Верификация параллельных программных и аппаратных систем

Курс лекций

Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского политехнического университета

karpov@dcn.infos.ru

План курса

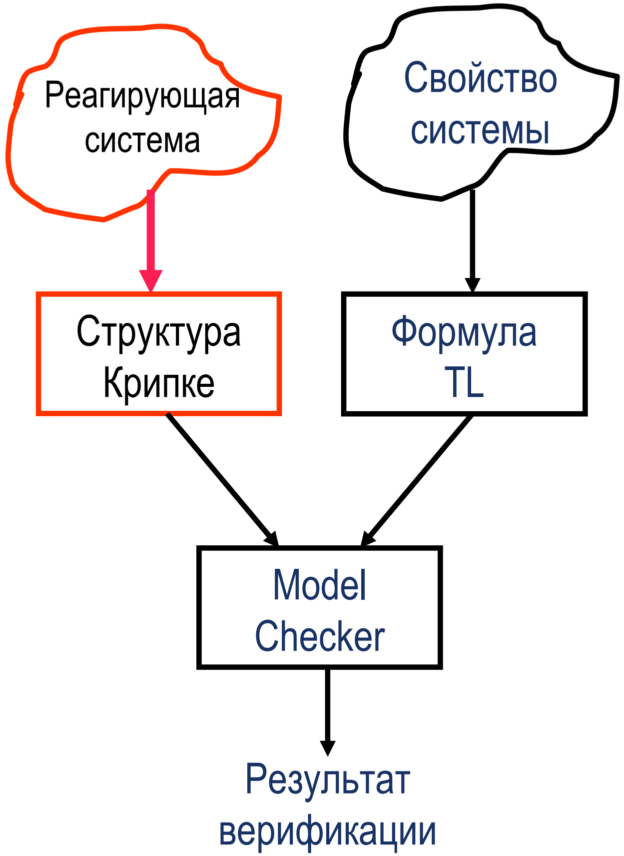
1. Введение
2. Метод Флойда-Хоара доказательства корректности программ
3. Исчисление взаимодействующих систем (CCS) Р.Милнера
4. Темпоральные логики
5. Алгоритм model checking для проверки формул CTL
6. Автоматный подход к проверке выполнения формул LTL
7. Структура Крипке как модель реагирующих систем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

Лекция 7

Структура Крипке как модель реагирующих систем

Как строить структуру Крипке для программных процессов -общие положения

- В абстрактную модель (структуру Крипке) перевод из реальной системы часто неформален (и неоднозначен)
- Структура Крипке – конечная система переходов. В структуре Крипке многие свойства системы, содержащей параметры с бесконечной областью определения, не сохраняются. *Т.о., не все реальные свойства могут быть проверены*
- Например, известно, что проблема останова неразрешима. А свойство «программа завершится» - проверяется в структуре Крипке
- Для систем с конечным числом состояний автоматический перевод возможен в эквивалентную структуру Крипке
- Для реальных систем построение соответствующей структуры Крипке часто проводится в два этапа:
  - перевод в программу на ЯВУ, в которой все параметры ограничиваются конечными областями определения (ручное построение)
  - после этого структура Крипке строится автоматически



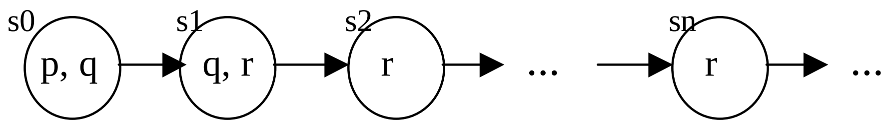
Трансформационные программы vs Реагирующие системы (Amir Pnueli, 1977)

Тип системы	***Трансформационные программные системы***	***Реагирующие программные системы***
Примеры	Пакетная обработка данных, вычисление функции, распознавание образов, ...	Протоколы, системы логического управления, операционные системы, ...
Цель	Получение значения как функции исходных данных	Обеспечение определенных правил взаимодействия с окружением
Вычисления	Всегда конечны с получением результата в конце вычислений	Всегда бесконечны; системы никогда не завершаются
Семантика	*Функция*: выход программы является функцией от входных данных	*Поведение*: последовательность реакций системы на внешние события
Спецификация (требования к системе)	Например, в виде логических пред- и постусловий: {Pre} S {Post}	Например, в виде множества формул темпоральной логики: $M \models \mathbf{G}(p \Rightarrow \mathbf{F}q)$

Моделирование систем

Класс систем – реагирующие системы. Предмет анализа – их *поведение*

Стандартной моделью реагирующих систем являются системы переходов – состояния и переходы между ними



Состояние описывает информацию о системе в некоторый момент времени

Состояние светофора – какой горит свет

Состояние программы – текущие значения программных переменных И счетчик команд, указывающий на ту команду, которая будет выполняться следующей

Состояние аппаратной схемы – состояние регистров и входов

Атомарные предикаты – соотношения между параметрами состояния

Моделирование систем (2)

Предмет анализа – *поведение систем*

Что такое поведение?

Поведение – (бесконечная) последовательность состояний

Что такое состояние?

Состояние – набор значений переменных и программного счетчика

Что такое переход из состояния в состояние?

Переход – изменение состояния, например, при выполнении оператора программы

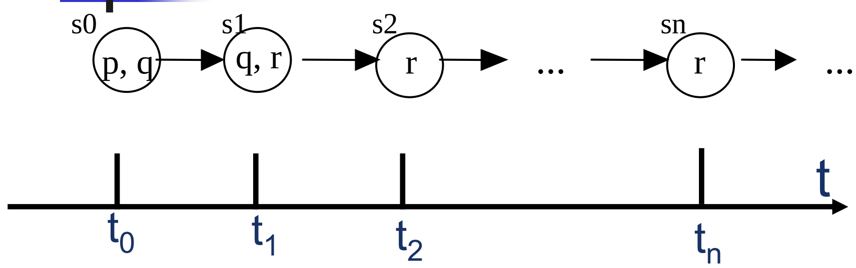
Как реальная система переходит из одного состояния в другое?

Какова протяженность перехода во времени?

Что происходит в процессе перехода?

При построении модели мы от этого абстрагируемся!

Мгновенность переходов



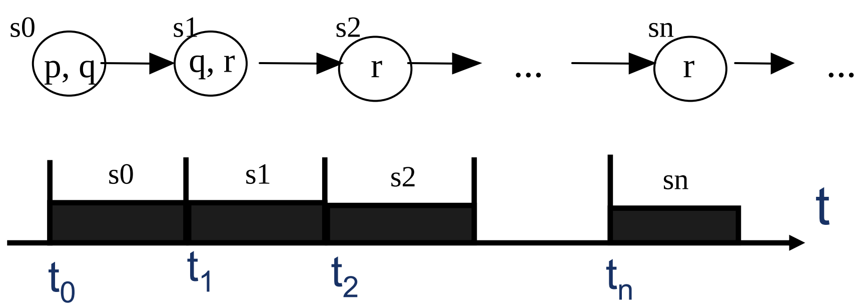
Как можно говорить о поведении системы (во времени), не упоминая время?

В темпоральной логике используется только порядок событий (состояний)

Принимается, что система находилась в некотором *стабильном* состоянии, а потом перешла в другое (стабильное) под влиянием выполнения оператора или события

Но как она перешла? Что было в процессе перехода? *Мы этим заниматься не хотим!!!*

При выполнении перехода система не проходит промежуточные состояния. Не существует части перехода. Система переходит из состояния в состояние *"мгновенно"*



Мы не можем использовать термин *«мгновенно»*, это связано со временем

Вместо мгновенности мы говорим об «*атомарности*» переходов

Фрагменты программ, системы переходов и структуры Крипке

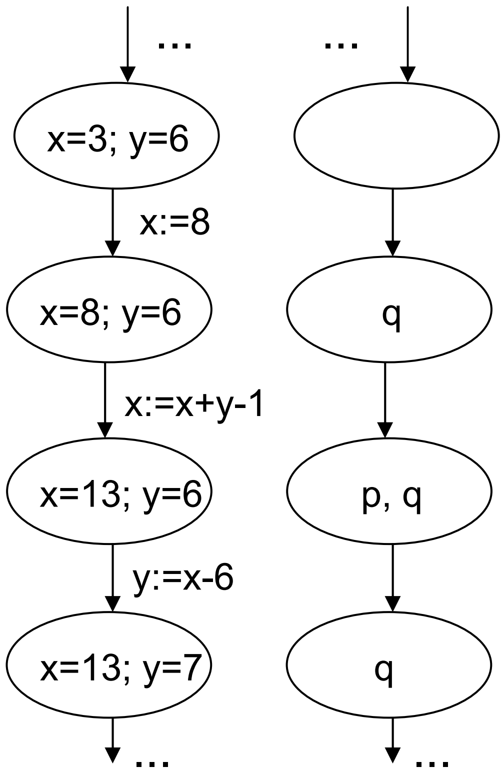
x=3, y=6

P::

k0: x:= 8;
k1: x:= x+y-1;
k2: y:= x-6;
k3: end

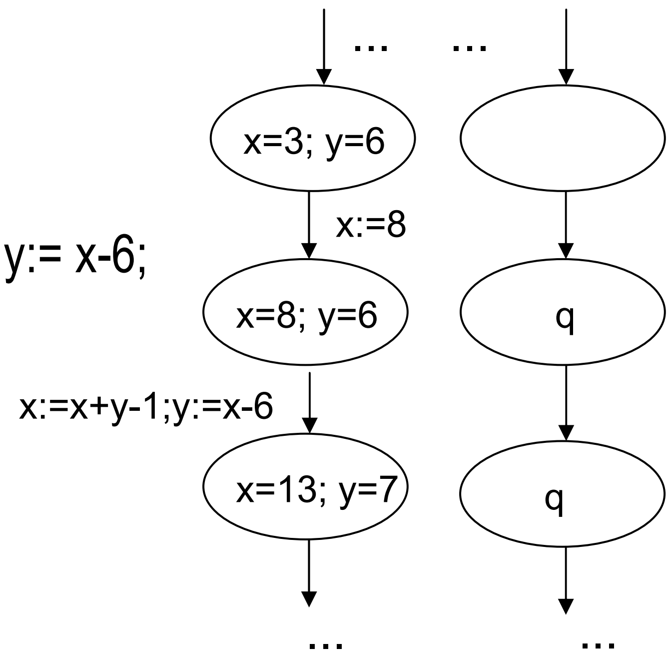
p = x>2*y

q = y<x



P::

k0: x:= 8;
k1: x:= x+y-1; y:= x-6;
k3: end



Гранулярность переходов – до какого предела?

На что это влияет?

Чем руководствоваться при построении модели?

