Верификация моделей программ

ЛЕКТОРЫ:

Владимир Анатольевич Захаров
Владислав Васильевич Подымов

zakh@cs.msu.su

Лекция 3.

Моделирование программ и схем

Верификация моделей программ

Основные принципы моделирования

Модели Крипке

Представления первого порядка

Степень детализации представления

Трансляция программ в модели Крипке

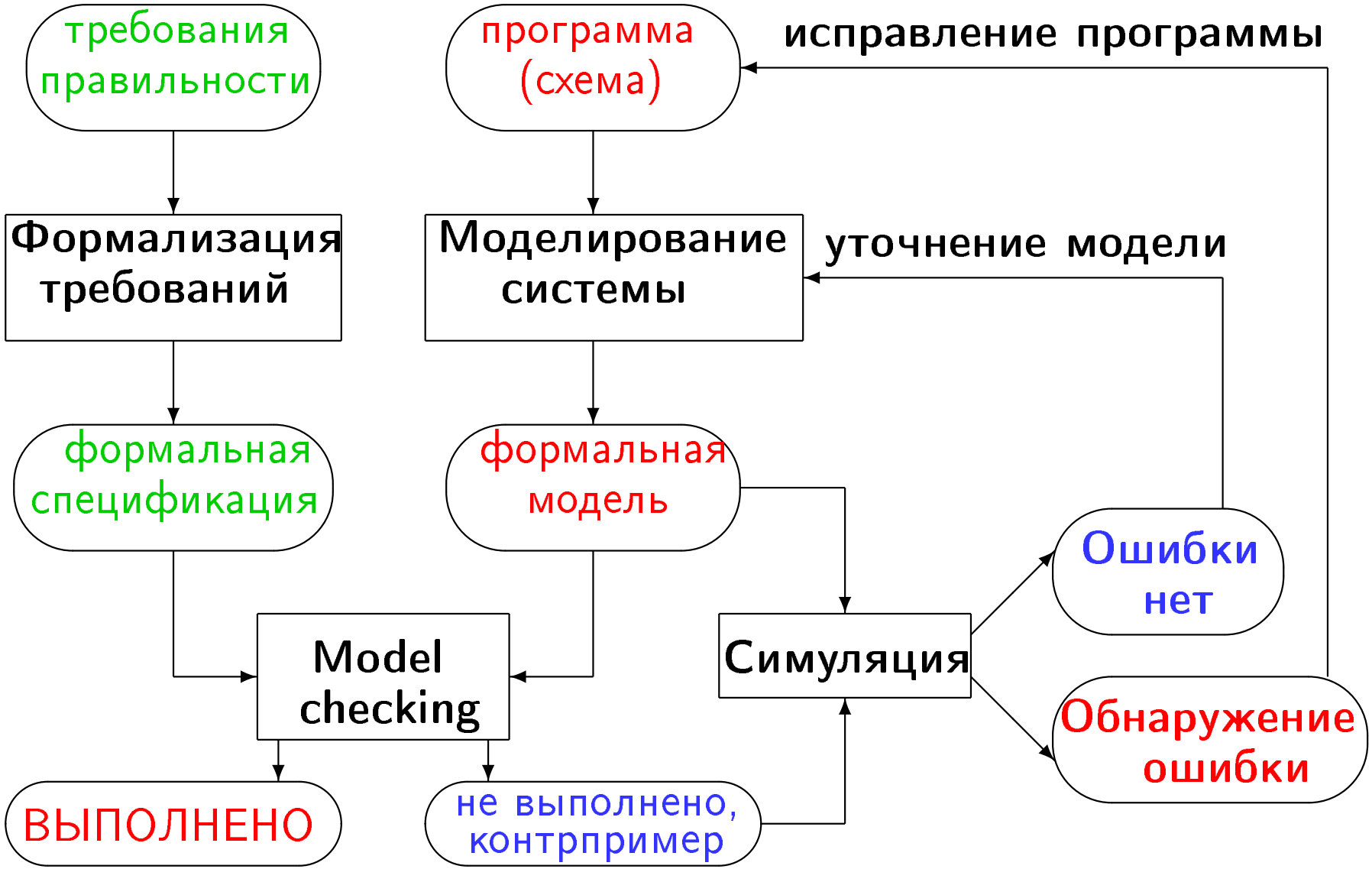
Верификация моделей программ

Метод верификации моделей (или model checking) заключается в исчерпывающем обходе пространства состояний системы. При наличии достаточных ресурсов эта процедура всегда завершается и может быть реализована достаточно эффективным алгоритмом.

В некоторых случаях системы с бесконечным числом состояний могут быть проверены этим методом в сочетании с разнообразными правилами абстракции и индукции.

Поскольку метод проверки на модели может применяться чисто автоматически, он предпочтительнее дедуктивного анализа в тех случаях, когда может быть использован.

Верификация моделей программ



Верификация моделей программ

Результат верификации
модели программ
настолько достоверен,
насколько хорошо построена
модель программы.

Плохая модель —
бесполезные результаты.

Принципы моделирования систем

Первым шагом при проверке корректности системы является обсуждение и спецификация свойств, которыми она должна обладать.

Как только станет известно, какие свойства являются важными, следующим шагом будет построение формальной модели системы.

Чтобы модель была пригодна для верификации, в ней должны проявляться те свойства, анализ которых необходим для установления ее корректности.

В то же время, она должна быть свободна от частных особенностей, не влияющих на проверяемые свойства, но усложняющих верификацию.

Принципы моделирования систем

Например, при моделировании цифровых схем целесообразно рассуждать в терминах логических ячеек и булевых значений, а не в терминах уровней напряжения.

А при исследовании коммуникационных протоколов желательно сосредоточить внимание на обмене сообщениями, игнорируя при этом содержание самих сообщений.

Принципы моделирования систем

При проектировании микроэлектронной аппаратуры имеют дело дело с реагирующими системами , поведение которых проявляется во взаимодействии с окружающей средой.

Первой характерной чертой реагирующей системы является ее состояние — «моментальный снимок» системы, в котором запечатлены значения переменных в заданный момент времени.

Необходимо также знать, как изменяется состояние системы в результате выполнения тех или иных действий. Это изменение можно описать, указав состояние системы до выполнения действия и ее состояние после выполнения действия. Такая пара состояний определяет переход системы.

Вычисление реагирующей системы определяется в терминах переходов системы. Вычисление – это бесконечная последовательность состояний, каждое из которых получено из предыдущего посредством некоторого перехода.

Принципы моделирования систем

Для формализации наших представлений о поведении реагирующих систем мы воспользуемся разновидностью размеченного графа переходов, которая называется моделью Крипке .

Модель Крипке состоит из множества состояний, множества переходов между состояниями и функции, которая помечает каждое состояние набором свойств, истинных в этом состоянии. Пути в модели Крипке соответствуют вычислениям системы.

Модели Крипке

Рассмотрим множество атомарных высказываний AP.

Моделью Крипке M над множеством атомарных высказываний AP назовем четверку $M = (S, S_0, R, L)$, в которой:

1) S — конечное множество состояний;
2) $S_0 \subseteq S$ — множество начальных состояний;
3) $R \subseteq S \times S$ — отношение переходов, которое обязано быть тотальным, т. е. для каждого состояния $s \in S$ должно существовать такое состояние $s' \in S$, что имеет место $R(s, s')$;
4) $L \colon S \to 2^{AP}$ — функция разметки, которая помечает каждое состояние множеством атомарных высказываний, истинных в этом состоянии.

Модели Крипке

Путь в модели M из состояния s_0 — это такая бесконечная последовательность состояний $\pi = s_0, s_1, s_2, \ldots$, что $s_0 = s$ и для всех $i \geqslant 0$ выполняется $R(s_i, s_{i+1})$.

Состояние s называется достижимым состоянием модели, если через s проходит путь, ведущий из какого-либо начального состояния модели.

Состояние s называется тупиковым состоянием модели, если любой путь из s проходит проходит только через состояние s (образует петлю).

