Формальные методы в программной инженерии

Проверка моделей

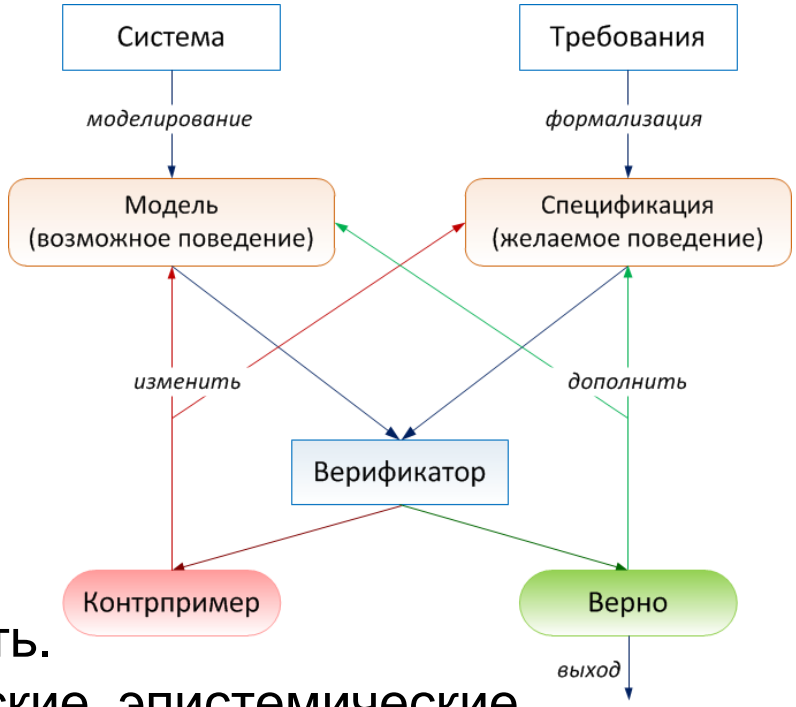
Проверка моделей

Лекция 2

Введение

- Проверка моделей — автоматическая техника верификации различных систем с (бес-)конечным числом состояний.
- Имеет ряд преимуществ относительно других подходов к верификации, основанных на тестировании, симуляции, дедукции и т.п.
- Этот метод успешно использовался в верификации очень сложных электронных схем, коммуникационных протоколах и современных полупроводниковых устройств и процессоров.
- Основная проблема проверки моделей — это «взрыв пространства состояний».
  - Он возникает в случае, когда система имеет много параллельно взаимодействующих компонент и/или содержит структуры данных, которые принимают много различных значений.

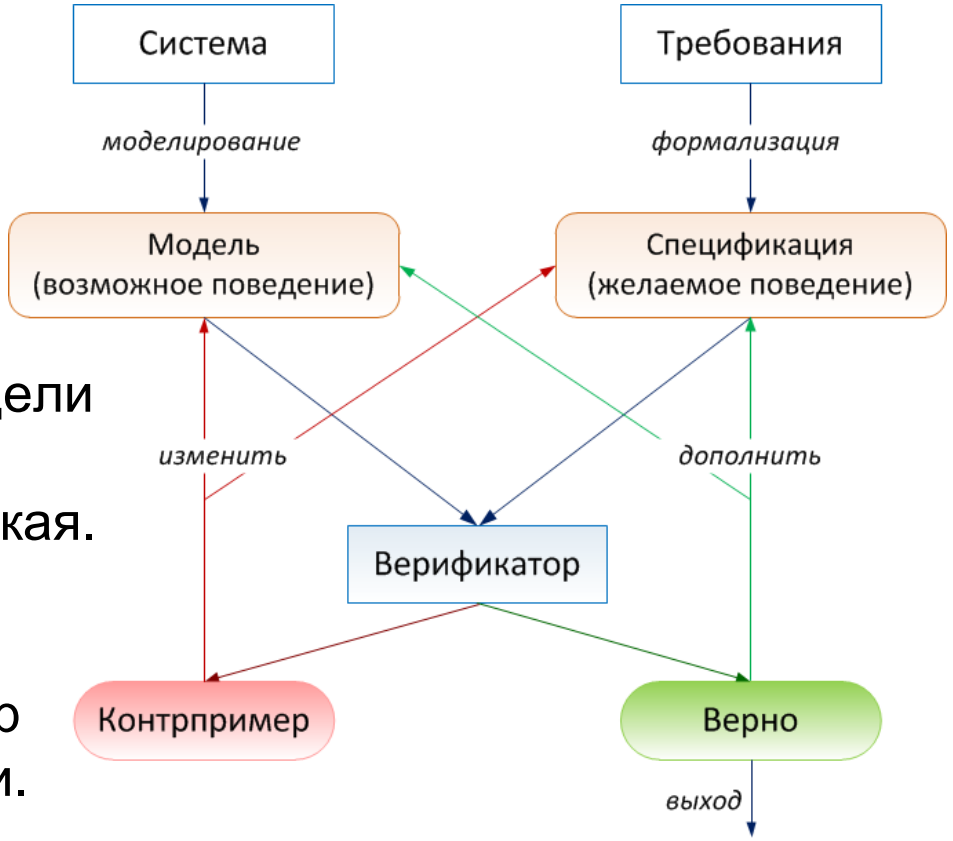
Процесс проверки моделей



- Моделирование
  - Приведение программной системы к формальному виду, подходящему для автоматической проверки.
    - Компиляция, трансляция.
    - Абстрагирование.
- Спецификация
  - Задание свойств программной системы, которые необходимо проверить.
    - Логики: темпоральные, динамические, эпистемические, деонтические и многие другие.
    - Полнота спецификации.
      - Безопасность: ничего плохого никогда не случится.
      - Живость: что-нибудь хорошее обязательно получится.
- Верификация

Процесс проверки моделей

- Моделирование
- Спецификация
- Верификация
  - Проверка соответствия модели системы её спецификации.
    - Полностью автоматическая.
    - Анализ результатов
      - Контрпример
      - Ложный контрпример
    - Большой размер модели.
      - Абстрагирование
    - Уточнение модели



Логики спецификаций и проверка модели

Модальные логики

- Темпоральные логики 🕓
  - LTL, CTL, CTL*
  - Утверждения о развитии событий системы во времени
    - Что-то когда-то обязательно случится.
      - **F** restart
    - Что-то, возможно, будет всегда.
      - **EG** blocked
    - Давным-давно наверняка было кое-что, пока не случилось нечто.
      - **A** move **U** stop
- Динамические логики