Finite-state programs : атомарность и гранулярность переходов

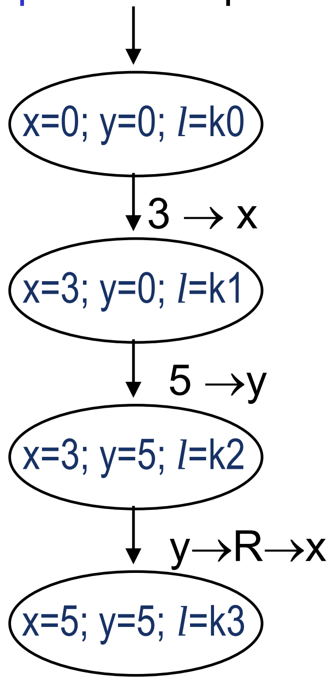
Доступ к памяти - атомарный: невозможно “увидеть” часть слова

Операции процессора - атомарные: невозможно “увидеть” часть результата

x=0, y=0

P1::

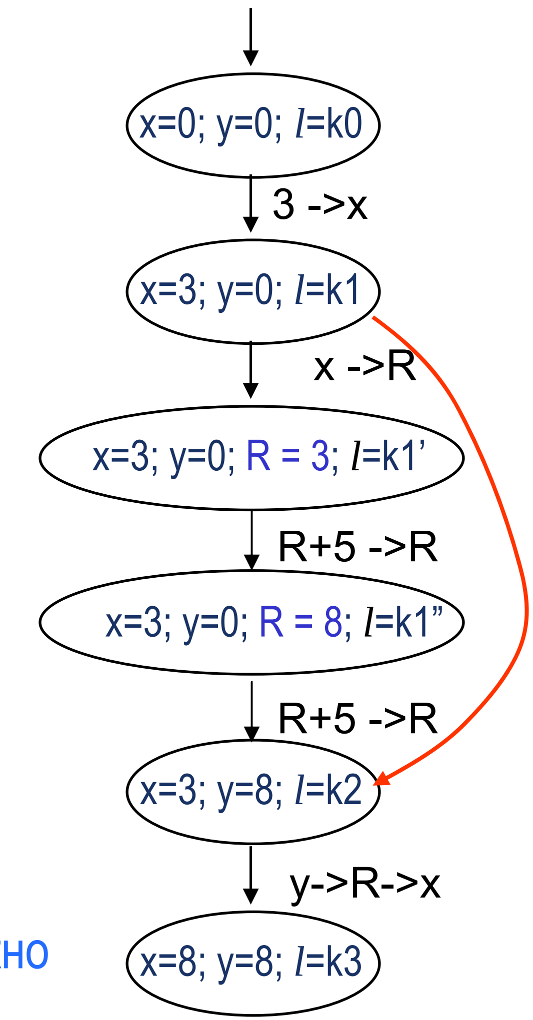
k0: x:= 3;
k1: y:= 5;
k2: x:= y;
k3: end



x=0, y=0

P2::

k0: x:= 3;
k1: y:= x + 5;
k2: x:= y;
k3: end



x := 3 - обычно одна операция процессора
x := y - обычно две операции процессора

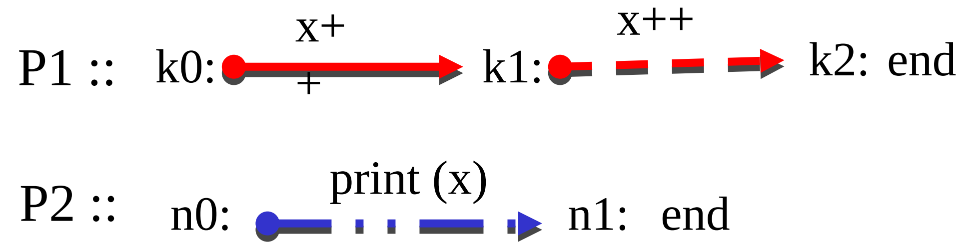
y:= x + 5; - обычно три операции процессора

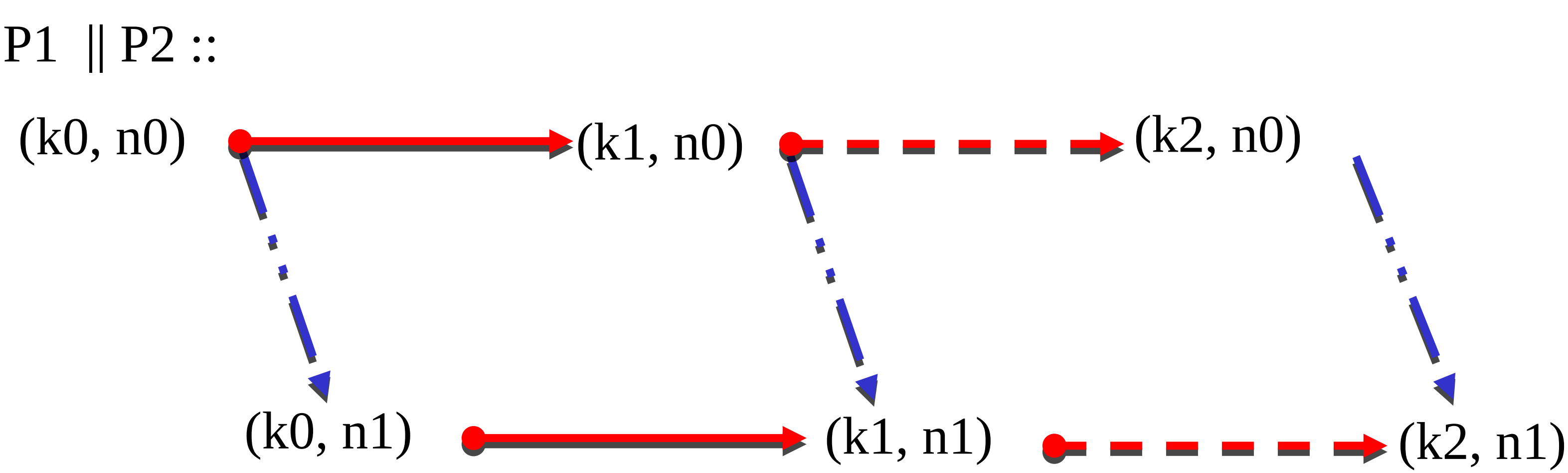
Где это важно? В последовательных системах обычно неважно

Эффект интерливинга

x=0

P1::	||	P2::
k0: x++;		n0: print(x);
k1: x++;		n1: end
k2: end		



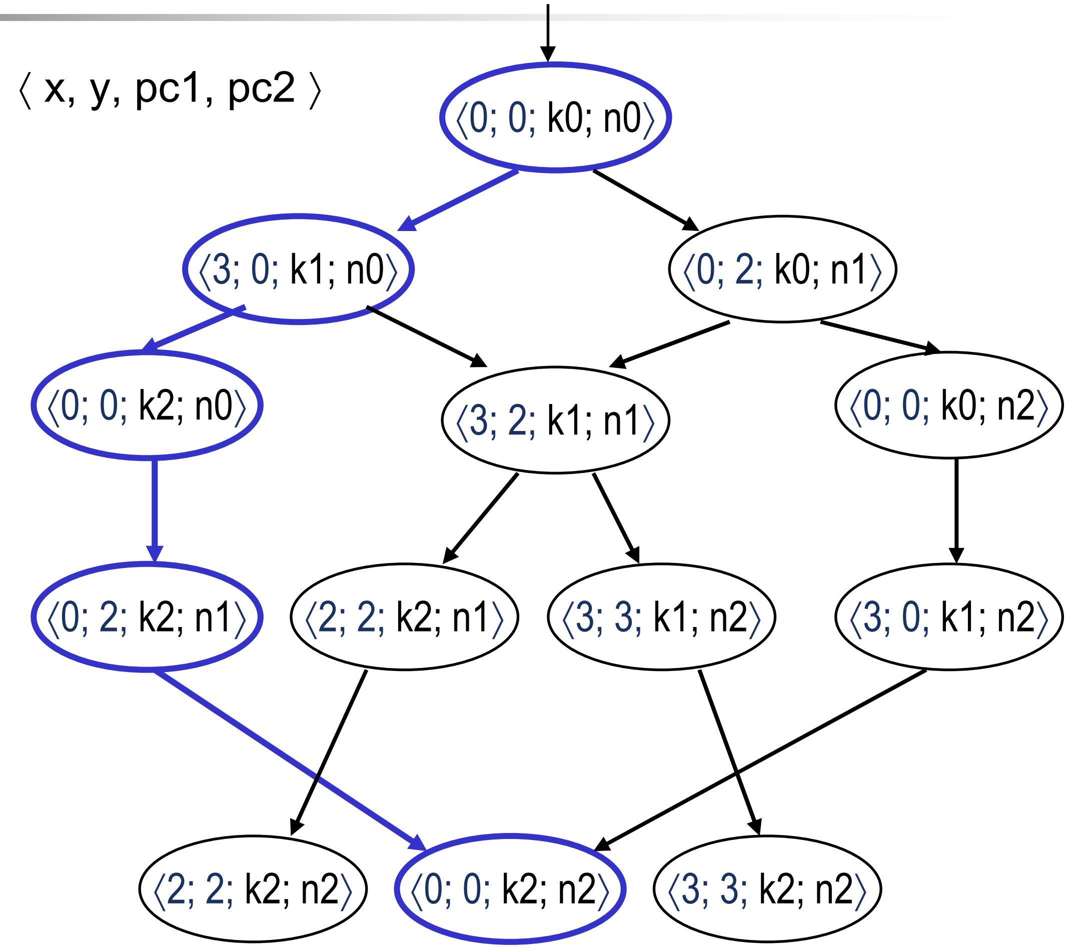


Как результат можем получить на печати 0, 1 или 2

Параллельные процессы: Интерливинг

x=0, y=0

P1::		P2::
k0: x:= 3;	||	n0: y:= 2;
k1: x:= y;		n1: y:= x ;
k2: end		n2: end



При анализе параллельных процессов нельзя делать никаких предположений об относительных скоростях выполнения процессов или стратегии планировщика процессов.