Иногда переходы модели Крипке помечаются именами тех действий программы, которые соответствуют этим переходам. В этом случае вводится множество действий Act и отношение переходов задается тройками $R \subseteq S \times Act \times S$.

Модель Крипке: Переправа.

Модель Крипке: Переправа.

Модель Крипке: Переправа.

Модель Крипке: Переправа.

Модель Крипке: Переправа.

Модель Крипке: Переправа.

Лодочник может переправиться через реку

один

Модель Крипке: Переправа.

Лодочник может переправиться через реку

один

Модель Крипке: Переправа.

или с пассажиром

Модель Крипке: Переправа.

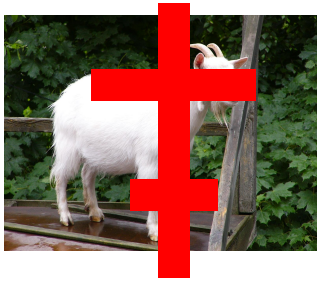
или с пассажиром

Модель Крипке: Переправа.

Без присмотра лодочника

некоторые пассажиры могут пострадать

Модель Крипке: Переправа.



Без присмотра лодочника

некоторые пассажиры могут пострадать

Модель Крипке: Переправа.

Что нам важно знать?

Важно знать

- жив персонаж или нет,
- где находится персонаж.

Модель Крипке: Переправа.

Множество состояний модели Крипке Переправа

$$S = \{-1, 0, 1\}^4 .$$

Для каждого состояния (x_1, x_2, x_3, x_4)

$x_i = -1$ означает, что персонаж i находится на левом берегу,

$x_i = 1$ означает, что персонаж i находится на правом берегу,

$x_i = 0$ означает, что персонажа i уже не существует.

Модель Крипке: Переправа.

Множество состояний модели Крипке Переправа

$$S = \{-1, 0, 1\}^4 \ .$$

Для каждого состояния (x_1, x_2, x_3, x_4)

$x_i = -1$ означает, что персонаж i находится на левом берегу,

$x_i = 1$ означает, что персонаж i находится на правом берегу,

$x_i = 0$ означает, что персонажа i уже не существует.

Множество начальных состояний

$$S_0 = \{(-1, -1, -1, -1)\}$$

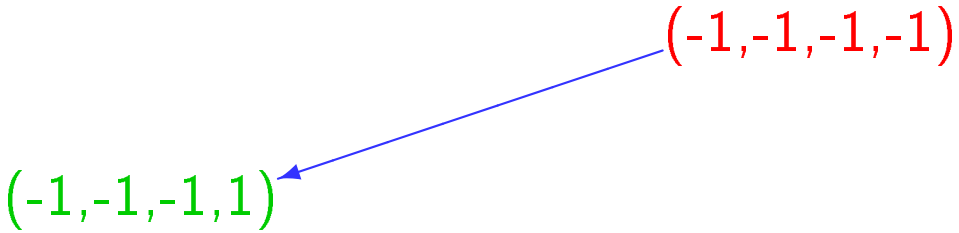
Модель Крипке: Переправа.

Отношение переходов R :

(-1,-1,-1,-1)

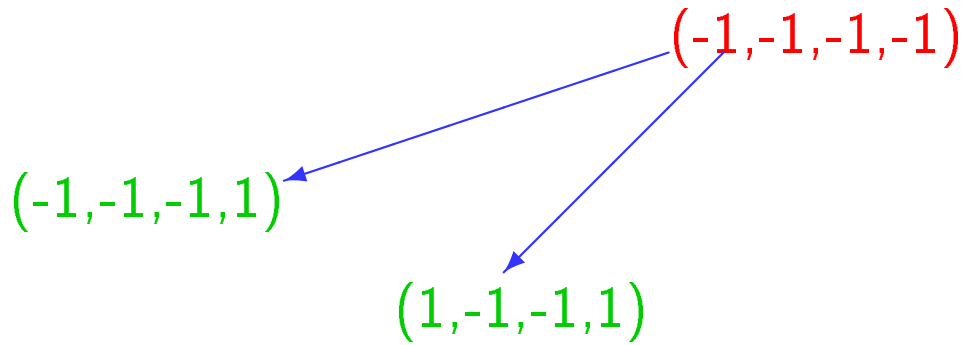
Модель Крипке: Переправа.

Отношение переходов R :



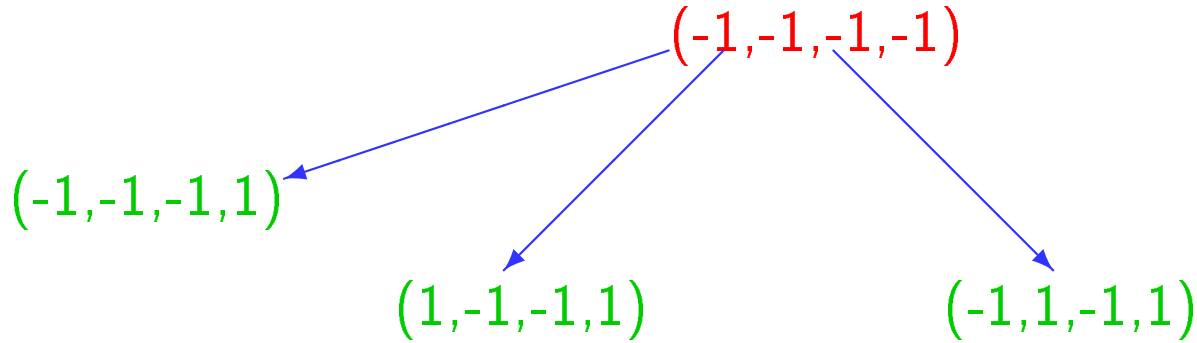
Модель Крипке: Переправа.

Отношение переходов R :



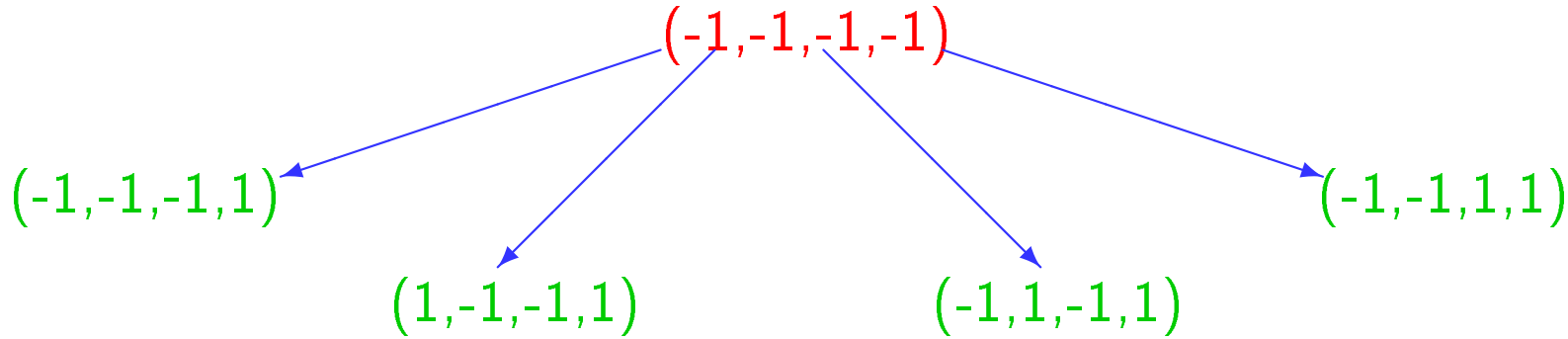
Модель Крипке: Переправа.