Логики спецификаций и проверка модели

Модальные логики

- Темпоральные логики 🕐
- Динамические логики 💣
  - PDL, PDL*, μ-исчисление
  - Утверждения о развитии событий системы в результате некоторых действий.
    - Если сделать кое-что, то потом обязательно получится что-то.
      - [go_down]*at_bottom*
    - Если многократно что-то делать, то, возможно, кое-что и выйдет.
      - <go_up*>*at_top*

Логики спецификаций и проверка модели

Модальные логики

- Темпоральные логики 🕐
- Динамические логики 💣
- Другие логики
  - Эпистемические 👓
    - PLK, PLC
    - Утверждения о знаниях.
      - Я знаю, что он знает, что они в курсе, что я знаю секрет.
  - Деонтические ☞
    - SDL, DDL
    - Утверждения об обязанностях
      - Нечто должно иметь место.
  - Веры
  - Комбинации логик

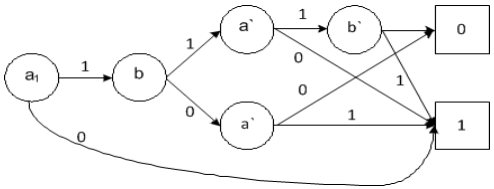
Логики спецификаций и проверка модели

Модальные логики

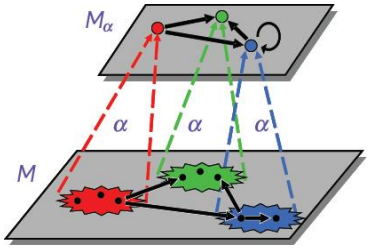
- Темпоральные логики 🕐
- Динамические логики 💣
- Другие логики
  - Эпистемические 👓
  - Деонтические ☞
  - Веры ☝
    - PLB, PPLB
    - Утверждения об убеждениях
      - Я верю, что что-нибудь истинно.
  - Комбинации логик 🕐 💣 👓 ☞ ☝
    - Всевозможные утверждения
      - Когда-нибудь после некоторого действия я узнаю, что раньше всегда должен был поступать не так, а иначе, чтобы поверить, что ничего не изменилось бы оттого, что я мог знать что-то очевидное.

Подходы к проблеме взрыва состояний

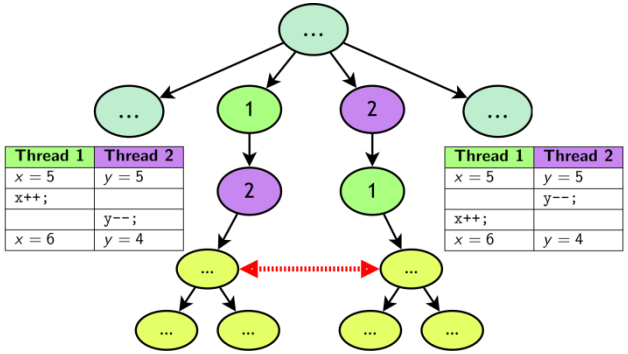
Символьные представления данных



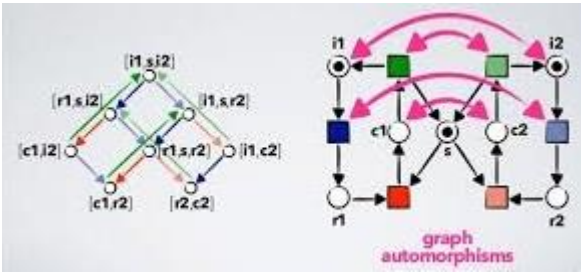
Абстракция



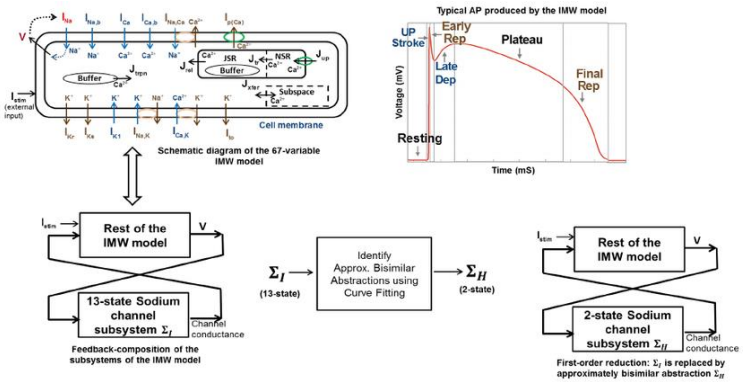
Редукция частичных порядков



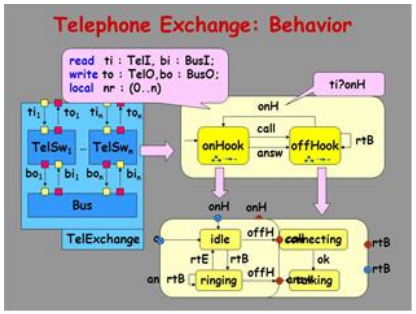
Симметрия



Композиция



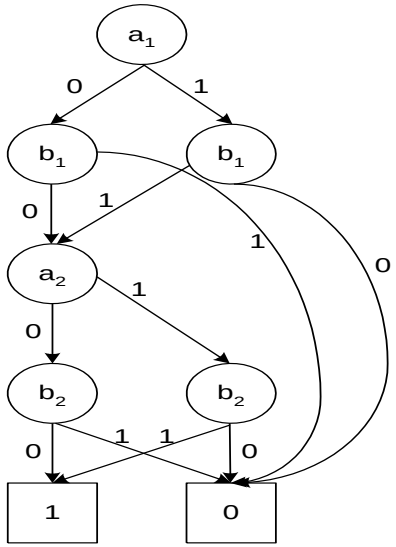
Бесконечные семейства конечных систем



Символьные алгоритмы

- Автоматическая проверка моделей:
  - Программы, её реализующие:
    - Алгоритмы + структуры данных = программы.
- Структуры данных:
  - Системы переходов
    1. состояния
    2. переходы между ними
  - Явное представление систем переходов
    1. перечисление
    2. списки смежности
  - Символьное представление систем переходов
    1. компактно заданное множество состояний (формула, интервал)
    2. компактно заданные переходы (формула, функция)