Непредсказуемость результатов выполнения параллельных процессов

Большинство ошибок в параллельных программах – из-за непредвиденных перекрытий операций параллельных процессов. Например, ошибки, найденные в DeepSpace 1

```
byte state = 1;
proctype A( ) {
byte tmp; (state==1) -> tmp = state; tmp = tmp+1;
state = tmp
}
proctype B( ) {
byte tmp; (state==1) -> tmp = state; tmp = tmp-1;
state = tmp
}
init {
run A(); run B()
}
```

Если какой-нибудь процесс завершится до того, как другой процесс выполнит проверку state==1, то “запоздавший” процесс будет навсегда блокирован. Если проверка условия выполнится процессами до того, как другой процесс завершится, то оба процесса завершатся, но значение переменной state будет непредсказуемым – она может принять любое значение: 0, 1 или 2

Параллельные процессы: гранулярность переходов

Process A::	Process B::
… k: x:=x+y …	… m: y:=y+x …

k_0: LD R_A, x k_1: ADD R_A, y k_2: ST R_A, x	m_0: LD R_B, y m_1: ADD R_B, x m_2: ST R_B, y

(x=1; y=2) – начальное состояние

1 вар. Если сложение *реализовано*, как одна атомарная операция процессора, то интерливинг даст два возможных результата: (x=3; y=5), (x=4; y=3)

2 вар. Если сложение *реализовано*, как три атомарных операции процессора, то интерливинг даст три возможных результата: (x=3; y=5), (x=4; y=3), (x=3; y=3)

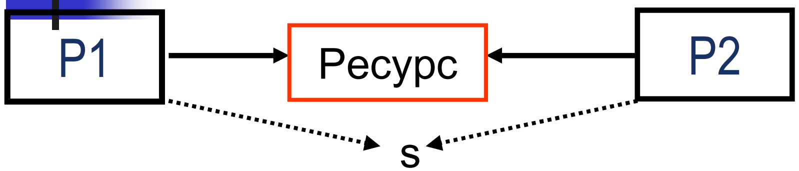
Пусть наличие состояния (x=3; y=3) нарушает некоторое проверяемое свойство R

Если в реализации - 1 вариант, а в модели используется 2 вариант, то мы ошибочно получим, что в системе свойство R не выполняется

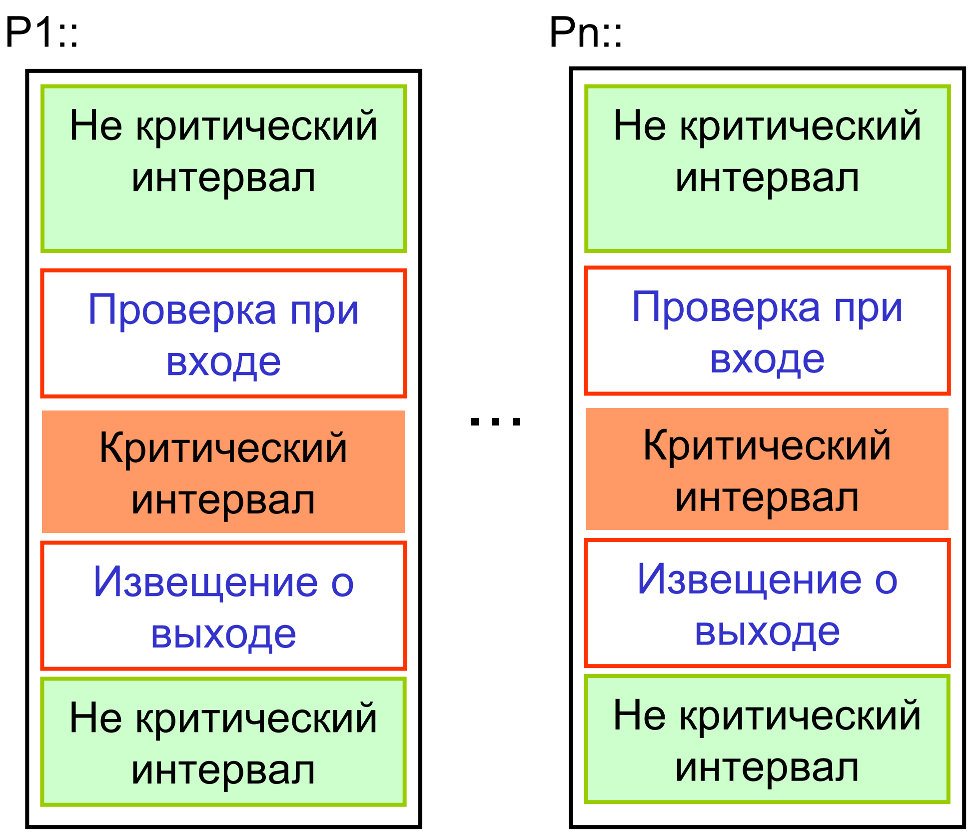
Если в реализации 2 вариант, а в модели используется 1 вариант, то мы ошибочно получим, что в системе свойство R выполняется

Преобразование в модель должно учитывать реализацию

Проблема взаимного исключения



s=1 – ресурс свободен



P1::

```
k0: noncritical1;
k1: if s>0 then s:=s-1else goto k1;
k2: critical1;
k3: s:=s+1;
k4: goto k0;
```

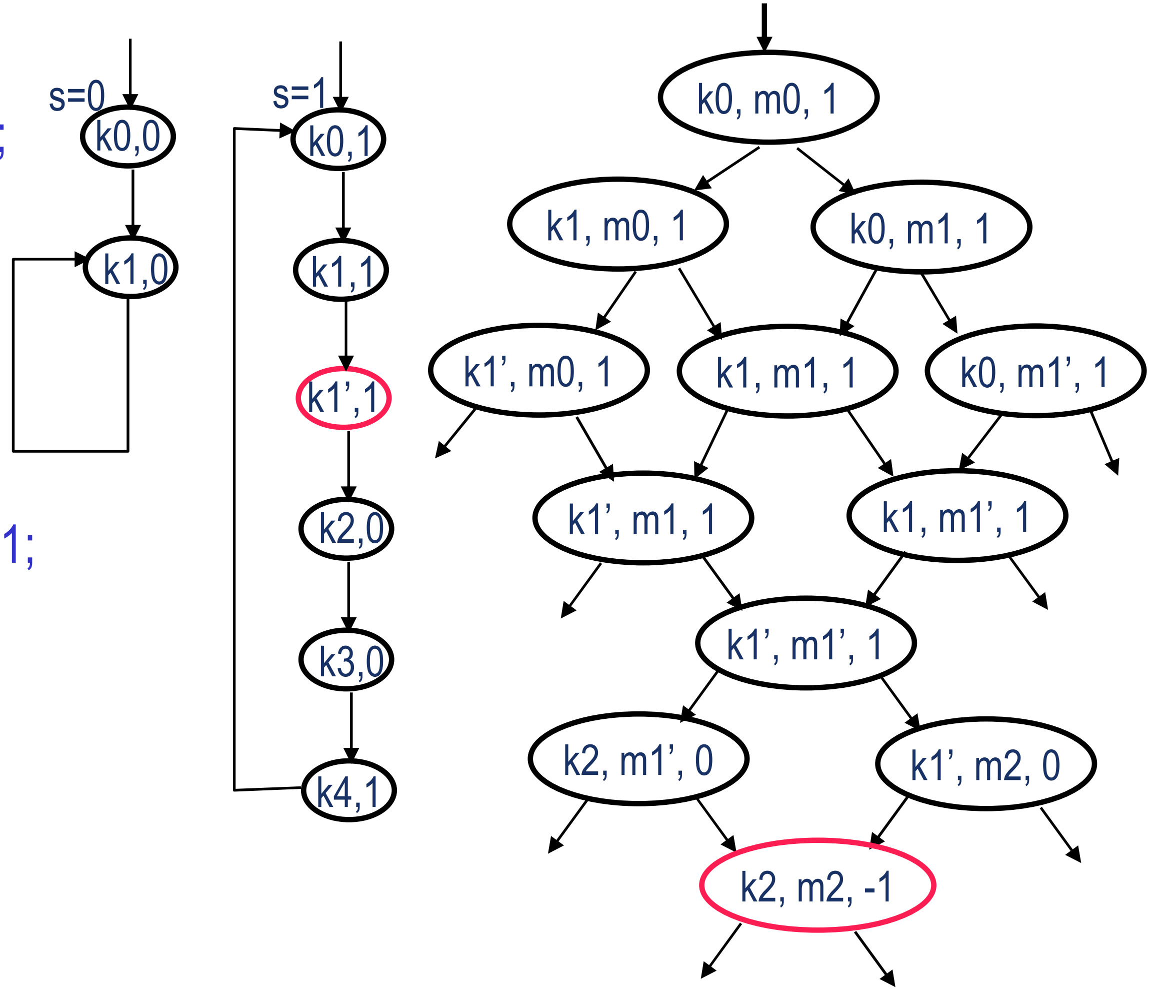
P2::

```
m0: noncritical2;
m1: if s>0 then s:=s-1else goto m1;
m2: critical2;
m3: s:=s+1
m4: goto m0;
```

Одно из решений проблемы взаимного исключения

P1::
k0: noncritical1;
k1: if s>0 then s:=s-1else goto k1;
k2: critical1;
k3: s:=s+1;
k4: goto k0;

P2::
m0: noncritical2;
m1: if s>0 then s:=s-1else goto m1;
m2: critical2;
m3: s:=s+1
m4: goto m0;



Решение некорректно: оператор *m: if s>0 then s:=s-1else goto m* состоит из двух операторов

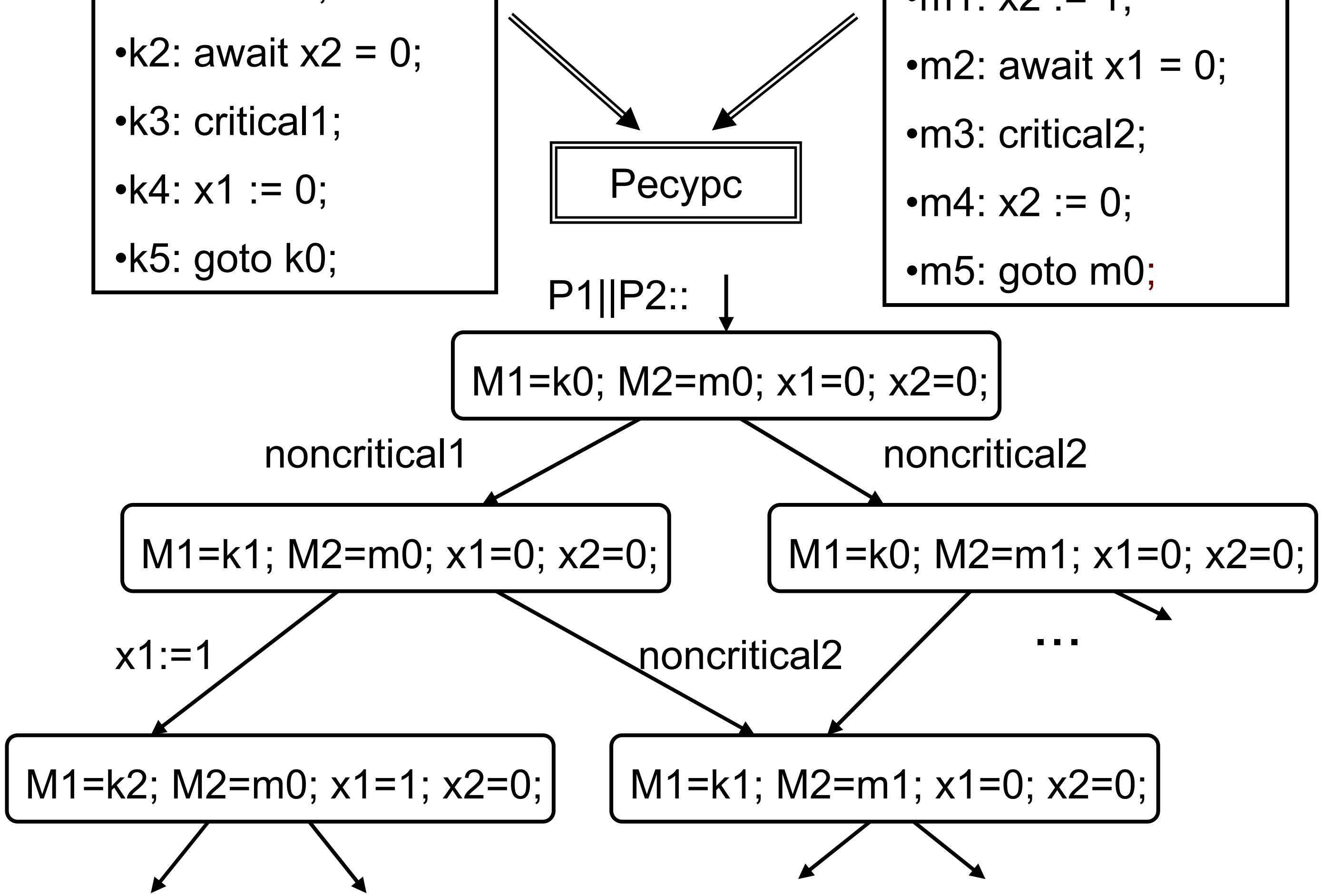
Другое решение проблемы взаимного исключения