Отношение переходов R :



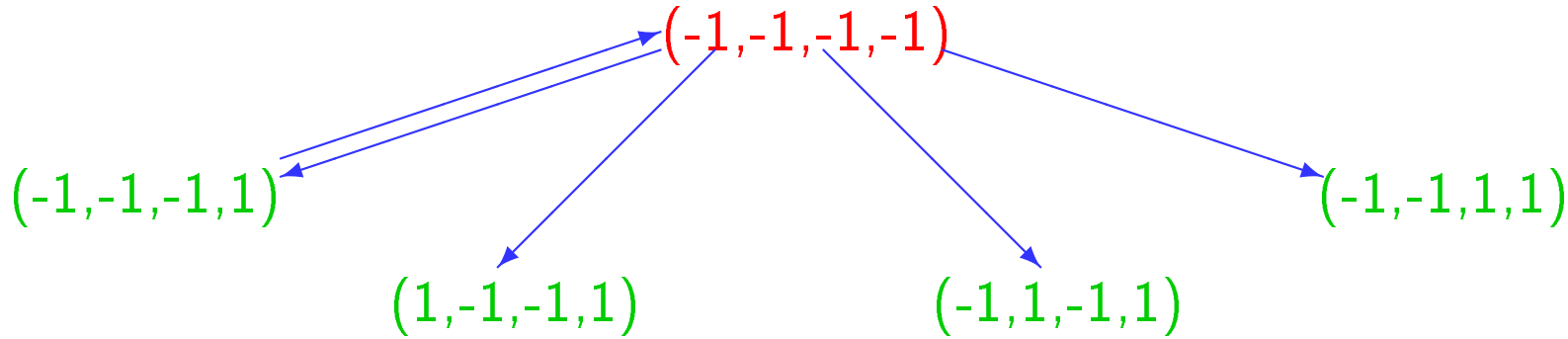
Модель Крипке: Переправа.

Отношение переходов R :



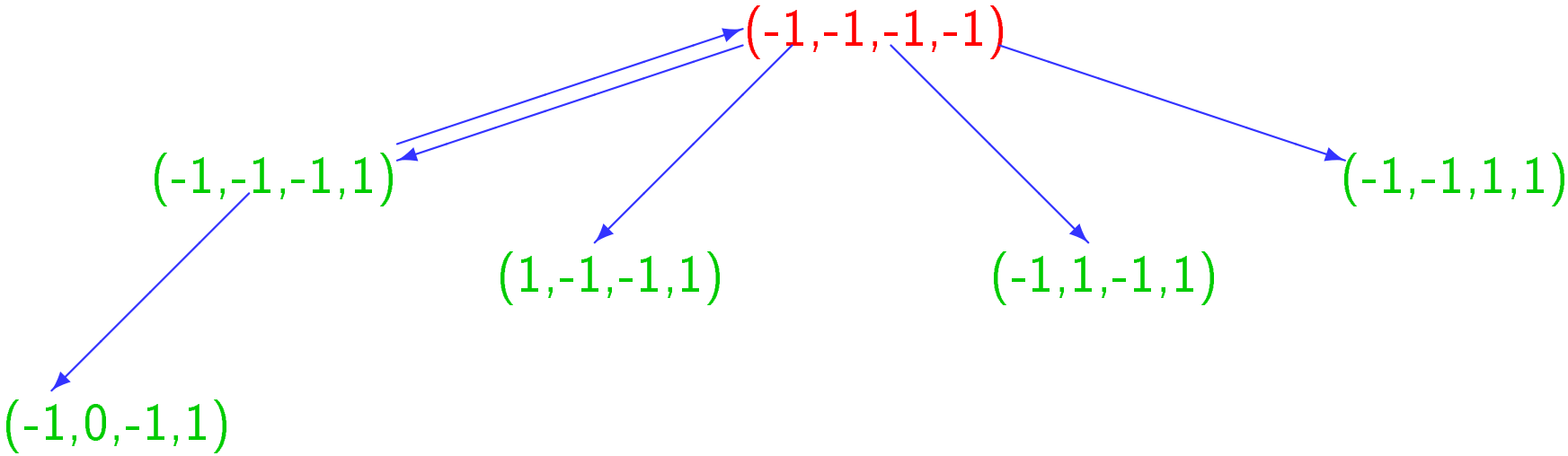
Модель Крипке: Переправа.

Отношение переходов R :



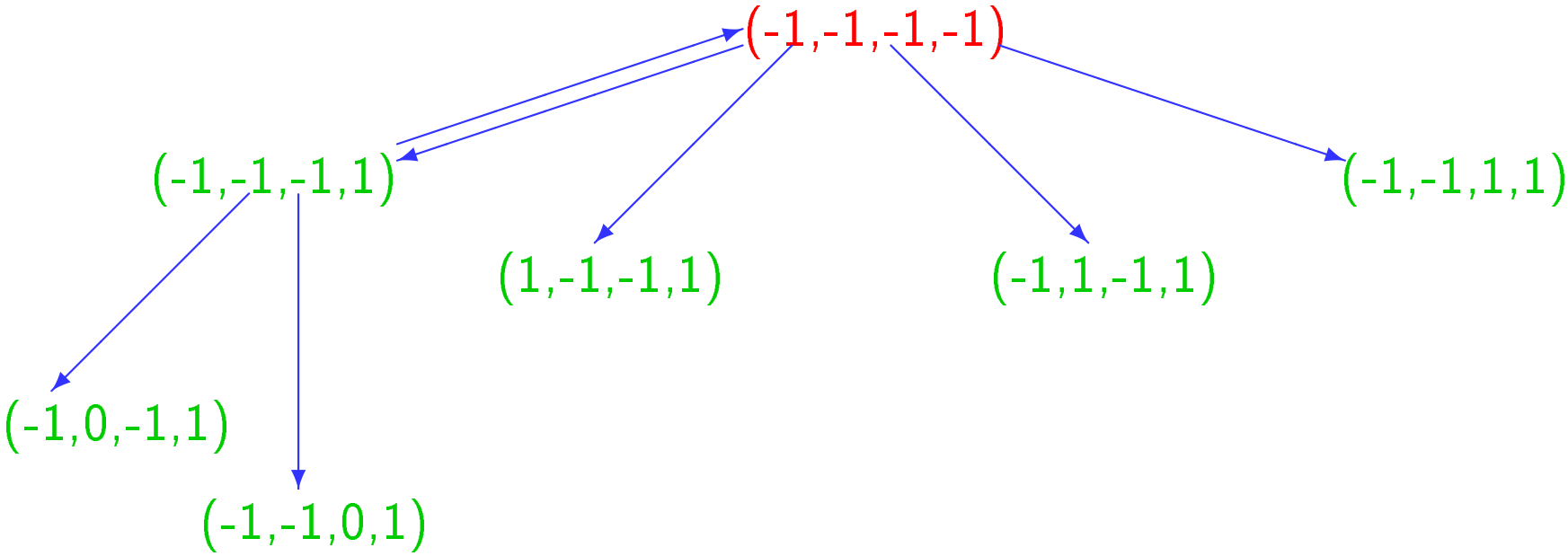
Модель Крипке: Переправа.