Символьные алгоритмы



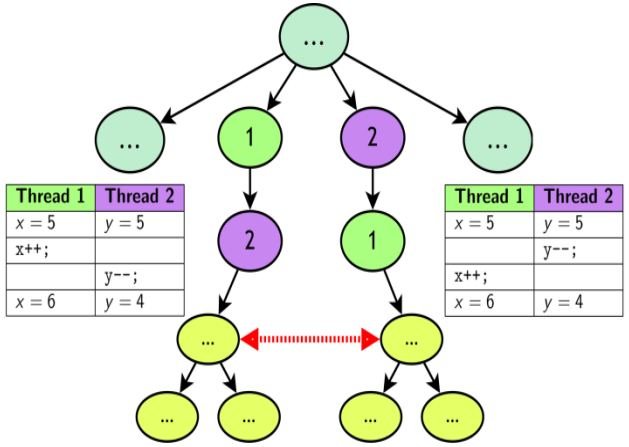
- Булевские модели
  - Бинарные Разрешающие Диаграммы BDD (Binary Decision Diagram)
    - Множество состояний, переход – булевская формула
    - Закодировать булевскую формулу как BDD

- Целочисленные модели
  - Формулы арифметики Пресбургера
    - Множество состояний, переход – формула арифметики Пресбургера

$x<100 \wedge y+z \geq 38 \rightarrow start_0$

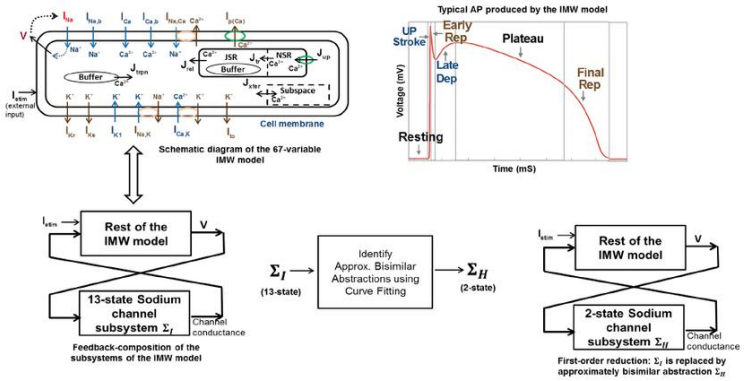
Редукция частичных порядков

- Асинхронные системы
  - Действия модулей не согласованы, нет глобальных часов.
- Независимость событий
  - События (переходы) независимы, если состояние, в которое приходит система в результате их осуществления, не зависит от порядка этих событий.
- Интерливинг (чередование)
  - Параллельные события в последовательности вычислений располагаются в произвольном порядке.
- Редукция частичных порядков
  - Последовательности вычислений разбиваются на независимые классы эквивалентности.

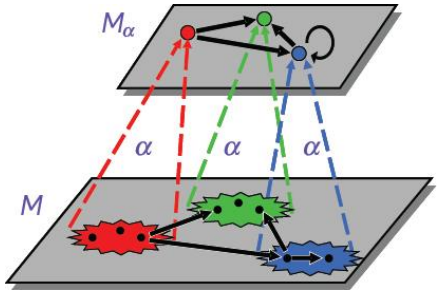


Другие подходы к проблеме взрыва состояний

- Композиционный вывод
  - Модульность системы
    - Локальные свойства → спецификация всей системы.
  - Сильная зависимость модулей
    - Локальные свойства с предположениями → спецификация всей системы.

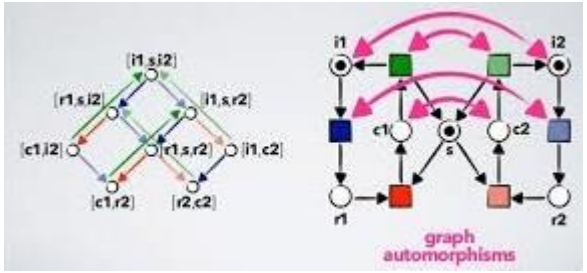


- Абстракция
  - Множество реальных значений данных системы кодируется множеством абстрактных значений.

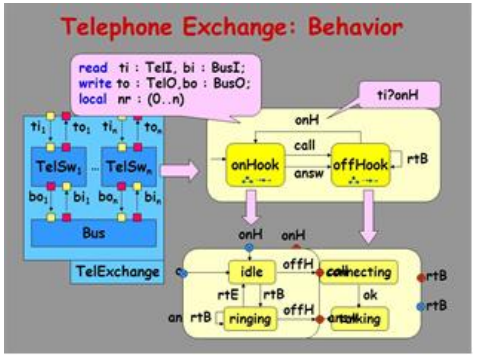


Другие подходы к проблеме взрыва состояний

- Симметрия
  - В системе параллельно действуют одинаковые процессы.



- Индукция
  - Параметризованные процессы
    - Инвариант поведения — спецификация
    - Индуктивно показать, что инвариант верен для всех значений параметра.



Моделирование систем

- *Формальная модель системы* для верификации свойств
  - Всё необходимое
  - Ничего лишнего
    - Цифровые схемы:
      - Логические ячейки и булевые значения
      - Не уровни напряжения
    - Коммуникационные протоколы:
      - Обмен сообщениями
      - Не содержание сообщений

Моделирование систем

- *Реагирующие системы* (Reactive systems)
  - Взаимодействие с окружением
  - Бесконечно долгая работа
  - Состояние
    - мгновенное описание системы, значение переменных системы в данный момент времени.
  - Переход между состояниями
    - Изменение состояний
    - Состояние до и состояние после
  - Вычисление
    - (Бес-)конечная последовательность состояний, каждое из которых получено из предыдущего посредством некоторого перехода.

Моделирование систем

- *Модель Крипке*
  - Граф переходов специального вида
    - Пути в графе соответствуют вычислениям системы
    - Простая и универсальная модель
- *Параллельные системы*
  - Текст программы или диаграмма
  - Типы систем
    - синхронные, асинхронные, с разделяемыми переменными, обменивающиеся сообщениями и т.п.
  - Описываются формулами логики предикатов первого порядка
    - Формулы корректно транслируются в модель Крипке

1. Моделирование параллельных систем

Множество атомарных высказываний AP

- x=13
- Двери лифта открыты
- Тает снег

- Модель Крипке M = (S, S_0, R, L)
  - S – конечное множество *состояний*
  - $S_0 \subseteq S$ – множество *начальных* состояний
  - $R \subseteq S \times S$ – тотальное *отношение переходов*: для каждого $s \in S$ существует $s' \in S$, такое что R(s,s').
  - $L : S \to 2^{AP}$ – функция *разметки*, сопоставляющая каждому состоянию множество атомарных высказываний, истинных в данном состоянии.
- Путь в модели M из состояния s
  - (бес-)конечная последовательность состояний модели $\pi = s_0, s_1, s_2, \ldots$, такая что $s_0 = s$ и для всех $i \geq 0$ верно, что $R(s_i, s_{i+1})$.