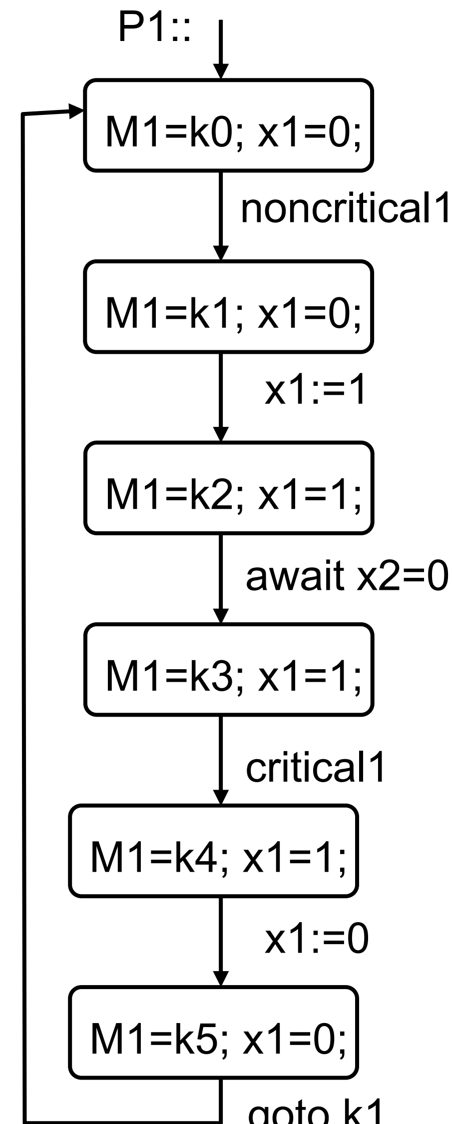
x1=0

P1::
- k0: noncritical1;
- k1: x1 := 1;
- k2: await x2 = 0;
- k3: critical1;
- k4: x1 := 0;
- k5: goto k0;

x2=0

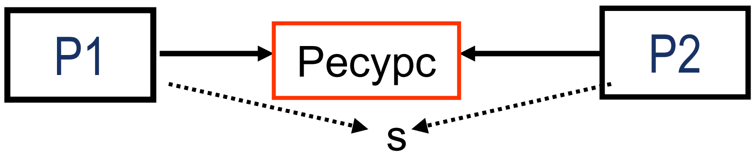
P2::
- m0: noncritical2;
- m1: x2 := 1;
- m2: await x1 = 0;
- m3: critical2;
- m4: x2 := 0;
- m5: goto m0;





Семафоры

Если оператор *m: if s>0 then s:=s-1* сделать ***мгновенным***, то решение будет корректным. Но это тоже доступ к общему ресурсу! Т.е. замкнутый круг!!



Нам не нужно мгновенное, нам нужно атомарное выполнение операции

Семафоры и операции над ними обеспечивают атомарное, *неделимое* ***(но не мгновенное)*** выполнение стандартной операции доступа к общему ресурсу – общей переменной

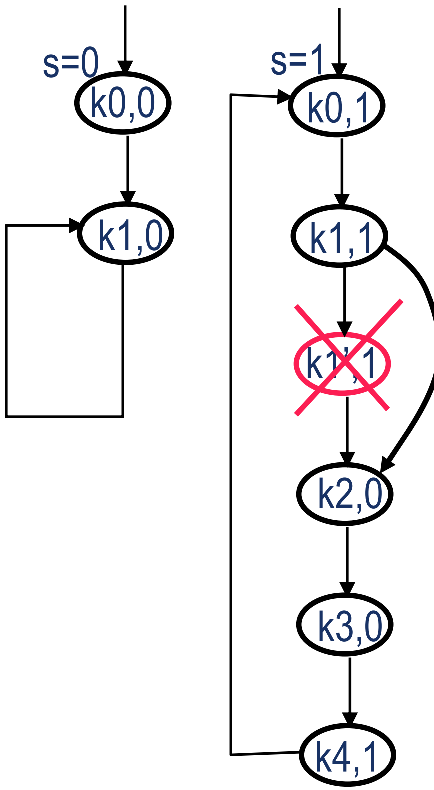
P(s) ≡ *m: if s>0 then s:=s-1*
выполняется атомарно

V(s) ≡ *s:=s+1* выполняется атомарно

P1::
k0: noncritical1;
k1: if s>0 then s:=s-1;
k2: critical1;
k3: s:=s+1;
k4: goto k0;

Взаимное исключение с помощью семафора

P1::
k0: noncritical1;
k1: P(s);
k2: critical1;
k3: V(s) ;
k4: goto k0;



Как в системах верификации?

- В реальных системах может быть любая степень грануляции – от микрокоманд до групп операторов, защищенных семафорами или другими средствами синхронизации
- Разработчик модели сам ответственен за то, будет ли модель адекватно отражать поведение системы (верификация мощна настолько, насколько адекватной является построенная для анализа модель)
- Неделимой единицей является обычно оператор входного языка системы верификации. Он целиком либо выполняется, либо нет.
- Атомарные последовательности операторов должны быть явно объявлены, например, специальные скобки $\langle \dots \rangle$
- В Promela: атомарными являются
  - любой отдельный оператор x = f(x, y), например m = (a>b $\rightarrow$ a : b)
  - *atomic* { ... } – группа операторов, заключенная в скобки с *atomic*

Выполнение параллельных процессов с атомарной цепочкой операторов

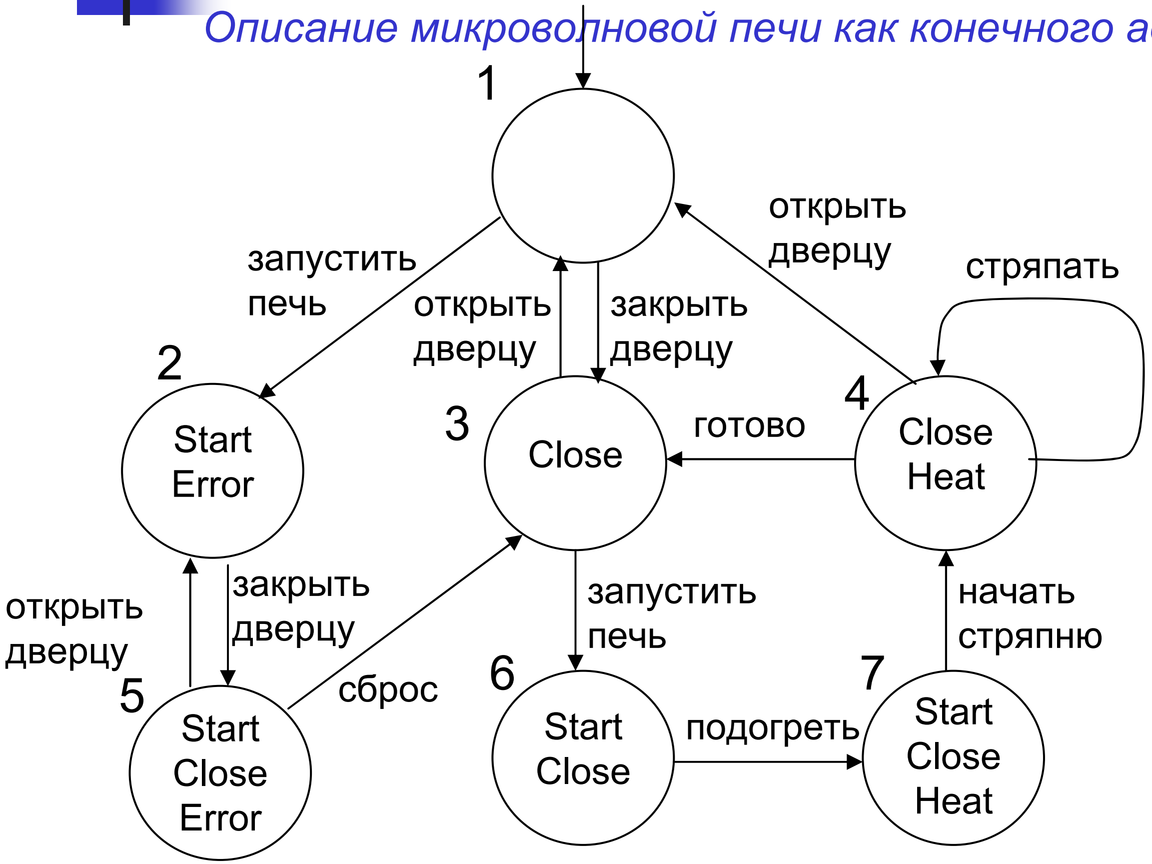
```
byte state = 1;

proctype A( ) {
atomic { (state==1) -> state = state+1 }
}

proctype B( ) {
atomic { (state==1) -> state = state-1 }
}

init {
 run A( ); run B( )
}
```

Как будет себя вести параллельная программа???