Отношение переходов R :



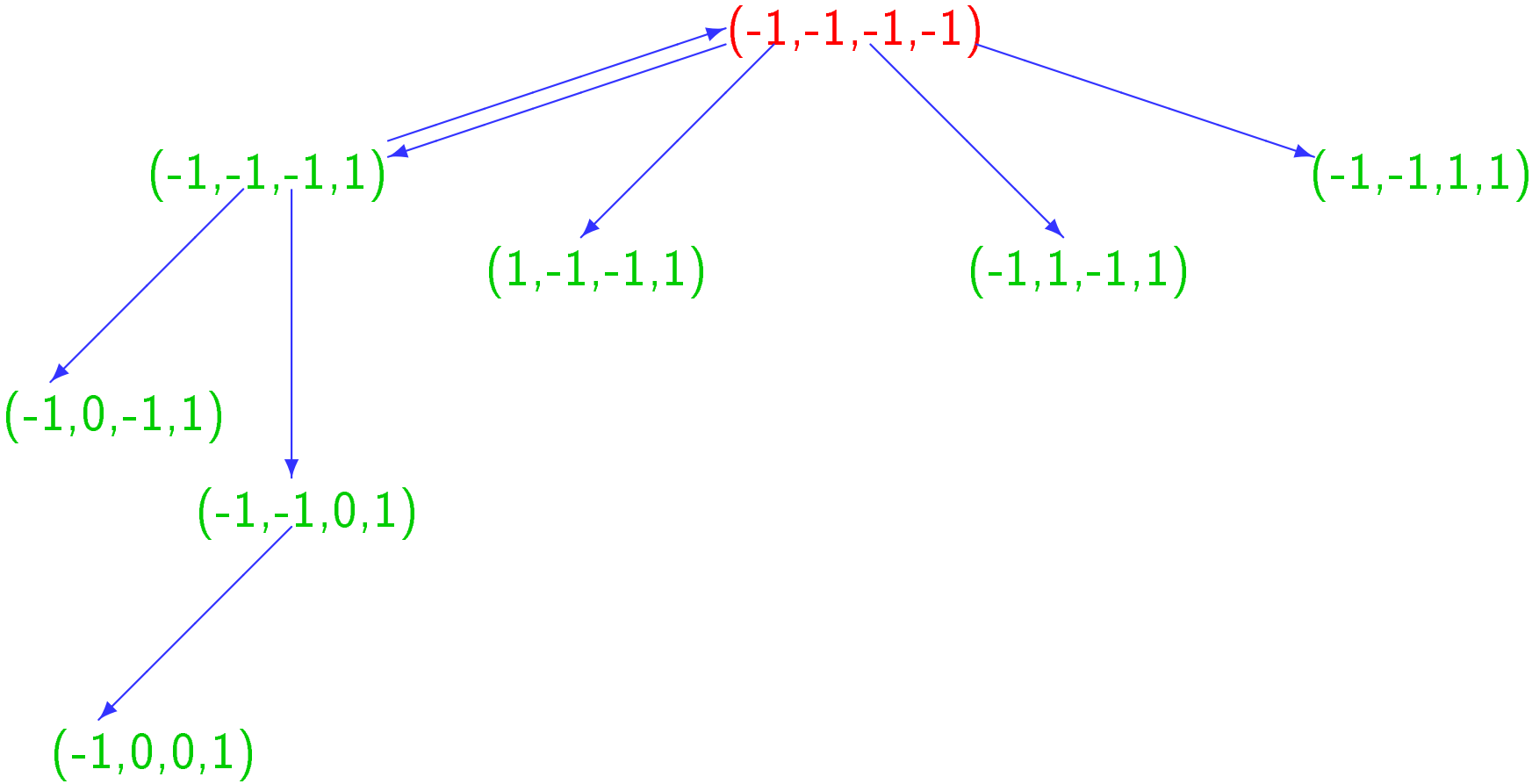
Модель Крипке: Переправа.

Отношение переходов R :



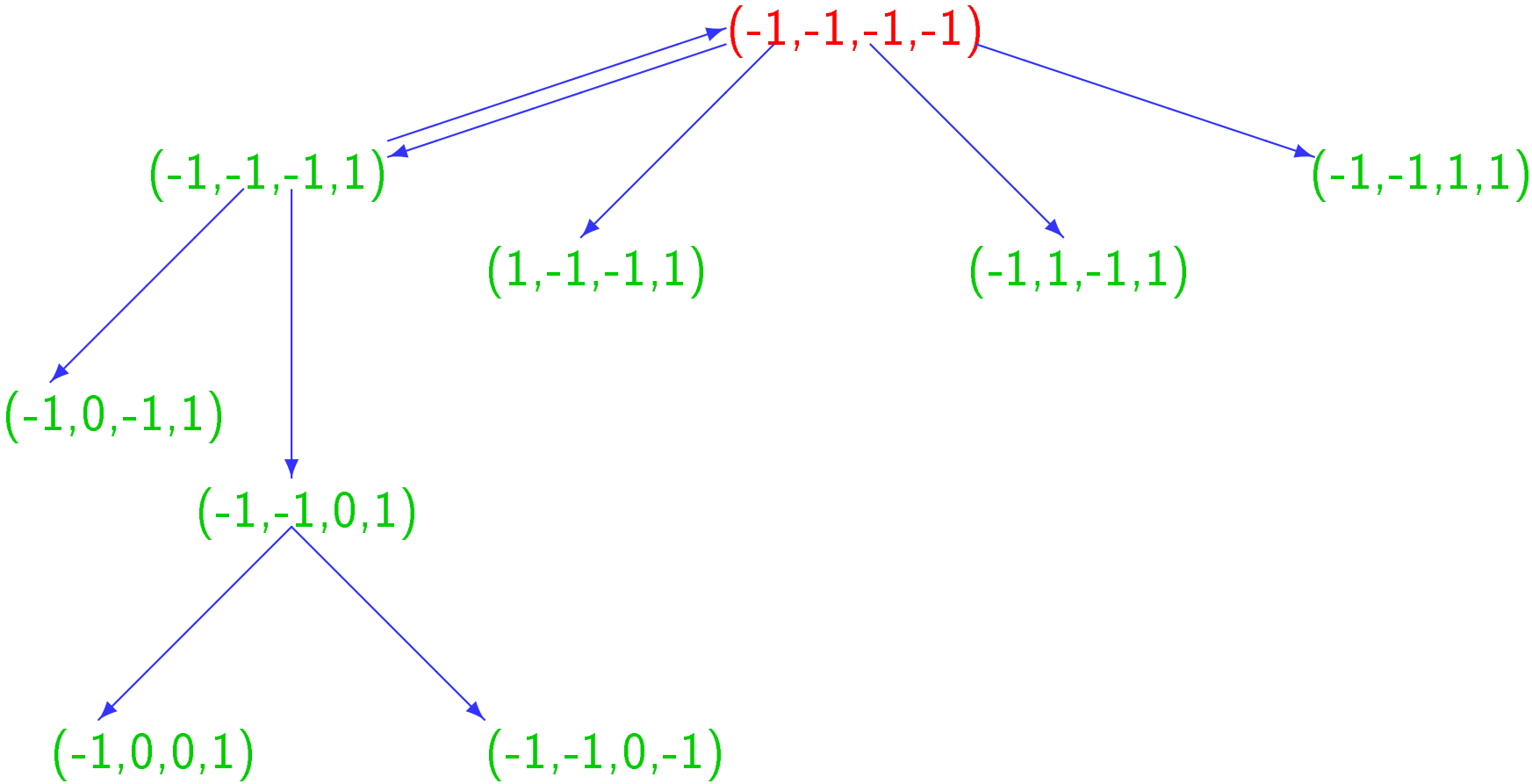
Модель Крипке: Переправа.

Отношение переходов R :