1.1. Представления первого порядка

- *Логика предикатов первого порядка*
  - Предикатные и функциональные символы
    - фиксированная интерпретация
  - Логические связки $\neg$, $\wedge$, $\vee$, $\rightarrow$
  - Кванторы $\forall$, $\exists$

- *Описание параллельных систем*
  - V = $\{v_0, v_1, v_2, \ldots v_n\}$ — переменные системы
  - D — домен интерпретации
  - Означивание для V — функция, сопоставляющая переменным из V их значения из D.
    - s: V $\rightarrow$ D

1.1. Представления первого порядка

- *Состояние* s — означивание s: V → D.
  - Формула, истинная *в точности* на данном означивании
    - V = {v_0, v_1, v_2}, D = {0, 1, 2}
      - s = {$v_0 \leftarrow 0$, $v_1 \leftarrow 1$, $v_2 \leftarrow 2$}:
        - φ: $(v_0 = 0) \wedge (v_1 = 1) \wedge (v_2 = 2)$
        - ~~ψ: $(v_0 = 0) \wedge (v_1 = 1) \vee (v_2 = 2)$~~
  - Формула представляет множество *всех* состояний, в которых она истинна.

- *Атомарные высказывания* AP
  - v=d, v∈V, d∈D:
    - v=d истинно в состоянии s, если s(v)=d.
  - При D={true, false}
    - вместо v=true пишем v, вместо v=false — ¬v

1.1. Представления первого порядка

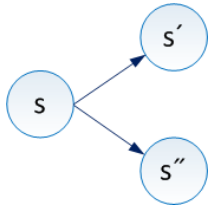
- *Состояние* s — означивание s: V → D.
  - Формула, истинная *в точности* на данном означивании
    - V = {v_0, v_1, v_2}, D = {0, 1, 2}
      - s = {$v_0 \leftarrow 0$, $v_1 \leftarrow 1$, $v_2 \leftarrow 2$}:
        - φ: $(v_0 = 0) \wedge (v_1 = 1) \wedge (v_2 = 2)$
        - ~~ψ: $(v_0 = 0) \wedge (v_1 = 1) \vee (v_2 = 2)$~~
  - Формула представляет множество *всех* состояний, в которых она истинна.

- *Атомарные высказывания* AP
  - v=d, v∈V, d∈D:
    - v=d истинно в состоянии s, если s(v)=d.
  - При D={true, false}
    - вместо v=true пишем v, вместо v=false — ¬v

1.1. Представления первого порядка

- *Переходы* между состояниями
  - Формулы для представления множества упорядоченных пар состояний
  - Пусть V — переменные системы, V’ — копии переменных системы
    - V — переменные *текущего* состояния
    - V’ — переменные *следующего* состояния

  - Любое означивание V и V’ *кодирует переход*
    - Означивание кодируется формулой первого порядка
    - $s = \{v_0 \leftarrow 0, v_1 \leftarrow 1, v_2 \leftarrow 2\}$, $s' = \{v'_0 \leftarrow 1, v'_1 \leftarrow 0, v'_2 \leftarrow 0\}$, $s'' = \{v'_0 \leftarrow 2, v'_1 \leftarrow 0, v'_2 \leftarrow 0\}$:



      - $(v_0 = 0) \wedge (v_1 = 1) \wedge (v_2 = 2) \wedge (v'_0 = 1) \vee (v'_0 = 2) \wedge \forall\, v \in \{v'_1, v'_2\} : (v = 0)$.

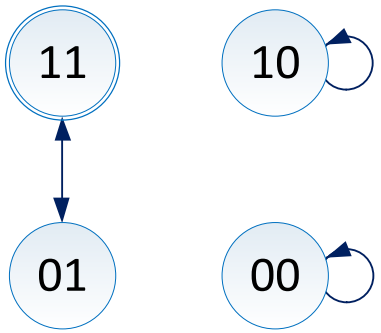
  - $R(V,V')$ — формула первого порядка, представляющая отношение переходов R

1.1. Представления первого порядка

- *Построение модели* Крипке M = (S, S_0, R, L) по формулам первого порядка S_0 и R(V,V').
  - *Состояния S* — множество всех означиваний переменных V
  - *Начальные состояния* S_0 — множество всех означиваний s_0 переменных V, которые удовлетворяют формуле S_0.
  - *Переходы R* — R(s,s') верно для всех s, s' таких, что формула *R* истинна при означиваниях s(v) и s(v') для всех $v \in V$ и $v' \in V'$.
    - Может оказаться, что какое-то состояние s не имеет последователей.
      - Дополним отношение R так, чтобы было верно R(s,s).
  - *Разметка* $L : S \to 2^{AP}$ — для всех состояний s множество L(s) — все атомарные высказывания, истинные в s.
    - Если переменная v — логическая, то пишем $v \in L(s)$ или $v \notin L(s)$.

1.1. Представления первого порядка

- Пример системы
  - Пусть V={x, y}, D={ 0, 1}
  - Означивания — пары $(d_1, d_2) \in D \times D$
  - Переход: x:=(x+y)(mod2),
    начальное состояние x=1 и y=1.
- Представление первого порядка
  - $S_0(x, y) \equiv (x=1) \wedge (y=1)$
  - $R(x, y, x', y') \equiv x'=(x+y)(\text{mod}2) \wedge (y'=y)$
- Модель Крипке M = (S, S_0, R, L)
  - S = D×D
  - S_0={(1,1)}
  - R={((1,1), (0,1)), ((0,1), (1,1)), ((1,0), (1,0)), ((0,0), (0,0))}
  - L((0,0))={x=0,y=0}, L((0,1))={x=0,y=1}, L((1,0))={x=1,y=0}, L((1,1))={x=1,y=1}
- Единственный путь в модели. Вычисление системы
  - (1,1), (0,1), (1,1), (0,1)…



1.2. Степень детализации переходов

- Гранулярность переходов
  - Излишняя детализация
    - Состояние не должно быть результатом выполнения только части действия
  - Излишнее огрубление
    - Состояния не должны пропадать из-за «огрубления» действий, когда последовательность действий представляется как одно действие.