Значение переменной state станет равным 2 или 0, в зависимости от того, какой из процессов, A или B, первым выполнит свою атомарную операцию. Другой процесс будет при этом заблокирован навсегда

Системы, специфицированные в виде КА

Описание микроволновой печи как конечного автомата



Код генерируется по спецификации системы, представленной в виде системы переходов

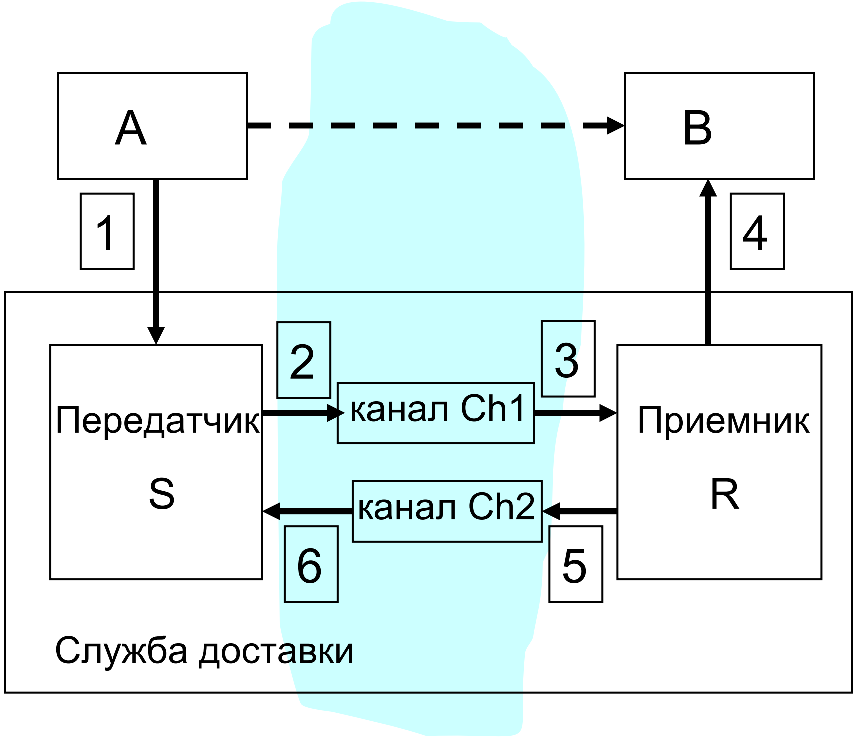
Структура Крипке получается отбрасыванием действий на переходах – неважно, какие последовательности входов привели к ошибке

Можно проверить систему относительно любой спецификации, основанной на атомарных предикатах, например: $\mathbf{AG}(\neg \mathit{Close} \Rightarrow \neg \mathit{Heat})$

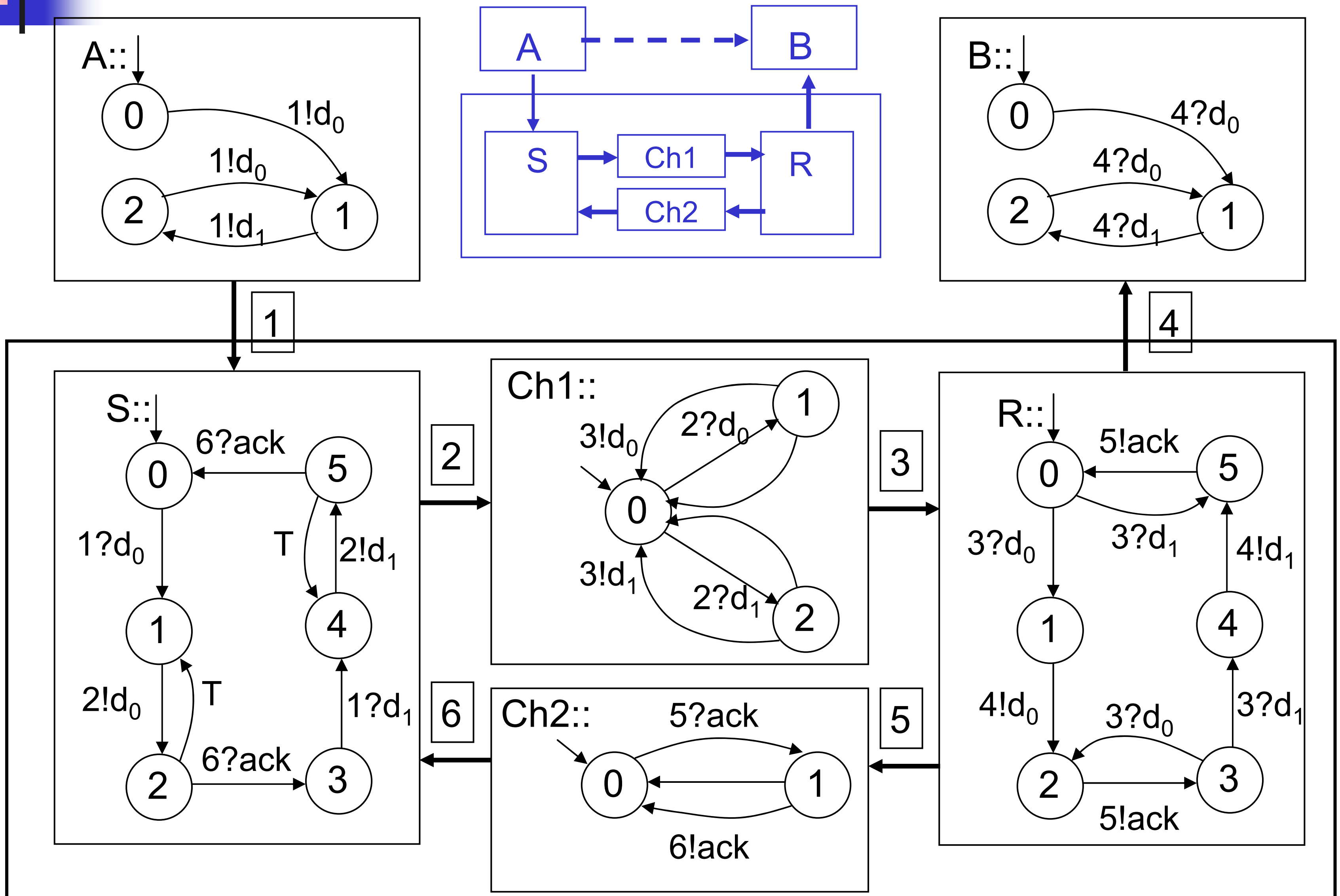
"*В любом состоянии всегда при открытой дверце нагревание не происходит*"