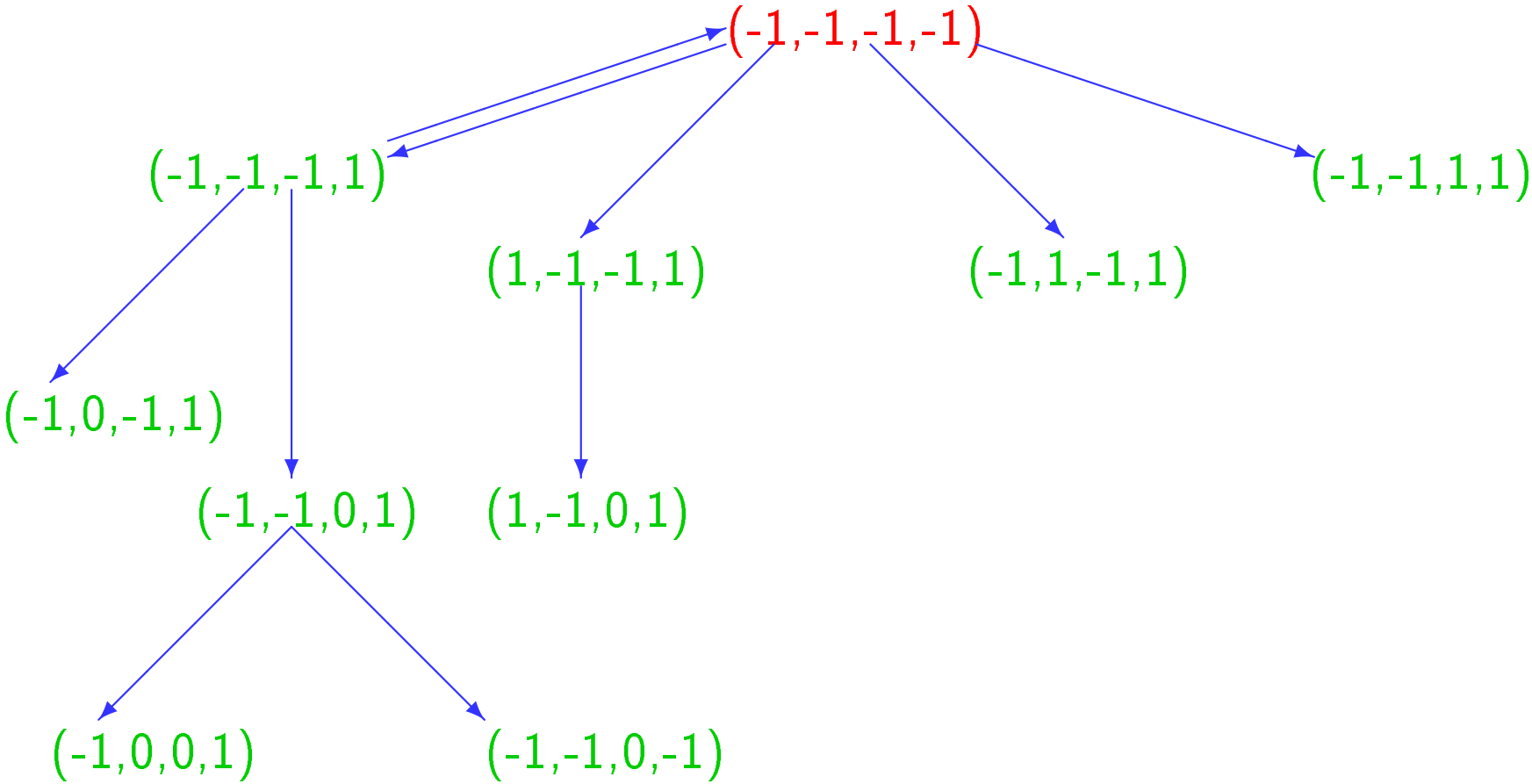
Модель Крипке: Переправа.

Отношение переходов R :



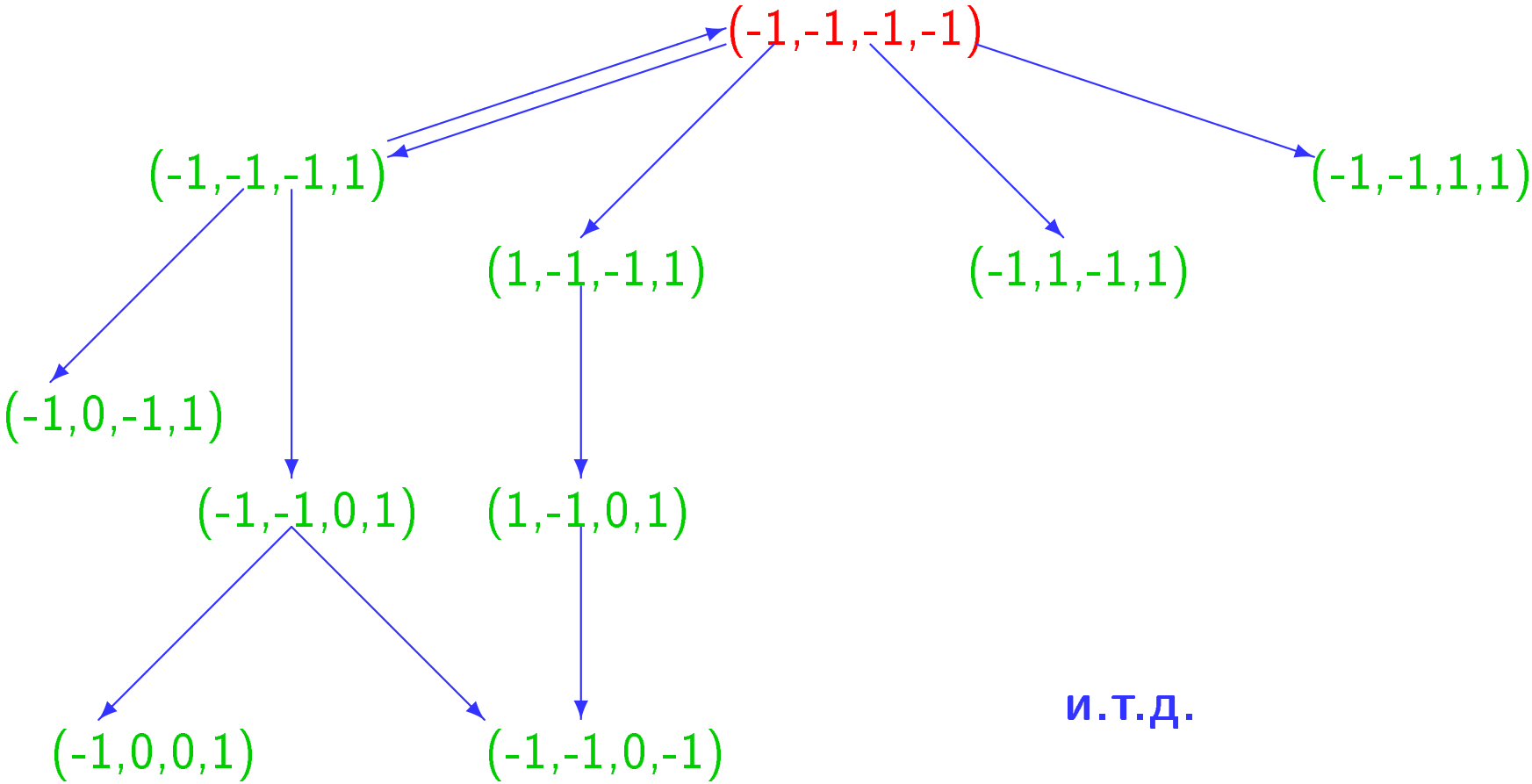
Модель Крипке: Переправа.

Отношение переходов R :



Модель Крипке: Переправа.

Отношение переходов R :



и.т.д.

Модель Крипке: Переправа.

Рассмотрим множество атомарных высказываний

$$AP = \{alive_i, left_i : i = 1, 2, 3, 4\}$$

Функция разметки L :

Модель Крипке: Переправа.

Рассмотрим множество атомарных высказываний

$$AP = \{alive_i, left_i : i = 1, 2, 3, 4\}$$

Функция разметки L :

$L((-1, -1, -1, -1)) = AP$,

$L((-1, 0, 1, 1)) = \{alive_1, alive_3, alive_4, left_1\}$,

и.т.д.

Принципы моделирования систем

Существуют различные типы параллельных систем (синхронные и асинхронные схемы, программы с разделяемыми переменными, программы, взаимодействующие путем обмена сообщениями, и т. д.). Ввиду такого разнообразия, нужен универсальный формализм, в рамках которого можно было бы представлять параллельную систему любого типа.

Для этого пригодны формулы логики предикатов первого порядка. По формуле, представляющей параллельную систему, строится модель Крипке, адекватно соответствующая этой системе.

Представления первого порядка

Для описания параллельных систем пригодны формулы первого порядка, имеющие фиксированную интерпретацию. Это означает, что предикатные и функциональные символы, встречающиеся в таких формулах, будут иметь предопределенное значение.

Рассмотрим множество $V = \{v_1, \ldots, v_n\}$ переменных системы. Мы предполагаем, что переменные из V принимают значения из конечного множества D, которое называется доменом или универсумом интерпретации.