1.2. Степень детализации переходов

- Пример
  - α: `x := x + y;` β: `y := y + x;`
  - `x = 1, y = 2`
  - α_0: `load` R_1, `x` β_0: `load` R_2, `y`
  - α_1: `add` R_1, `y` β_1: `add` R_2, `x`
  - α_2: `store` R_1, `x` β_2: `store` R_2, `y`
  - $\alpha\beta$: `x = 3, y = 5`
  - $\beta\alpha$: `x = 4, y = 3`
  - $\alpha_0\beta_0\alpha_1\beta_1\alpha_2\beta_2$: `x = 3, y = 3`

2. Параллельные системы

- Параллельные системы
  - Набор компонентов, которые исполняются одновременно.
  - Компоненты могут взаимодействовать друг с другом.
  - Типы исполнения и взаимодействия могут быть разными.
    - синхронное
    - асинхронное
    - общие переменные
    - общение через каналы связи

2. Параллельные системы

- Параллельные системы
  - Исполнение:
    - асинхронное
      - в любой момент времени только один компонент делает шаг вычисления
    - синхронное
      - все компоненты делают шаг вычисления одновременно
  - Взаимодействие:
    - изменения значения общих переменных
    - обмен сообщениями
      - очереди
      - протоколы синхронной взаимосвязи

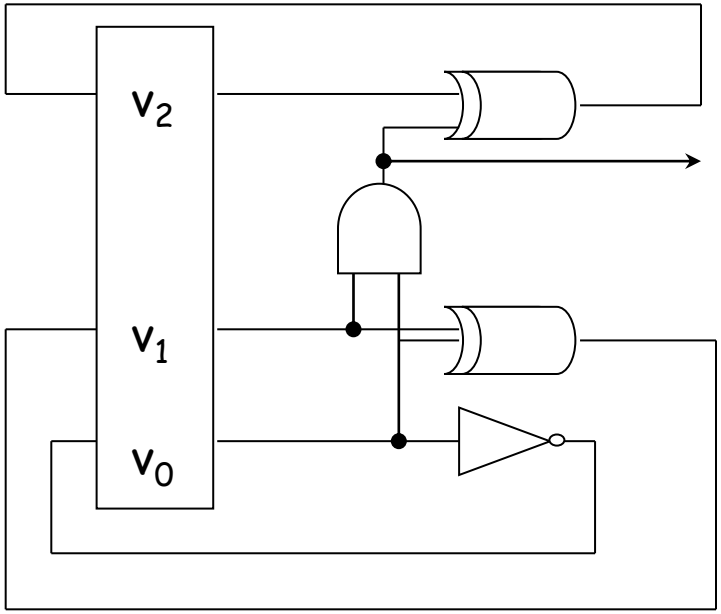
2.1. Цифровые схемы

- V — множество элементов, определяющих состояние схемы
  - Элементы принимают значения 0 или 1.
  - Для синхронных схем множество V
    - выходы всех регистров схемы
    - всевозможные первичные входы
  - Для асинхронных схем множество V
    - все соединения
- Каждому элементу из V сопоставляется логическая переменная
  - Состояние определяется означиванием переменных как 0 или 1
  - Для любого означивания есть формула, истинная только на нём
    - $V = \{v_1, v_2\}$ и $(v_1 \leftarrow 1, v_2 \leftarrow 0) : v_1 \wedge \neg v_2$
- Для описания электронных схем не требуется логики предикатов первого порядка.
  - Достаточно булевых формул
  - Пусть булевы формулы
    - S_0 — множество начальных состояний схемы
    - $R(V,V')$ — отношение переходов схемы

2.1. Цифровые схемы

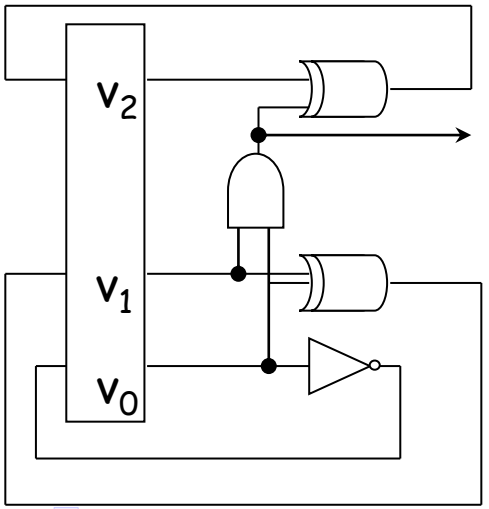
- Синхронные схемы
  - Работа состоит из последовательности шагов
    1. На каждом шаге изменяются входы схемы, пока она не стабилизируется.
    2. Через схему проходит импульсный синхронизирующий сигнал, и значение элементов изменяется.

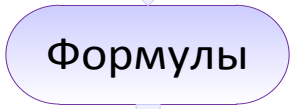
- Счетчик по модулю 8.

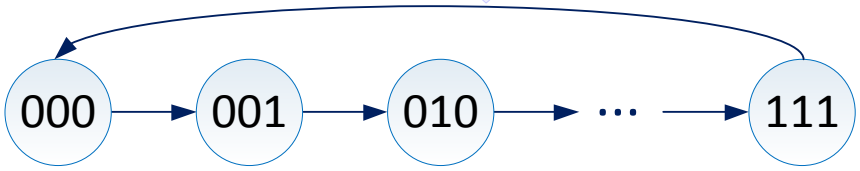


2.1. Цифровые схемы

- Счетчик по модулю 8.
  - $V = \{v_0, v_1, v_2\}$ —переменные состояния схемы,
  - $V` = \{v`_0, v`_1, v`_2\}$ — копия переменных состояния.
  - Переходы счетчика:
    - $v`_0 = \neg v_0$
    - $v`_1 = v_0 \oplus v_1$
    - $v`_2 = (v_0 \wedge v_1) \oplus v_2$
  - Отношения переходов:
    - $R_0(V,V`) \equiv (v`_0 \Leftrightarrow \neg v_0)$
    - $R_1(V,V`) \equiv (v`_1 \Leftrightarrow v_0 \oplus v_1)$
    - $R_2(V,V`) \equiv (v`_2 \Leftrightarrow (v_0 \wedge v_1) \oplus v_2)$
  - Все изменения происходят одновременно:
    - $R(V,V`) \equiv R_0(V,V`) \wedge R_1(V,V`) \wedge R_2(V,V`)$.