Параллельные системы, специфицированные в виде взаимодействующих КА

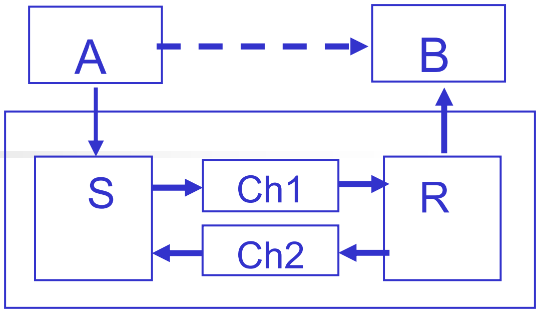
Архитектура протокола PAR

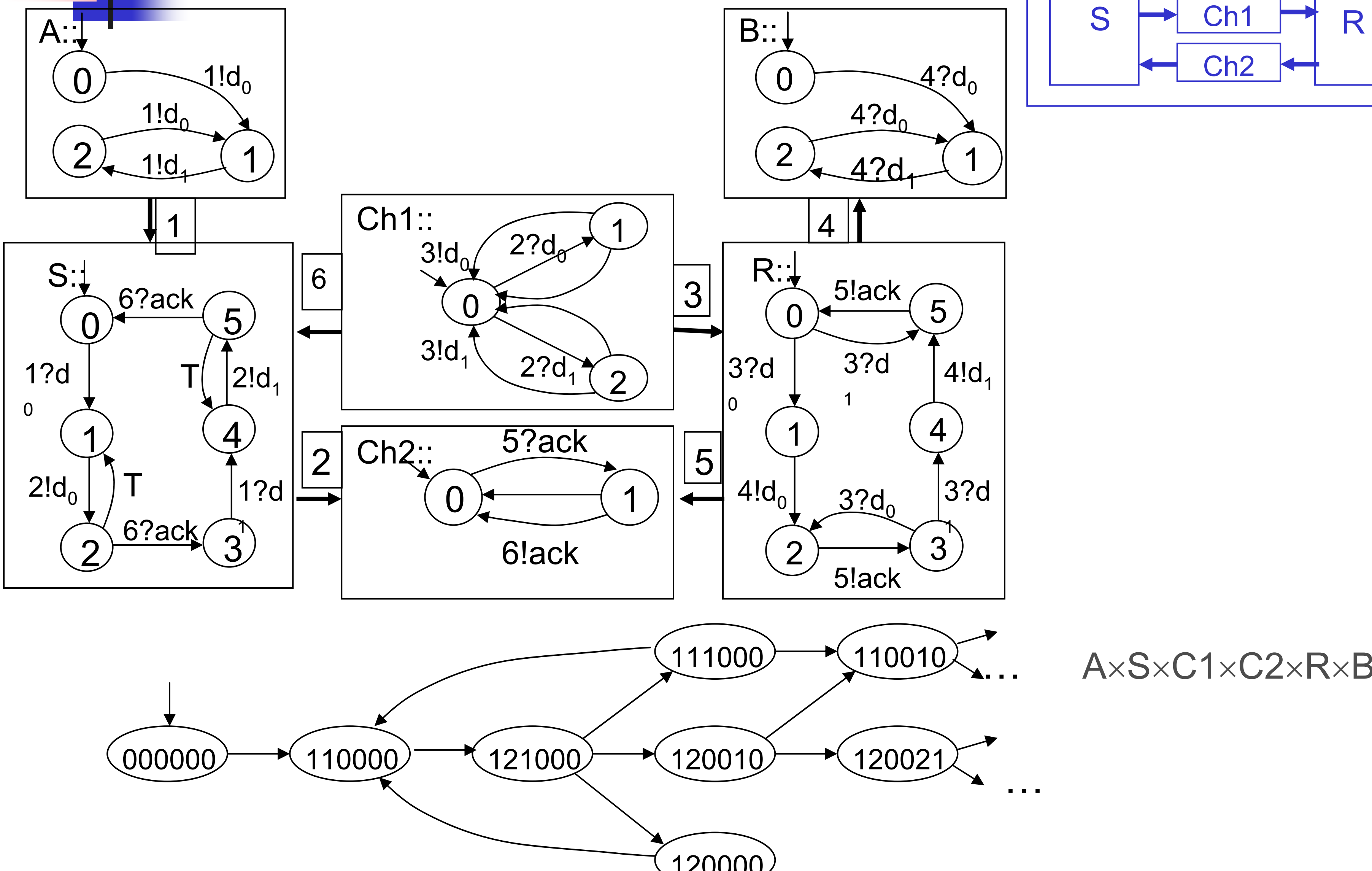


Автоматы, моделирующие протокол PAR

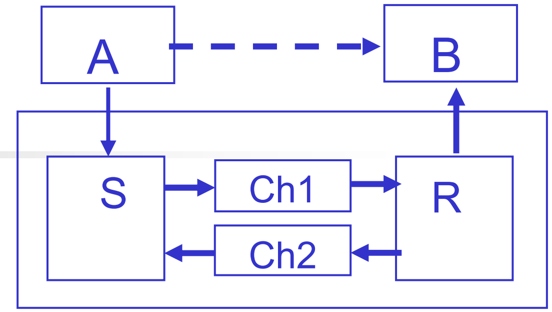


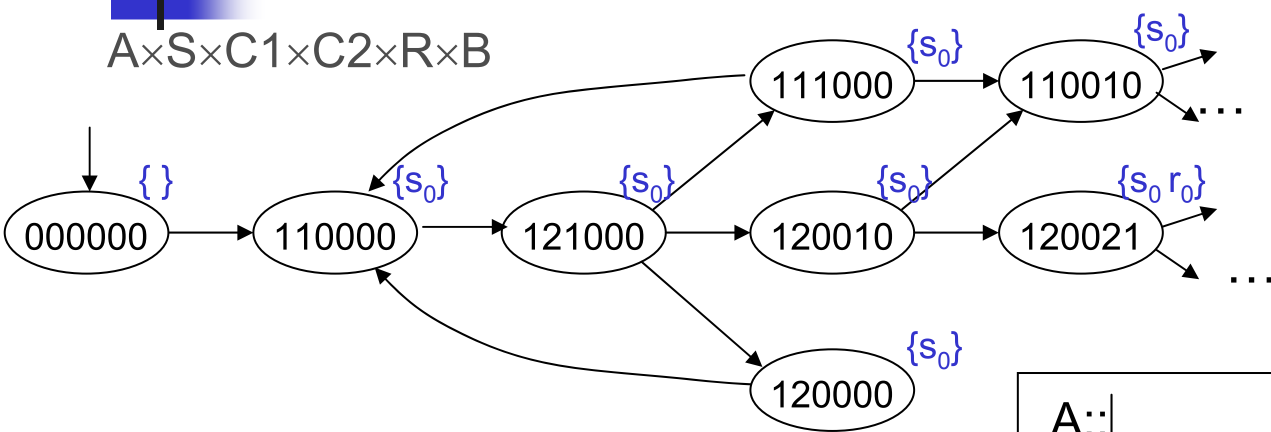
Система переходов протокола PAR





Структура Крипке протокола PAR

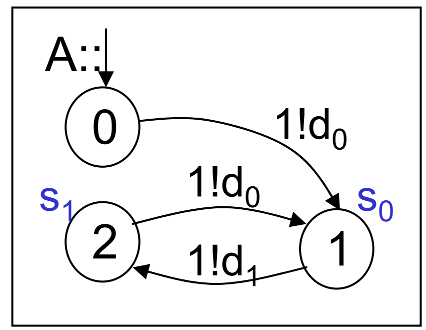


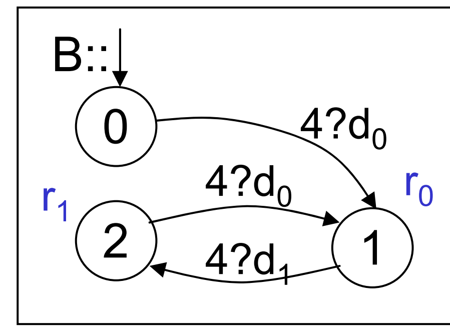


Атомарные предикаты:

s_k: *А послал сообщение, занумерованное k*

r_k: *В принял сообщение, занумерованное k*





s_0 *истинен в состоянии 1 автомата A*

s_1 *истинен в состоянии 2 автомата A*

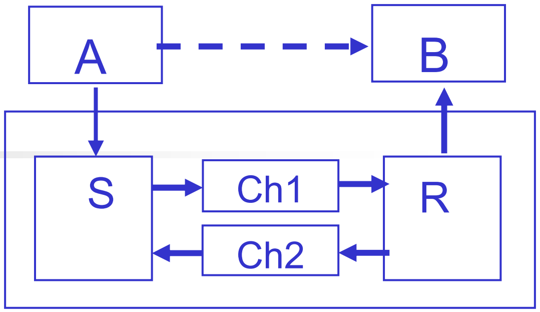
r_0 *истинен в состоянии 1 автомата B*

r_1 *истинен в состоянии 2 автомата B*

000000{ }→110000 $\{s_0\}$→121000 $\{s_0\}$→120010 $\{s_0\}$ →110021 $\{s_0,r_0\}$→...

{ } → $\{s_0\}$ →$\{s_0\}$ → $\{s_0\}$ → $\{s_0,r_0\}$ →...

Спецификация протокола PAR



Spec1: $\mathbf{G}((s_0 \Rightarrow \mathbf{F} s_1) \wedge (s_1 \Rightarrow \mathbf{F} s_0))$

"для любого состояния вычислений верно, что если А послал сообщение, то когда-нибудь в будущем он сможет послать следующее"

Spec 2: $\mathbf{G}(s_0 \Rightarrow (\neg s_1 \mathbf{U} r_0)) \wedge \mathbf{G}(r_0 \Rightarrow (\neg r_1 \mathbf{U} s_1)) \wedge \mathbf{G}(s_1 \Rightarrow (\neg s_0 \mathbf{U} r_1)) \wedge \mathbf{G}(r_1 \Rightarrow (\neg r_0 \mathbf{U} s_0))$

"если А послал сообщение, то он не посылает следующее, пока В не примет его, и если В принял сообщение, то он не принимает никакого другого сообщения, пока А не пошлет следующее"

Ошибочная траектория протокола PAR

000000{ } $\rightarrow$ А посылает сообщение d_0 передатчику S $\rightarrow$
110000{s_0} $\rightarrow$ S посылает d_0 в канал Ch_1 $\rightarrow$
121000{s_0} $\rightarrow$ Ch_1 доставляет d_0 приемнику R $\rightarrow$
120010{s_0} $\rightarrow$ "*нетерпеливый передатчик*" не дождался подтверждения $\rightarrow$
110010{s_0} $\rightarrow$ приемник R доставляет d_0 пользователю B $\rightarrow$
110021{s_0,r_0} $\rightarrow$ передатчик повторно посылает d_0 в канал Ch_1 $\rightarrow$
121021{s_0,r_0} $\rightarrow$ приемник R посылает в канал Ch_2 подтверждение приема d_0 $\rightarrow$
121131{s_0,r_0} $\rightarrow$ передатчик S получает из Ch_2 подтверждение приема d_0 $\rightarrow$
131031{s_0,r_0} $\rightarrow$ канал Ch_1 повторно доставляет d_0 приемнику R $\rightarrow$
130021{s_0,r_0} $\rightarrow$ передатчик S принимает от А новые данные d_1 $\rightarrow$
240021{s_1,r_0} $\rightarrow$ S передает эти данные в канал Ch_1 $\rightarrow$
252021{s_1,r_0} $\rightarrow$ R передает в Ch_2 подтверждение получения d_0 $\rightarrow$
252131{s_1,r_0} $\rightarrow$ канал Ch_1 теряет сообщение d_1 $\rightarrow$
250131{s_1,r_0} $\rightarrow$ S получает подтверждение от Ch_2 (ошибочно считает, что d_1 дошло) $\rightarrow$
200031{s_1,r_0} $\rightarrow$ S принимает от А новые данные d_0 $\rightarrow$
110031{s_0,r_0} $\rightarrow$ S посылает d_0 в канал Ch_1 ***здесь нарушение*** $\boldsymbol{G(s_1 \Rightarrow (\neg\, s_1 U r_1))}$ $\rightarrow$
121031{s_0,r_0} $\rightarrow$ Ch_1 доставляет d_0 приемнику R $\rightarrow$ ...

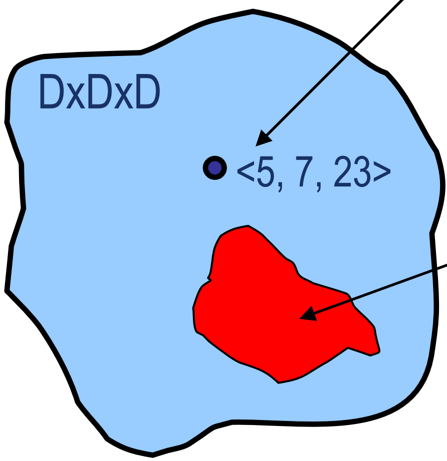
Некорректность протокола устанавливается обнаружением неправильной последовательности событий на вычислениях. События устанавливаются с помощью атомарных предикатов, связанных с состояниями

Как строить систему переходов из программ

V={v1, …, vn) – множество переменных, принимающих значения в D, $v_i \in$ D

D - конечно

s: V→D



Состояние s={v_1=5; v_2=3; v_3=23} $s(v_1)=5; s(v_2)=3; s(v_3)=23$

Но это можно рассматривать как предикат! $(v_1=5)\&(v_2=3)\&(v_3=23)$

Итак, предикат задает состояние системы

Множество состояний – предикат, например: $S=(v_1>v_2)\ \&\ \neg v_3=6$

Вывод: Предикатом – формулой над множеством переменных из V - можно задать как одно состояние, так и множество состояний

Множество состояний, определяемых формулой (предикатом) S(V), это множество таких значений переменных из V, на которых предикат *S* истинен

Переход – это упорядоченная пара состояний $\langle s,s' \rangle \in \Omega^2$

Множество переходов можно тоже задать предикатом $\Re$(V, V’)

Множество переходов - это множество пар $\langle s,s' \rangle$ из $V \times V'$, таких, что $\Re$(V, V’)=True

Логика и структура Крипке

Пусть задана реактивная система с множеством переменных V над областью D

Структура Крипке – К=(S, S_0, R, L)

Множество состояний S – предикат Ω, т.е. множество всех $V \to D$

Множество начальных состояний S_0– предикат Ω_0

Множество переходов R – множество таких пар <s,s'>, таких, что $\Re(V, V')=$*True*

Атомарные утверждения AP – отношение на множестве значений переменных, например x>2*y

Утверждение x>2*y истинно в состоянии s, если s(x) > 2* s(y)

Функция пометок $L:S \to 2^{AP}$
Для каждого состояния s – это множество атомарных утверждений, истинных в s

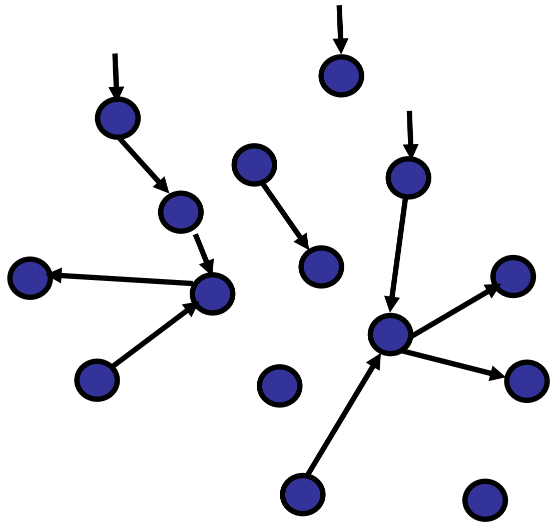
Моделирование программ

Состояние программы - мгновенный снимок <V,PC>
• множества значений всех переменных из V
• значения счетчиков команд pc для каждого из параллельных процессов

Фактически, проблема построения структуры Крипке по программе – это разработка компилятора, переводящего программу в формулы логики I порядка: в формулу $S_0(V, PC)$ и формулу $\mathfrak{R}(V, PC, V', PC')$

$$S_0(V, PC) \equiv S^0_1 \vee S^0_2 \vee \ldots \vee S^0_m$$

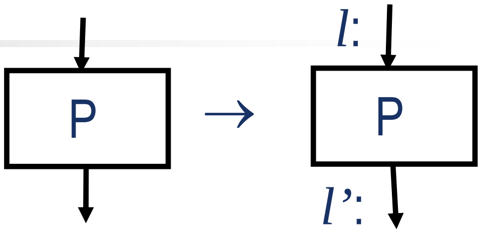
для m параллельных процессов $P_1, P_2, \ldots, P_m$



$$\mathfrak{R}(V, PC, V', PC') \equiv \mathfrak{R}_1 \vee \mathfrak{R}_2 \vee \ldots \vee \mathfrak{R}_n$$

Каждый дизъюнкт $\mathfrak{R}_k(V_k, PC_k, V'_k, PC'_k)$ определяет один из возможных переходов системы из состояния <V_k, PC_k> в состояние <V'_k, PC'_k> (действие одного процесса)

Построение отношения перехода на множестве состояний



1 шаг: Разметка программы – компиляция в идентичный помеченный текст

---

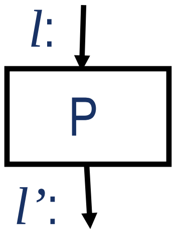
2 шаг: Построение формулы, определяющей множество начальных состояний программы.