Оценка для V — это функция, которая сопоставляет каждой переменной v из V значение из множества D.

Представления первого порядка

Состояние параллельной системы может быть описано указанием значений, сопоставленных всем элементам V. Другими словами, состояние — это оценка $s\colon V \to D$ для множества переменных V.

По заданной оценке можно написать формулу, истинную в точности на этой оценке. Например, для множества переменных $V = \{v_1, v_2, v_3\}$ и оценки $\langle v_1 \leftarrow 2, v_2 \leftarrow 3, v_3 \leftarrow 5\rangle$ можно вывести формулу $(v_1 = 2) \wedge (v_2 = 3) \wedge (v_3 = 5)$.

Одна и та же формула может быть истинна на многих оценках. Если мы условимся, что формула представляет множество *всех* оценок, которые делают ее истинной, то мы сможем описать конкретное множество состояний формулой первого порядка. В частности, начальные состояния системы могут быть описаны формулой первого порядка $\mathcal{S}_0$ над переменными из V.

Представления первого порядка

Для описания переходов между состояниями воспользуемся формулами для представления множества упорядоченных пар состояний.

Имея множество переменных системы V, создадим второй набор переменных V'. Мы будем рассматривать переменные из V как переменные текущего состояния, а переменные из V' — как переменные следующего состояния. Каждая переменная v из V имеет соответствующую переменную следующего состояния из V', которую мы обозначим v'.

Всякая оценка переменных из V и V' может рассматриваться как определение упорядоченной пары состояний, или перехода, а множества таких оценок можно представлять, используя формулы, как было указано ранее.

Мы будем называть множество пар состояний отношением переходов. Если R есть отношение переходов, то запись $\mathcal{R}(V, V')$ будет обозначать формулу, которая его представляет.

Представления первого порядка

Для того чтобы написать спецификации, которые выражают свойства параллельной системы, нужно определить множество атомарных высказываний AP.

Атомарные высказывания обычно имеют вид $v = d$, где $v \in V$ и $d \in D$.

Высказывание $v = d$ истинно в состоянии s, если $s(v) = d$.

Когда v — переменная над логическим доменом $\{True, False\}$, нет необходимости включать и $v = True$, и $v = False$ в AP. Вместо обозначения $v = True$ мы будем писать просто v, а вместо $v = False$ — просто $\neg v$.

Представления первого порядка

Теперь покажем, как построить модель Крипке $M = (S, S_0, R, L)$ на основе формул первого порядка $\mathcal{S}_0$ и $\mathcal{R}$.

- Множество состояний S есть множество всех оценок над V
- Множество начальных состояний S_0 есть множество всех оценок s_0 над V, которые удовлетворяют формуле $\mathcal{S}_0$.
- Какова бы ни была пара состояний s и s', отношение $R(s, s')$ соблюдается в том и только том случае, когда формула $\mathcal{R}$ обращается в $True$, после того как каждой переменной $v \in V$ присвоено значение $s(v)$ и каждой переменной $v' \in V'$ — значение $s'(v')$.
- Функция разметки $L\colon S \to 2^{AP}$ определяется так, чтобы $L(s)$ было множеством всех атомарных высказываний, истинных в s. Если v — логическая переменная, то $v \in L(s)$ означает, что $s(v) = True$, а $v \notin L(s)$ означает, что $s(v) = False$.

Пример

Рассмотрим простую систему с переменными x и y, принимающими значения из множества $D = \{0, 1\}$. Сответственно оценкой переменных x и y будет просто пара $(d_1, d_2) \in D \times D$, где d_1 — значение x, а d_2 — значение y. Система содержит один переход, который определяется действием

$$x := (x + y)(\mathrm{mod} 2),$$

и начальным значением данных $x = 1$ и $y = 1$.

Пример

Система будет охарактеризована двумя формулами первого порядка.

$$\mathcal{S}_0(x, y) \equiv x = 1 \wedge y = 1 ,$$
$$\mathcal{R}(x, y, x', y') \equiv x' = (x + y)(\mathrm{mod}2) \wedge y' = y .$$

Модель Крипке $M = (S, S_0, R, L)$ такова:

- $S = D \times D$;
- $S_0 = \{(1, 1)\}$;
- $R = \{((1, 1), (0, 1)), ((0, 1), (1, 1)),$
  $((1, 0), (1, 0)), ((0, 0), (0, 0))\}$;
- $L((1, 1)) = \{x = 1, y = 1\}, L((0, 1)) = \{x = 0, y = 1\}$,
  $L((1, 0)) = \{x = 1, y = 0\}, L((0, 0)) = \{x = 0, y = 0\}$.

Единственный путь в модели Крипке, исходящий из начального состояния, имеет вид $(1, 1)(0, 1)(1, 1)(0, 1) \ldots$

Степень детализации описания

При моделировании параллельных систем большую роль играет степень детализации (гранулярность) перехода. Важно достичь атомарности переходов, т. е. добиться того, чтобы ни одно наблюдаемое состояние системы не являлось результатом выполнения всего лишь некоторой части одного из переходов.

Определив переходы чрезмерно крупно, мы совершим распространенную ошибку. В этом случае модель Крипке может не включать некоторые наблюдаемые состояния. В результате методы верификации наподобие проверки на модели могут пропустить некоторые существенные ошибки системы.

Проблемы возникают и при излишне подробном описании. В этом случае переходы могут взаимодействовать, образуя новые состояния, которые никогда не будут достигнуты в реальной системе. Поэтому при верификации модели могут обнаружиться призрачные ошибки, которые никогда не случаются на практике.

Модель Крипке: Переправа.

Модель Крипке: Переправа.

Нужно ли считать отдельным действием

плавание по реке?

Модель Крипке: Переправа.

Нужно ли считать отдельным действием

плавание по реке?

посадку на лодку?

Модель Крипке: Переправа.

Нужно ли считать отдельным действием

- плавание по реке?
- посадку на лодку?
- высадку с лодки?

Модель Крипке: Переправа.

Нужно ли считать отдельным действием

плавание по реке?

посадку на лодку?

высадку с лодки?

А можно ли считать отдельным действием

плавание лодочника в обе стороны?

Степень детализации описания

Рассмотрим систему с двумя переменными x и y и двумя переходами α и β, которые могут исполняться параллельно:

$$\begin{aligned} \alpha: &\quad x{:=}x+y, \\ \beta: &\quad y{:=}y+x, \end{aligned}$$

с начальным состоянием $x=1 \ \wedge \ y=2$.

Степень детализации описания

Рассмотрим систему с двумя переменными x и y и двумя переходами α и β, которые могут исполняться параллельно:

$$\begin{aligned} \alpha: &\quad x{:=}x+y, \\ \beta: &\quad y{:=}y+x, \end{aligned}$$

с начальным состоянием $x=1 \ \wedge \ y=2$.

Кроме того, рассмотрим более детальную реализацию этих переходов ная языке ассемблера:

$$\begin{aligned} \alpha_0: &\quad \text{load } R_1, x \\ \alpha_1: &\quad \text{add } R_1, y \\ \alpha_2: &\quad \text{store } R_1, x \end{aligned} \qquad \begin{aligned} \beta_0: &\quad \text{load } R_2, y \\ \beta_1: &\quad \text{add } R_2, x \\ \beta_2: &\quad \text{store } R_2, y \end{aligned}$$

Степень детализации описания

Вычисление α , β приводит программу в состояние $x = 3 \ \wedge \ y = 5$.

А при вычислении β , α , мы получаем $x = 4 \ \wedge \ y = 3$.

Степень детализации описания

Вычисление α , β приводит программу в состояние $x = 3 \ \wedge \ y = 5$.
А при вычислении β , α , мы получаем $x = 4 \ \wedge \ y = 3$.

Порядок исполнения операций более детальной реализации может быть таков:

$$\alpha_0, \ \beta_0, \ \alpha_1, \ \beta_1, \ \alpha_2, \ \beta_2,$$

и результатом будет состояние $x = 3 \ \wedge \ y = 3$.