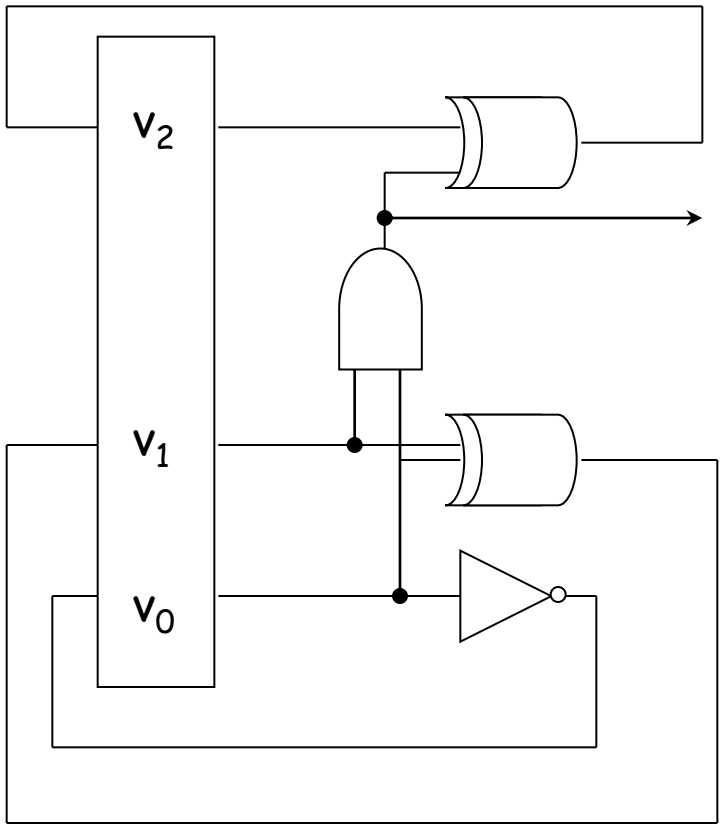






2.1. Цифровые схемы

- В общем случае $V = \{v_0,\ldots,v_n\}$ и $V` = \{v`_0,\ldots,v`_n\}$
- Каждой переменной состояния $v`_i$ сопоставляется булева формула $f_i(V)$, так что $v`_i = f_i(V)$.
- Определяются отношения
  - $R_i(V,V`) \equiv (v`_i \Leftrightarrow f_i(V))$
- Отношение переходов — конъюнкция
  - $R(V,V`) \equiv R_0(V,V`) \wedge \ldots \wedge R_n(V,V`)$.
- Отношение переходов синхронной схемы представляется конъюнкцией отношений переходов для отдельных процессов.



2.1. Цифровые схемы

- Асинхронные схемы
  - Отношение переходов выражается дизъюнкцией.

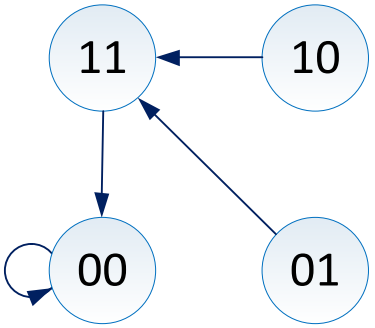
- Пусть все компоненты схемы имеют ровно один выход и не имеют внутренних переменных состояния.
  - Каждый компонент определяется функцией $f_i(V)$
    - по значениям переменных текущего состояния из V компонент выдает на выходе значение $f_i(V)$.
  - Если у компонента несколько выходов – используем соответствующее количество функций $f^1_i(V)$ … $f^k_i(V)$

2.1. Цифровые схемы

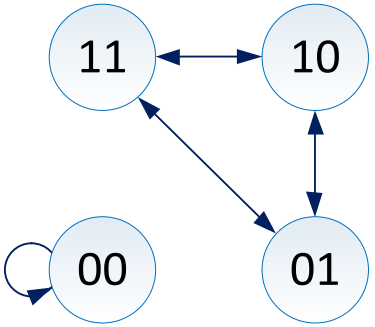
- Считается, что значение компонента изменяется достаточно быстро для того, чтобы два компонента изменили свое значение одновременно.
  - Используется семантика чередования
    - в каждый момент времени только один компонент изменяет свое состояние
  - Дизъюнкция $R(V,V') \equiv R_0(V,V') \vee \ldots \vee R_n(V,V')$
    - где $R_i(V,V') \equiv (v'_i \Leftrightarrow f_i(V)) \wedge \bigwedge_{j \neq i} (v'_j \Leftrightarrow v_j)$
- Состояния некоторых компонентов могут изменяться намного чаще, чем других.
  - Во многих случаях это маловероятно.
    - В модель вводятся дополнительные ограничения *справедливости*, которые не будут допускать подобного поведения системы.

2.1. Цифровые схемы

- Разница между синхронной и асинхронной моделями
  - Пусть V = {v_0,v_1}, $v`_0 = v_0 \oplus v_1$ и $v`_1 = v_0 \oplus v_1$
  - Пусть s — состояние, в котором $v_0 = 1 \wedge v_1 = 1$
  - Синхронная модель
    - единственный последователь состояния s
      - состояние $v_0 = 0 \wedge v_1 = 0$



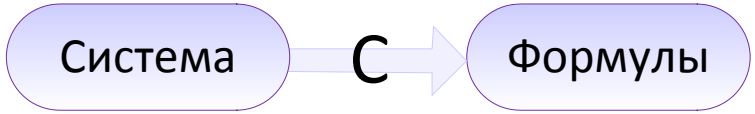
  - Асинхронная модель
    - состояние s имеет двух последователей:
      - $v_0 = 0 \wedge v_1 = 1$
      - $v_0 = 1 \wedge v_1 = 0$



2.2. Программы

- *Последовательные программы*
  - Состоит из последовательности операторов
    - присваивание, условный оператор, цикл и т.п.

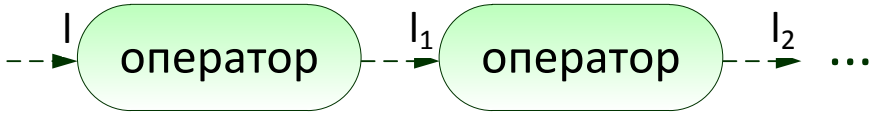
- *Механизм трансляции C* преобразует
  - текст последовательной программы P
    - *в формулу первого порядка R*, представляющую множество переходов программы.



- Каждый оператор имеет единственную точку входа и единственную точку выхода.
  - Эти точки помечены уникальным образом.

2.2. Программы

- Определим преобразование разметки
  - по заданной неразмеченной программе P строит размеченную программу P^L
    - Приписать метку точке входа каждого оператора в P, кроме самой программы P
      - Все метки считаются попарно различными.
    - Пометить точки входа и выхода программы P



- Уникальная разметка точек входа и выхода для всех операторов программы.