S_0: Pre(V) & pc = l_0

Pre(V) – предусловие значений переменных

---

3 шаг: Построение формулы переходов – каждый оператор программы определяет возможные переходы между ссостояниями. Компиляция выполняется на основе атрибутной грамматики



$\mathfrak{R}_k$:
$\&_{v \in V}\, v_i = d_i \,\&\, pc = l$
$\&_{v' \in V'}\, v'_i = d'_i \,\&\, pc' = l'$

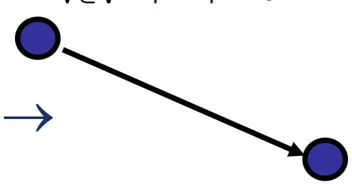
---

4 шаг: Структура Крипке определяется формулами

S_0(V,pc) и
$\mathfrak{R}$(V, pc, V', pc')

$L(S_1) = \&_{v \in V}\, v_i = d_i \,\&\, pc = l$

$\mathfrak{R}_k$ →

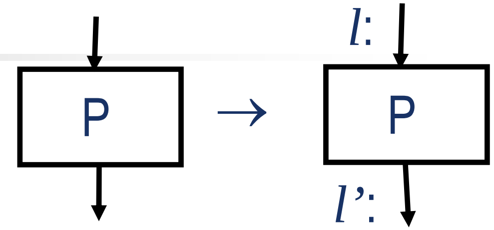


$L(S_2) = \&_{v' \in V'}\, v'_i = d'_i \,\&\, pc' = l'$

Последовательные программы (1)

Язык:

P ::= v ← e| skip | P; P | if b then P else P fi | while b do P od



1 шаг: Разметка программы – компиляция в идентичный помеченный текст

Если P – простой оператор (v ← e | skip ...) – то $\mathbb{P} = l'$: P

P = P_1; P_2 $\mathbb{P}$ = l_0 : P_1; l_1: P_2

P= if b then P_1 else P_2 fi $\mathbb{P}$ = l_0: if b then l_1 : P_1 else l_2: P_2 fi

P = while b do P_1 od $\mathbb{P}$ = l_0: while b do l_1: P_1 od

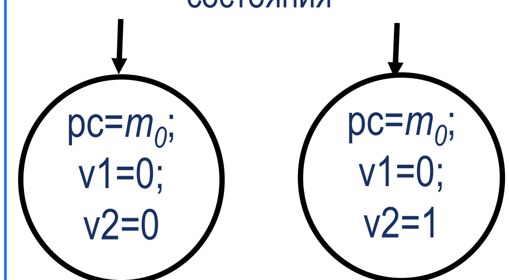
Необходимо указывать только метку начала операторов

Метка конца оператора – это метка следующего оператора

Пример V={v1,v2}

Pre={v1=0; v2 ∈{0,1}}

Дает два начальных состояния



2 шаг: Построение формулы начальных состояний программы

$$\boxed{S_0(V, pc) \equiv Pre(V) \,\&\, pc = m_0}$$

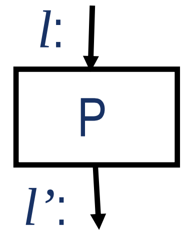
Последовательные программы (2)

3 шаг: Построение формулы переходов –

компиляция на основе атрибутной грамматики

$C(l, v \leftarrow e, l') \equiv pc=l \ \& \ pc' = l' \ \& \ v'=e \ \& \ same\,(V \setminus \{v\})$

$same(V) \equiv \&_{v \in V} (v'=v)$

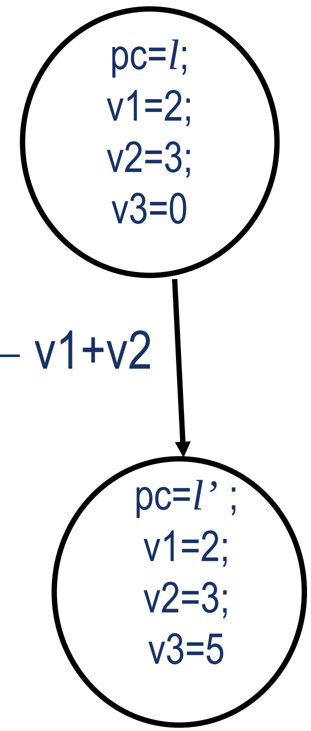


$\rightarrow$ $\mathfrak{R}_k$:
$\&_{v \in V} \, v_i=d_i \ \& \ pc=l$
$\&_{v' \in V'} \, v'_i=d'_i \ \& \ pc'=l'$

$C(l, skip, l') \equiv pc=l \ \& \ pc' = l' \ \& \ same\,(V)$

$C(l, \text{if } b \text{ then } l_1{:}\ P_1 \text{ else } l_2{:}\ P_2 \text{ fi}, \ l') \equiv$

$$pc=l \ \& \ pc'= l_1 \ \& \ b \ \& \ same\,(V)$$
$$\vee \ pc=l \ \& \ pc'= l_2 \ \& \ \neg b \ \& \ same\,(V)$$
$$\vee \ C\,(l_1, P_1, \ l')$$
$$\vee \ C\,(l_2, P_2, \ l')$$



P: $v3 \leftarrow v1+v2$

$C(l, \text{while } b \text{ do } l_1{:}\ P_1 \text{ od}, \ l') \equiv$

$$pc=l \ \& \ pc'= l_1 \ \& \ b \ \& \ same\,(V)$$
$$\vee \ pc=l \ \& \ pc'= l' \ \& \ \neg b \ \& \ same\,(V)$$
$$\vee \ C(l_1, P1, \ l)$$

Параллельные программы

Параллельная программа образована множеством процессов (последовательных программ), которые исполняются параллельно.

Рассматриваем асинхронные программы (в точности один процесс может совершить переход в каждый момент времени), взаимодействующие через
- разделяемые переменные
- передачу сообщений
- рандеву (хендшейк)

pc_i – счетчик команд процесса P_i
V_i – множество переменных, которые может изменить процесс P_i
$V_i \cap V_k$ – множество переменных разделяемых процессами P_i и P_k

Передача сообщений и рандеву реализуются в программах процессов с помощью операторов wait(b), lock(v), unlock(v) над разделяемыми переменными

Трансляция параллельной программы в структуру Крипке

К языку последовательных программ добавляется оператор параллельного запуска и операторы синхронизации

P::= cobegin P1|| P2 || ... || Pn coend

1 шаг: Разметка. Вход и выход всей || программы помечаем m и m'

$P^L ::= m\text{: cobegin } l_1\text{: P1 } l_1' \| \; l_2\text{: P2 } l_2' \; \| \dots \| \; l_n\text{: Pn } l_n' \text{ coend } m'$

2 шаг: Начальное состояние

$$\boxed{S0(V, PC) \equiv pre(V) \;\&\; pc = m \;\&\; \&_{i=1:n} pc_i = \perp}$$

3 шаг: Построение формулы переходов

$C(m, \text{cobegin } l_1\text{: } P_1 \; l_1' \| \; l_2\text{: } P_2 \; l_2' \; \| \dots \| \; l_n\text{: } P_n \; l_n' \text{ coend}, m') \equiv$

$pc = m \;\&\; pc'_1 = l_1 \;\&\; \dots \;\&\; pc_n' = l_n \;\&\; pc' = \perp$ // инициализация

$\vee \, [\, \vee_{i=1:n} (C(l_i, P_i, l_i') \;\&\; same(V \setminus V_i) \;\&\; same(PC \setminus \{pc_i\})\,)\,]$ // вычисления || процессов

$\vee \; pc = \perp \;\&\; pc_1 = l_1' \;\&\; \dots \;\&\; pc_n = l_n' \;\&\; pc' = m' \;\&\; \&_{i=1:n}(pc_i' = \perp)$ // завершение всех

Трансляция параллельной программы в структуру Крипке (2)

3 шаг: Построение формулы для переходов для разделяемых переменных:

Оператор wait(b): периодическая проверка b до тех пор пока не станет true

$C(\, l, \text{wait (b)}, l\,'\,) \equiv$

$\quad pc_i = l \ \&\ pc_i' = l \ \&\ \neg b \ \&\ same(V_i)$ // b не выполняется - ждем

$\quad \vee\ pc_i = l \ \&\ pc_i' = l\,' \&\ \ b \ \&\ same(V_i)$ // b выполняется - переходим

Оператор lock(v): аналогично wait(v=0), но как только v=0 увеличивает v на 1

$C(\, l, \text{lock (v)}, l\,'\,) \equiv$

$\quad pc_i = l \ \&\ pc_i' = l \ \&\ v=1 \ \&\ same(V_i)$

$\quad \vee\ pc_i = l \ \&\ pc_i' = l\,' \&\ v=0 \ \&\ v'=1 \ \&\ same(V_i \backslash \{v\})$

Оператор unlock(v): присваивает 0 переменной v

$C(\, l, \text{unlock (v)}, l\,'\,) \equiv$

$\quad pc_i = l \ \&\ pc_i' = l\,' \&\ v'=0 \ \&\ same(V_i \backslash \{v\})$

Пример трансляции программы в структуру Крипке

P::= *m*: cobegin P_0 || P_1 coend *m’*

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’:

pc – программный счетчик P, принимает три значения: $\{m, m', \bot\}$

$\bot$ - процесс P неактивен (управление в P_0 и в P_1)

pc_i – программный счетчик P_i, принимает значения: $\{l_i, l_i', NC_i, Cr_i, \bot\}$ ($\bot$ - когда управление в P)

turn – разделяемая переменная: $V=V_0=V_1=\{ turn \}$

$PC=\{pc, pc_0, pc_1\}$ (обозначим *turn* буквой t)

Начальное состояние S0(V,PC) (*t* не определена!)