Корректность системы зависит от того, в какой модели параллельных вычислений реализуется эта программа.

Параллельные системы

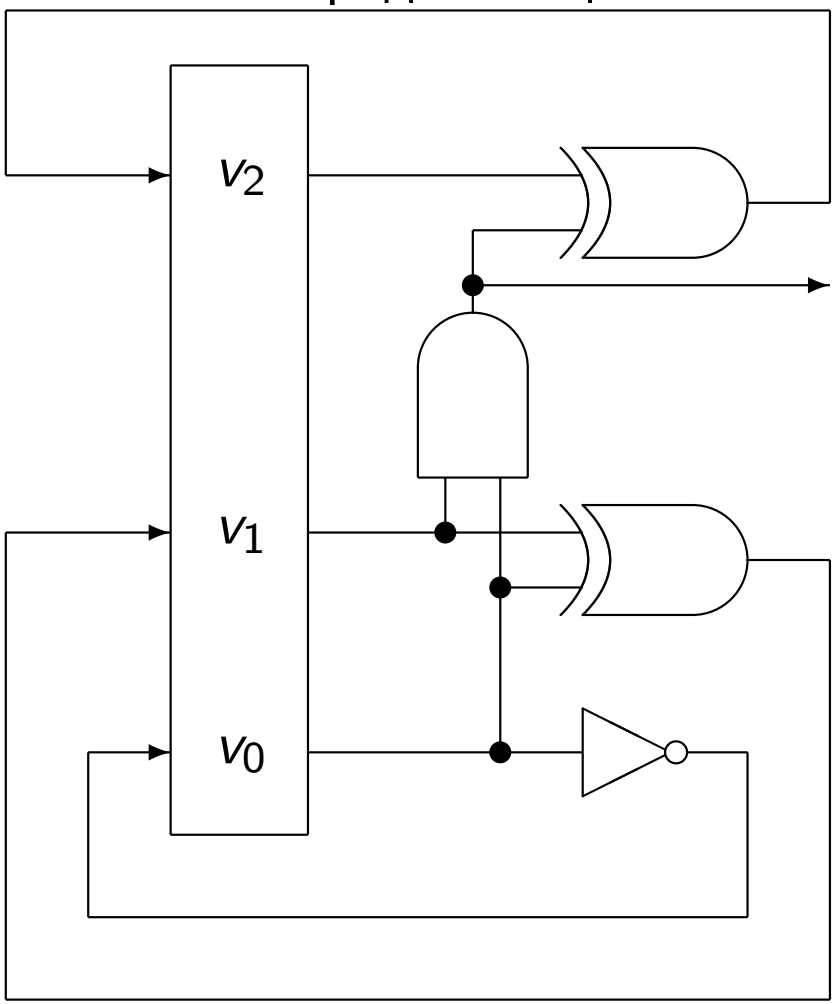
Параллельные системы состоят из набора компонентов, которые исполняются совместно. Обычно компоненты наделены некоторыми средствами взаимодействия друг с другом. Режим исполнения и тип взаимодействия в разных системах могут различаться.

Два основных типа исполнения: асинхронное , или чередующееся исполнение , в котором в любой момент времени только один компонент делает шаг вычисления, и синхронное исполнение , в котором все компоненты делают шаг вычисления одновременно.

Два основных механизма взаимодействия: либо путем изменения значения общих переменных , либо путем обмена сообщениями .

Синхронные схемы

На каждом шаге работы синхронной электронной схемы изменяются ее входы, и схеме дают стабилизироваться. Затем через схему проходит импульсный синхронизирующий сигнал, и элементы, определяющие состояние, изменяются.



Синхронные схемы

Переходы счетчика задаются уравнениями

$$v_0' = \neg v_0 \ ,$$
$$v_1' = v_0 \oplus v_1 \ ,$$
$$v_2' = (v_0 \wedge v_1) \oplus v_2 \ ,$$

Эти уравнения можно использовать для определения отношений

$$\mathcal{R}_0(V, V') \equiv (v_0' \Leftrightarrow \neg v_0) \ ,$$
$$\mathcal{R}_1(V, V') \equiv (v_1' \Leftrightarrow v_0 \oplus v_1) \ ,$$
$$\mathcal{R}_2(V, V') \equiv (v_2' \Leftrightarrow (v_0 \wedge v_1) \oplus v_2) \ ,$$

описываюших ограничения, которым должны удовлетворять все переменные v_i' в допустимом переходе. Так как все изменения происходят в одно и то же время, для построения формулы отношения переходов ограничения соединяются конъюнкцией:

$$\mathcal{R}(V, V') \equiv \mathcal{R}_0(V, V') \wedge \mathcal{R}_1(V, V') \wedge \mathcal{R}_2(V, V') \ .$$

Асинхронные схемы

Отношение переходов для асинхронных схем выражается дизъюнкцией. Предположим, что все компоненты схемы имеют ровно один выход и не имеют внутренних переменных состояния. В этом случае каждый компонент можно охарактеризовать функцией $f_i(v)$; по заданным значениям переменных текущего состояния v компонент выдает на выходе значение $f_i(v)$.

Так как значение компонента изменяется достаточно быстро, маловероятно, чтобы два компонента изменили свое значение одновременно. Поэтому обычно используется семантика чередования , при которой в каждый момент времени только один единственный компонент изменяет свое состояние. Это отражается в дизъюнкции вида

$$\mathcal{R}(V, V') \equiv \mathcal{R}_0(V, V') \vee \cdots \vee \mathcal{R}_{n-1}(V, V') \ ,$$

где $\mathcal{R}_i(V, V') \equiv \left(v_i' \Leftrightarrow f_i(V)\right) \wedge \bigwedge_{j \neq i}(v_j' \Leftrightarrow v_j)$.

Трансляция программ в модели Крипке

Определим механизм трансляции $\mathcal{C}$, который преобразует текст последовательной программы P в формулу первого порядка $\mathcal{R}$, представляющую множество переходов программы.

Наши программы:

- $x := e$, **skip**, **wait**(b);
- $\pi = \pi_1; \pi_2;$
- $\pi =$ **if** b **then** π_1 **else** π_2 **fi**,
- $\pi =$ **while** b **do** π_1 **od**,
- **cobegin** $\pi_1 \parallel \pi_2 \parallel \cdots \parallel \pi_m$ **coend**.

Трансляция программ в модели Крипке

Не ограничивая общности, предположим, что каждый оператор имеет единственную точку входа и единственную точку выхода. Все метки считаются попарно различными. Процедура трансляции совмещает точку выхода каждого оператора с точкой входа следующего за ним оператора. В результате получается уникальная разметка точек входа и выхода для всех операторов программы.

Трансляция программ в модели Крипке

Не ограничивая общности, предположим, что каждый оператор имеет единственную точку входа и единственную точку выхода. Все метки считаются попарно различными. Процедура трансляции совмещает точку выхода каждого оператора с точкой входа следующего за ним оператора. В результате получается уникальная разметка точек входа и выхода для всех операторов программы.

x:=y; **if** x>0 **then** y:=z **else** z:=x **fi**; x:=z;

Трансляция программ в модели Крипке

Не ограничивая общности, предположим, что каждый оператор имеет единственную точку входа и единственную точку выхода . Все метки считаются попарно различными. Процедура трансляции совмещает точку выхода каждого оператора с точкой входа следующего за ним оператора. В результате получается уникальная разметка точек входа и выхода для всех операторов программы.

x:=y; **if** x>0 **then** y:=z **else** z:=x **fi**; x:=z;

0: x:=y;:1 2: **if** x>0 **then** 3: y:=z:4 **else** 5: z:=x:6 **fi**;:7 8: x:=z;:9

Трансляция программ в модели Крипке

Не ограничивая общности, предположим, что каждый оператор имеет единственную точку входа и единственную точку выхода . Все метки считаются попарно различными. Процедура трансляции совмещает точку выхода каждого оператора с точкой входа следующего за ним оператора. В результате получается уникальная разметка точек входа и выхода для всех операторов программы.

x:=y; **if** x>0 **then** y:=z **else** z:=x **fi**; x:=z;

0: x:=y;:1 2: **if** x>0 **then** 3: y:=z:4 **else** 5: z:=x:6 **fi**;:7 8: x:=z;:9

0: x:=y;:1 1: **if** x>0 **then** 3: y:=z:4 **else** 5: z:=x:4 **fi**;:4 4: x:=z;:9

Трансляция программ в модели Крипке

Введем специальную переменную pc, которая называется счетчиком команд; ее значениями являются метки программы, а также особый элемент $\perp$, который соответствует неопределенному значению. Неопределенное значение требуется, когда речь идет о параллельных программах. В таком случае равенство $pc = \perp$ является признаком того, что программа неактивна.