2.2. Программы

- Для заданного оператора P определим размеченный оператор P^L
  - P не составной (т. е. `x:=e`, **`skip`**, **`wait`**, **`lock`**, **`unlock`** и т. п.)
    - $P^L = P$
  - $P = P_1 ; P_2$
    - $P^L = P^L_1 ; l_2 : P^L_2$
  - P = **`if`** b **`then`** P_1 **`else`** P_2 **`fi`**
    - P^L = **`if`** b **`then`** $l_1 : P^L_1$ **`else`** $l_2 : P^L_2$ **`fi`**
  - P = **`while`** b **`do`** P_1 **`od`**
    - P^L = **`while`** b **`do`** $l_1 : P^L_1$ **`od`**

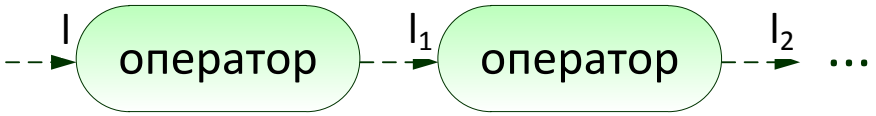
2.2. Программы

- Разметка оператора P — точка вход m и точка выхода m`
- *pc* — счетчик команд
  - значения — метки программы и $\perp$
    - *pc* = $\perp$ если компонента параллельной программы неактивна.

- *Для формул системы*: определить переменные, начальные состояния и переходы
- V — *множество переменных* программы и *pc* — счётчик.
- V` — копия множества переменных программы и *pc*` — копия счётчика.
- same(Y) = $\wedge_{y \in Y}(y`=y)$
- Формула для *множества начальных состояний* программы P.
  - Задано начальное условие pre(V)
    - $S_0(V,pc) = pre(V) \wedge pc = m$.

2.2. Программы

- Процедура трансляции C зависит от параметров:
  - входная метка I
  - размеченный оператор P
  - выходная метка I`.
  - Определяется рекурсивно.
- Формула *C(I, P, I`)* представляет множество переходов программы P в виде *дизъюнкции всех переходов* этого множества.
  - Каждый дизъюнкт
    - соответствует отдельному переходу
    - определяет условия перехода:
      - значение булевого условия
      - значение счетчика команд
    - Истинен, когда переход осуществим
    - Ложен, иначе.



2.2. Программы

- Оператор присваивания
  - $C(l, \mathbf{v:=e}, l') \equiv pc = l \wedge pc' = l' \wedge v' = e \wedge same(V \setminus \{v\})$
- Оператор **skip**
  - $C(l, \mathbf{skip}, l') \equiv pc = l \wedge pc' = l' \wedge same(V)$
- Последовательная композиция операторов
  - $C(l, P_1 \mathbf{;} l'' \mathbf{:} P_2, l') \equiv C(l, P_1, l'') \vee C(l'', P_2, l')$

2.2. Программы

- Оператор ветвления **`if-then-else`**
  - C(l, **`if`** b **`then`** l_1: P_1 **`else`** l_2: P_2 **`end if`**, l`) дизъюнкция формул:
    1. $pc = l \wedge pc` = l_1 \wedge b \wedge same(V)$
    2. $pc = l \wedge pc` = l_2 \wedge \neg b \wedge same(V)$
       - переходы, изменяющие показания счетчика команд.
    3. $C(l_1, P_1, l`)$
    4. $C(l_2, P_2, l`)$
       - формулы для переходов P_1 и P_2
  - Аналогично для оператора недетерминированного выбора между несколькими альтернативами.

2.2. Программы

- Оператор итерации **`while-do`**
  - C(l, **`while`** b **`do`** l_1: P_1 **`end while`**, l`) дизъюнкция формул:
    1. pc = l $\wedge$ pc` = l_1 $\wedge$ b $\wedge$ same(V)
       - условие b истинно и следующий оператор — P_1.
    2. pc = l $\wedge$ pc` = l` $\wedge$ $\neg$b $\wedge$ same(V)
       - условие b ложно, цикл завершается.
    3. C(l_1, P_1, l)
       - формула для переходов оператора P_1
  - Точка выхода оператора P_1 совпадает с точкой входа в оператор итерации.

2.3. Параллельные программы

- Параллельная программа
  - Множество процессов исполняются параллельно
    - Процесс — последовательность операторов
    - Взаимодействующие процессы
- Асинхронные программы
  - Ровно один процесс может совершить переход в каждый момент времени.
- V_i — множество переменных, изменяемых процессом P_i
  - Эти множества могут пересекаться
- pc_i — счетчик команд процесса P_i
- PC — множество всех счетчиков
- Параллельная программа P:
  - **cobegin** $P_1 \parallel P_2 \parallel ... \parallel P_n$ **coend**
    - P_1,..., P_n — процессы

2.3. Параллельные программы

- Параллельная программа P:
  - **cobegin** $P_1 \| P_2 \| \ldots \| P_n$ **coend**
    - $P_1,\ldots, P_n$ — процессы
- Новое правило преобразования разметки
  - Параллельная программа имеет статус оператора в составе последовательной программы
  - Метка в точке входа и точке выхода каждого оператора
  - Никакая точка выхода параллельного процесса
    - *не отождествляется* с какой-либо точкой входа
  - Точки выхода процессов помечаются отдельно.
    - все метки попарно различны
    - точки входа и выхода программы P: m и m`
- P = **cobegin** $P_1 \| P_2 \| \ldots \| P_n$ **coend**
  - P^L = **cobegin** $l_1: P^L_1\ l`_1 \| l_2: P^L_2\ l`_2 \| \ldots \| l_n: P^L_n\ l`_n$ **coend**

2.3. Параллельные программы

- *Для формул системы*: определить переменные, начальные состояния и переходы

- V — *множество переменных и счетчиков* программы.
- V` — копия множества переменных и счетчиков программы.
- *Начальные состояния* параллельной программы P
  - $S_0(V,PC) = pre(V) \wedge pc = m \wedge \wedge^n_{i=1} (pc_i = \perp)$
    - $pc_i = \perp$
      - процесс P_i еще не был активизирован и не исполняется

2.3. Параллельные программы

- C(l, **cobegin** $P_1 \parallel P_2 \parallel \ldots \parallel P_n$ **coend**, l`) дизъюнкция формул:
  1. $pc = l \wedge pc`_1 = l_1 \wedge \ldots \wedge pc`_n = l_n \wedge pc` = \bot$
     - Инициализация параллельных процессов
       - Переход из точки входа **cobegin** в точки входа процессов
  2. $pc = \bot \wedge pc_1 = l`_1 \wedge \ldots \wedge pc_n = l`_n \wedge pc` = l` \wedge \wedge^n_{i=1} (pc`_i = \bot)$
     - Завершение параллельной программы
       - Переход из точек выхода процессов в точку выхода оператора **cobegin**
         - Исполняется при завершении всех процессов
  3. $\vee^n_{i=1} (C(l_i, P_i, l`_i) \wedge same(V \backslash V_i) \wedge same(PC \backslash \{pc_i\}))$
     - Чередующееся исполнение параллельных процессов
       - Конъюнкция:
         - формула для отношения переходов процесса P_i
         - формула неизменности переменных и счётчиков других процессов, в каждый момент времени только один процесс может совершить переход.