$$pre(V) \& pc= m \ \&_i \ pc_i=\bot \equiv (t=0 \vee t=1) \& \ pc= m \ \&_i \ pc_i =\bot \equiv$$
$$t=0 \ \& \ pc = m \ \& \ pc_0 = \bot \ \& \ pc_1 =\bot ,$$
$$\vee \ t=1 \ \& \ pc = m \ \& \ pc_0 = \bot \ \& \ pc_1 =\bot$$

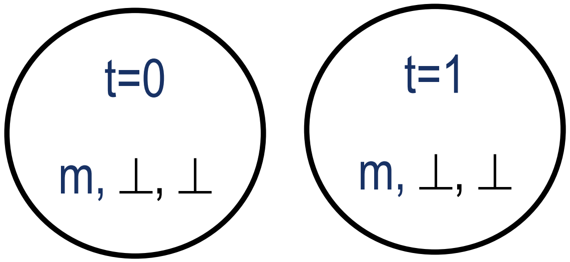
Множество переходов:

$[\ pc = m \ \& \ pc_0' = l_0 \ \& \ pc_1' = l_1 \ \& \ pc' = \bot\ ]$	// инициализация
$\vee\ [\ pc_0= l_0' \ \& \ pc_1 = l_1' \ \& \ pc' = m' \ \& \ pc_0' = \bot \ \& \ pc_1' = \bot]$	// завершение всех
$\vee\ [\ C(l_0, P_0, l_0') \ \& \ same(pc, pc_0)]$	// работа P_0
$\vee\ [\ C(l_1, P_1, l_1') \ \& \ same(pc, pc_1)]$	// работа P_1

Пример трансляции программы в структуру Крипке (2)

P::= m: cobegin P_0 || P_1 coend m'

$s = \langle t, pc, pc_0, pc_1 \rangle$



Множество начальных состояний

P_i:: l_i: while *True* do
    NC_i: wait ($turn$ =i);
    CR_i: $turn$:= i ⊕1;
  end while;
l_i':

$C(l_i, P_i, l_i') =$

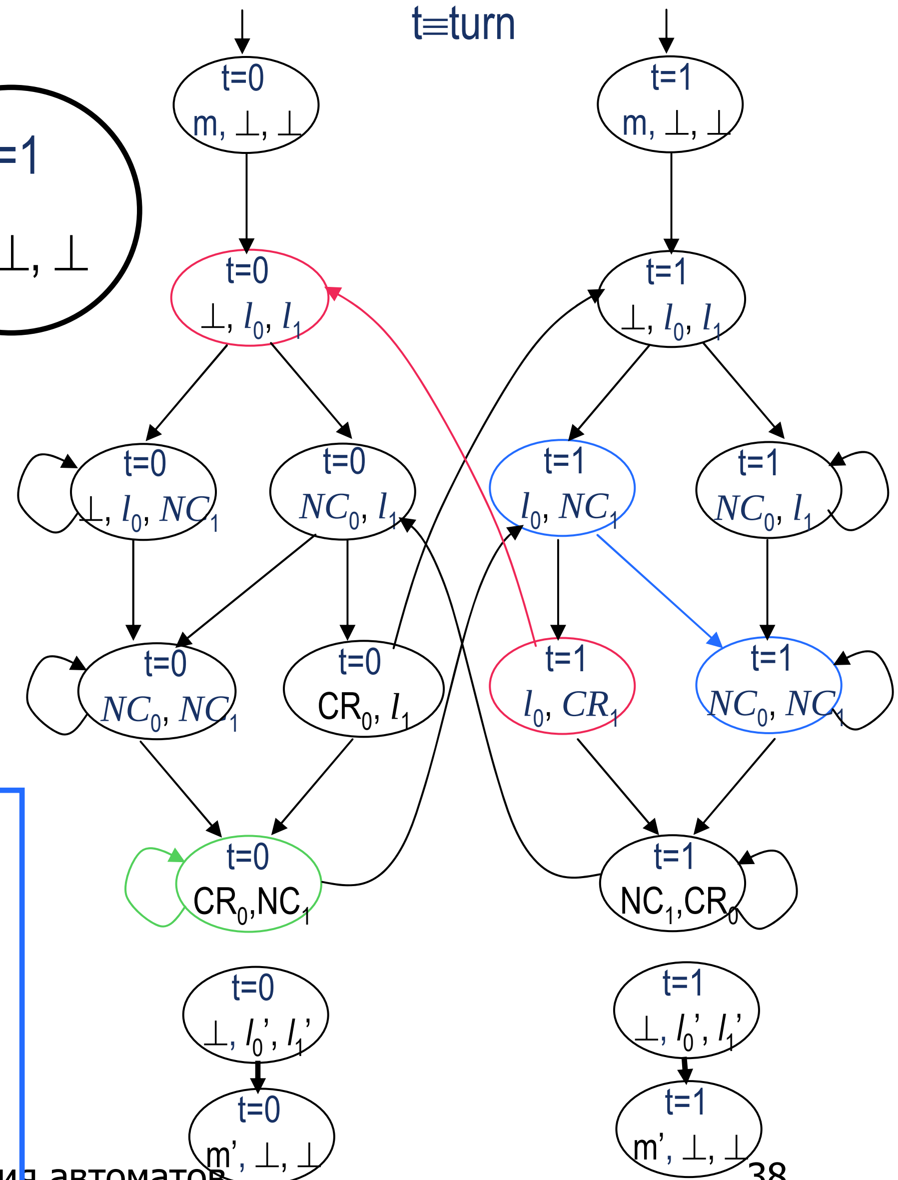
$[pc_i = l_i \ \& \ pc_i' = NC_i \ \& \ True \ \& \ same(turn)]$

$\vee \ [pc_i = NC_i \ \& \ pc_i' = CR_i \ \& \ turn = i \ \& \ same(turn)]$

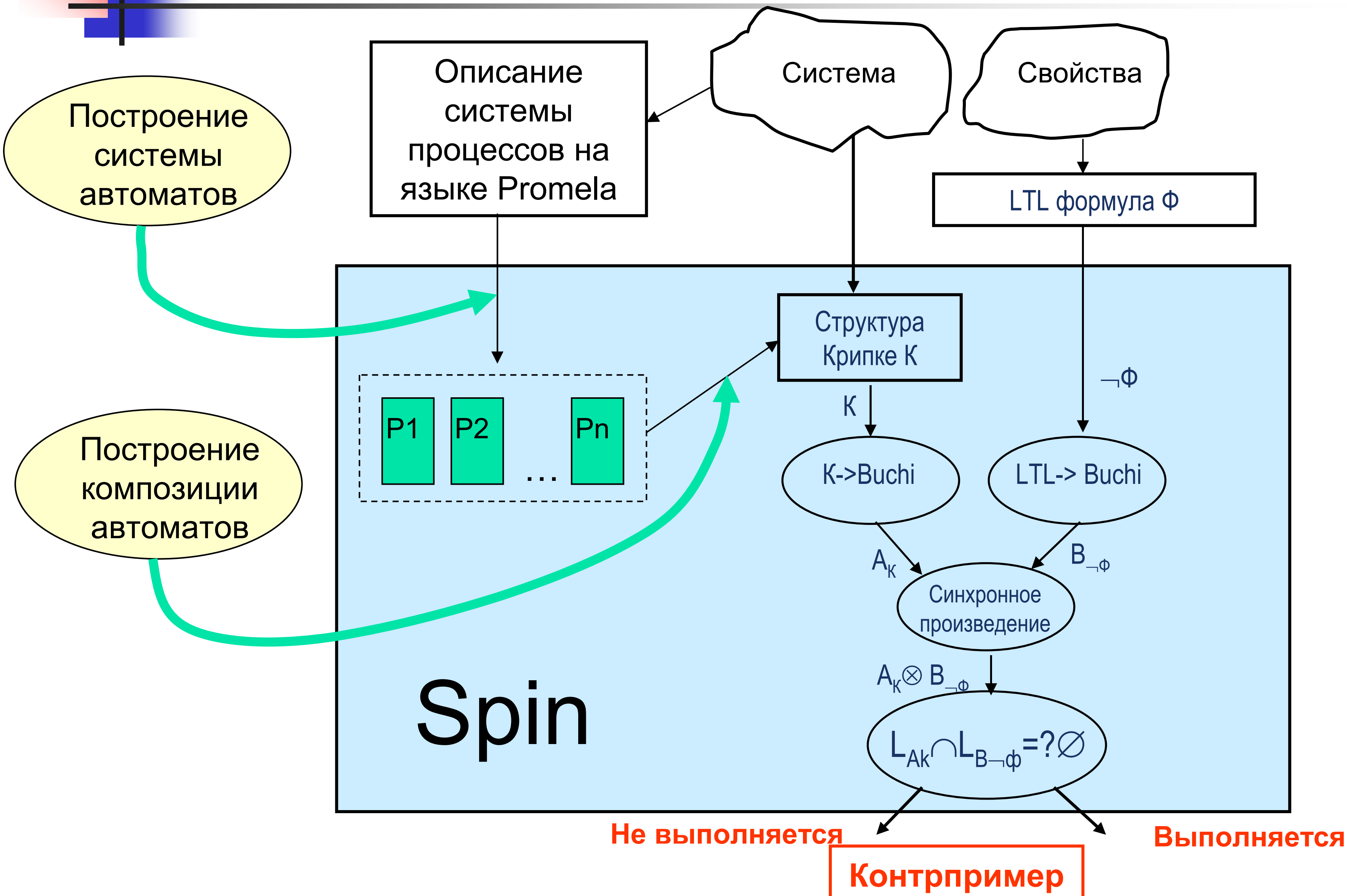
$\vee \ [pc_i = CR_i \ \& \ pc_i' = l_1 \ \& \ turn' = i \oplus 1]$

$\vee \ [pc_i = NC_i \ \& \ pc_i' = NC_i \ \& \ turn \neq i \ \& \ same(turn)]$

$\vee \ [pc_i = l_i \ \& \ pc_i' = l_i' \ \& \ False \ \& \ same(turn)]$



Система верификации Spin



Заключение

- В абстрактную модель (структуру Крипке) перевод из реальной системы часто неформален (и неоднозначен)
- Структура Крипке – конечная система переходов, поэтому в структуре Крипке многие свойства реальной системы могут не сохраняться. Значит, не все реальные свойства могут быть проверены
- Для систем с конечным числом состояний формальный перевод спецификации системы в структуру Крипке возможен
- Понятие атомарности операций в модели должно соответствовать ограничениям взаимной видимости состояний процессов в реальной параллельной системе
- Язык логики (булевых формул) является естественным для представления как состояний, так и переходов структуры Крипке
- Разработаны алгоритмы компиляции из ограниченных языков высокого уровня в булевы формулы, представляющие структуру Крипке

Задача. Проверить корректность алгоритма взаимного исключения Петерсона

P1::

```
loop forever
noncritical1;
⟨b1:=true; x:=2⟩;
wait until (x=1 ∨ ¬b2)
critical1;
b1 := false;
end loop;
```

P2::

```
loop forever
noncritical 2;
⟨b2:=true; x:=1⟩;
wait until (x=2 ∨ ¬b1)
critical2;
b2 := false;
end loop;
```

Спасибо за внимание