- получаем новое Q
- Повторяем пока Q не перестанет изменяться



Начальное состояние
помечено формулой,
значит, свойство
выполняется на
структуре Крипке

Заключение (1)





Премия Тьюринга ACM

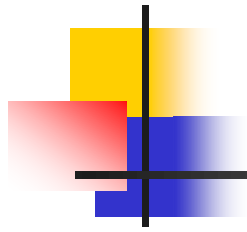
21 июня 2008 г премия Тьюринга была вручена трем создателям техники MODEL CHECKING, внесшим в нее наиболее существенный вклад:

Edmund M. Clarke (CMU),
E. Allen Emerson (U Texas, Austin),
Joseph Sifakis, (Verimag, France)

"за их роль в превращении метода Model checking в высокоэффективную технологию верификации, широко используемую в индустрии разработки программного обеспечения и аппаратных средств" .

ACM President Stuart Feldman :

"Это великий пример того, как технология, изменившая промышленность, родилась из чисто теоретических исследований".



Спасибо за внимание