2.3. Параллельные программы

Разделяемые переменные

- Программы с разделяемыми переменными
  - параллельные программы, для которых множества переменных V_i, изменяемых процессами, пересекаются

- Процедура трансляции С для общепринятых операторов синхронизации процессов (wait, lock, unlock…)
  - Операторы синхронизации
    - средства исключительного доступа процессов к разделяемым переменным
    - атомарные операторы, обрабатываются преобразованием разметки обычным образом.

2.3. Параллельные программы

- Оператор **`wait(b)`**
  - Реализация оператора посредством ожидания по условию
    - программы с конечным числом состояний
    - не рассматриваем реализации, требующие сложных структур данных (очереди процессов)
  - **`wait(b)`** периодически проверяет значение булевой переменной **`b`**, пока не узнает, что **`b`** стала истиной, после чего осуществляется переход к следующему оператору программы
  - C(l, **`wait(b)`**, l`) дизъюнкция формул
    - $pc_i = l \wedge pc`_i = l \wedge \neg b \wedge same(V_i)$
    - $pc_i = l \wedge pc`_i = l` \wedge b \wedge same(V_i)$

2.3. Параллельные программы

- Оператор **`lock(v)`**.
  - Действует как **`wait(v = 0)`** и при выходе значение **`v`** изменяется на 1.
  - Применяется, чтобы обеспечить взаимное исключение
    - В критической секции не больше одного процесса
  - C(l, **`lock(v)`**, l`) дизъюнкция формул:
    - $pc_i = l \wedge pc`_i = l \wedge v = 1 \wedge same(V_i)$
    - $pc_i = l \wedge pc`_i = l` \wedge v = 0 \wedge v` = 1 \wedge same(V_i \setminus \{v\})$
- Оператор **`unlock(v)`**
  - Присваивает значение 0 переменной v
    - Дает разрешение процессу на вход в критическую секцию
  - C(l, **`unlock(v)`**, l`) $\equiv pc_i = l \wedge pc`_i = l` \wedge v` = 0 \wedge same(V_i \setminus \{v\})$

3. Пример трансляции программы

- Взаимное исключение
  - P = m: **cobegin** P_0 || P_1 **coend** m`

```
P0:: l0: while True do
           NC0: wait (turn = 0);
           CR0: turn := 1;
         end while
      l`0

P1:: l1: while True do
           NC1: wait (turn = 1);
           CR1: turn := 0;
         end while
      l`1
```

- pc — счетчик команд P
  - pc = m — входная метка P
  - pc = m` — выходная метка P
  - pc = ⊥, когда P_0 и P_1 активны
- pc_i — счетчик команд процессов P_i
  - его значения: l_i, $l`_i$, NC_i, CR_i и ⊥
- Разделяемая переменная **turn**.
- $V = V_0 = V_1 = \{\textbf{turn}\}$ и $PC = \{pc, pc_0, pc_1\}$.
- $pc_i = CR_i$: процесс i находится в критической секции.
  - Запрещается находиться в критической секции одновременно.
- $pc_i = NC_i$: процесс i пребывает в некритической секции.
  - Процесс дожидается выполнения **turn** = i для входа в критическую секцию.

3. Пример трансляции программы

- Начальные состояния параллельной программы P:
  - $S_0(V,PC) \equiv pc = m \wedge pc_0 = \bot \wedge pc_1 = \bot$
    - Ограничений на значение turn не налагается
- В результате трансляции C формула для отношения переходов R(V, PC, V`, PC`) дизъюнкция формул:
  - $pc = m \wedge pc`_0 = l_0 \wedge pc`_1 = l_1 \wedge pc` = \bot$
  - $pc_0 = l`_0 \wedge pc_1 = l`_1 \wedge pc` = m` \wedge pc`_0 = \bot \wedge pc`_1 = \bot$
  - $C(l_i, P_i, l`_i) \wedge same(V \backslash V_i) \wedge same(PC \backslash \{pc_i\})$
    - $C(l_i, P_i, l`_i) \wedge same(pc, pc_{\neg i}), i \in \{0,1\}$

P_0:: l_0: **while** ***True*** **do**
NC_0: **wait** **(turn = 0);**
CR_0: **turn := 1;**
end while
$l`_0$

P_1:: l_1: **while** ***True*** **do**
NC_1: **wait** **(turn = 1);**
CR_1: **turn := 0;**
end while
$l`_1$

3. Пример трансляции программы

- Для каждого P_i формула $C(l_i, P_i, l`_i)$ — дизъюнкция формул:
  - $pc_i = l_i \wedge pc`_i = NC_i \wedge True \wedge same(\mathbf{turn})$
  - $pc_i = NC_i \wedge pc`_i = NC_i \wedge \mathbf{turn} \neq i \wedge same(\mathbf{turn})$
  - $pc_i = NC_i \wedge pc`_i = CR_i \wedge \mathbf{turn} = i \wedge same(\mathbf{turn})$
  - $pc_i = CR_i \wedge pc`_i = l_i \wedge \mathbf{turn} = (i+1)(mod\ 2)$
  - $pc_i = l_i \wedge pc`_i = l`_i \wedge False \wedge same(\mathbf{turn})$

```
P0:: l0: while True do
            NC0: wait (turn = 0);
            CR0: turn := 1;
         end while
      l`0

P1:: l1: while True do
            NC1: wait (turn = 1);
            CR1: turn := 0;
         end while
      l`1
```

3. Пример трансляции программы

- Модель Крипке строится по формулам S_0 и R
  - Процессы никогда не окажутся в своих критических секциях одновременно
    - Свойство взаимного исключения
  - Не обеспечивается отсутствие удушения (starvation)
    - один процесс может непрерывно пытаться войти в свою критическую секцию, никогда не получая при этом доступа туда, и при этом другой процесс находится в своей критической секции бесконечно долго.