Пусть V — множество переменных программы. Определим множество V' штрихованых переменных v', по одной для каждой переменной $v \in V$, а также штрихованую переменную pc' для счетчика команд pc.

Так как каждый переход обычно изменяет небольшое число переменных программы, запись $same(Y)$ будет использоваться для обозначения формулы

$$\bigwedge_{y \in Y} (y' = y) .$$

Трансляция программ в модели Крипке

Сначала выпишем формулу, описывающую множество начальных состояний программы P. Для заданного условия $pre(V)$, определяющего начальные значения переменных программы P, она имеет вид

$$\mathcal{S}_0(V, pc) \equiv pre(V) \wedge pc = m.$$

Процедура трансляции $\mathcal{C}$ зависит от трех параметров: входной метки ℓ, размеченного оператора P и выходной метки ℓ'. Процедура определяется рекурсивно и использует по одному правилу для каждого типа операторов языка. Предикат $\mathcal{C}(\ell, P, \ell')$ представляет множество переходов программы P в виде дизъюнкции всех переходов этого множества. Каждое слагаемое такой дизъюнкции, соответствующее отдельному переходу, определяет значение булевого условия и значение счетчика команд, для которых возможно выполнение этого перехода. Указанное слагаемое обращается в истину всякий раз, когда переход осуществим; в противном случае оно будет ложным.

Трансляция программ в модели Крипке

Оператор присваивания:

$$\mathcal{C}(\ell,\ v := e,\ \ell') \ \equiv$$
$$pc = \ell \ \wedge \ pc' = \ell' \ \wedge \ v' = e \ \wedge \ same(V \setminus \{v\}) \ .$$

Трансляция программ в модели Крипке

Оператор присваивания:

$$\mathcal{C}(\ell,\ v := e,\ \ell') \ \equiv \\ pc = \ell \ \wedge \ pc' = \ell' \ \wedge \ v' = e \ \wedge \ same(V \setminus \{v\}) \ .$$

Оператор **skip**:

$$\mathcal{C}(\ell, \textbf{skip},\ \ell') \ \equiv \ pc = \ell \wedge \ (pc' = \ell \ \vee \ pc' = \ell') \ \wedge \ same(V) \ .$$

Трансляция программ в модели Крипке

Оператор присваивания:

$$\mathcal{C}(\ell,\ v := e,\ \ell') \equiv \\ pc = \ell \ \wedge\ pc' = \ell' \ \wedge\ v' = e \ \wedge\ same(V \setminus \{v\})\ .$$

Оператор **skip**:

$$\mathcal{C}(\ell, \textbf{skip},\ \ell') \equiv\ pc = \ell \wedge\ (pc' = \ell \ \vee\ pc' = \ell') \ \wedge\ same(V)\ .$$

Последовательная композиция операторов:

$$\mathcal{C}(\ell,\ P_1;\ \ell'' : P_2,\ \ell') \equiv\ \mathcal{C}(\ell, P_1, \ell'') \ \wedge\ \mathcal{C}(\ell'', P_2, \ell')\ .$$

Трансляция программ в модели Крипке

Оператор ветвления **if-then-else**:

$\mathcal{C}(\ell,\ \textbf{if}\ b\ \textbf{then}\ \ell_1 : P_1\ \textbf{else}\ \ell_2 : P_2\ \textbf{end if},\ \ell')$
представляет собой дизъюнкцию следующих четырех формул:

- $pc = \ell\ \wedge\ pc' = \ell_1\ \wedge\ b\ \wedge\ same(V)$,
- $pc = \ell\ \wedge\ pc' = \ell_2\ \wedge\ \neg b\ \wedge\ same(V)$,
- $\mathcal{C}(\ell_1, P_1, \ell')$,
- $\mathcal{C}(\ell_2, P_2, \ell')$.

Первый дизъюнкт соответствует случаю, когда условие b истинно. При этом следующим будет выполняться оператор P_1 . Второй дизъюнкт соответствует случаю, когда условие b ложно. Тогда следующим будет исполняться оператор P_2 . Оба дизъюнкта описывают переходы, занятые исключительно изменением показания счетчика команд. Третий и четвертый дизъюнкты — это формулы для переходов P_1 и P_2 .

Трансляция программ в модели Крипке

Оператор итерации **while-do**:

$\mathcal{C}(\ell,$ **while** b **do** $\ell_1 : P_1$ **end while**, $\ell')$
представляет собой дизъюнкцию следующих трех формул:

- $pc = \ell \ \wedge \ pc' = \ell_1 \ \wedge \ b \ \wedge \ same(V)$,
- $pc = \ell \ \wedge \ pc' = \ell' \ \wedge \ \neg b \ \wedge \ same(V)$,
- $\mathcal{C}(\ell_1, P_1, \ell)$.

Первый дизъюнкт соответствует случаю, когда условие b истинно. При этом следующим будет исполняться оператор P_1 . Второй дизъюнкт соответствует случаю, когда условие b ложно, и тогда исполнение оператора итерации завершается. Третий дизъюнкт — это формула для переходов оператора P_1 . Заметим, что точка выхода оператора P_1 совпадает с точкой входа в оператор итерации. Таким образом, как только завершается исполнение оператора P_1 , оператор итерации запускается повторно.

Трансляция программ в модели Крипке

Параллельная композиция P :

$P = \ell : \textbf{cobegin } \ell_1 : P_1^{\mathcal{L}} \ell'_1 \parallel \ell_2 : P_2^{\mathcal{L}} \ell'_2 \parallel \ldots \parallel \ell_n : P_n^{\mathcal{L}} \ell'_n \textbf{ coend} : L'.$

Трансляция программ в модели Крипке

Параллельная композиция P :
$P = \ell : \textbf{cobegin}\ \ell_1 : P_1^{\mathcal{L}} \ell_1' \parallel \ell_2 : P_2^{\mathcal{L}} \ell_2' \parallel \ldots \parallel \ell_n : P_n^{\mathcal{L}} \ell_n'\ \textbf{coend} : L'.$

формула
$\mathcal{C}(\ell,\ \textbf{cobegin}\ P_1 \parallel P_2 \parallel \ldots \parallel P_n\ \textbf{coend},\ \ell')$
представляет собой дизъюнкцию трех следующих подформул:

- $pc = \ell\ \wedge\ pc_1' = \ell_1\ \wedge \ldots \wedge\ pc_n' = \ell_n\ \wedge\ pc' = \bot$,
- $pc = \bot\ \wedge\ pc_1 = \ell_1'\ \wedge \ldots \wedge\ pc_n = \ell_n'\ \wedge\ pc' = \ell'\ \wedge$
  $\wedge\ \bigwedge_{i=1}^{n}(pc_i' = \bot)$,
- $\bigvee_{i=1}^{n}\big(\mathcal{C}(\ell_i, P_i, \ell_i')\ \wedge\ same(V \setminus V_i)\ \wedge\ same(PC \setminus \{pc_i\})\big)$.

Первый дизъюнкт описывает инициализацию параллельных процессов. Второй дизъюнкт описывает завершение параллельной программы. Третий дизъюнкт описывает чередующееся исполнение параллельных процессов.

Трансляция программ в модели Крипке

Оператор **wait**.

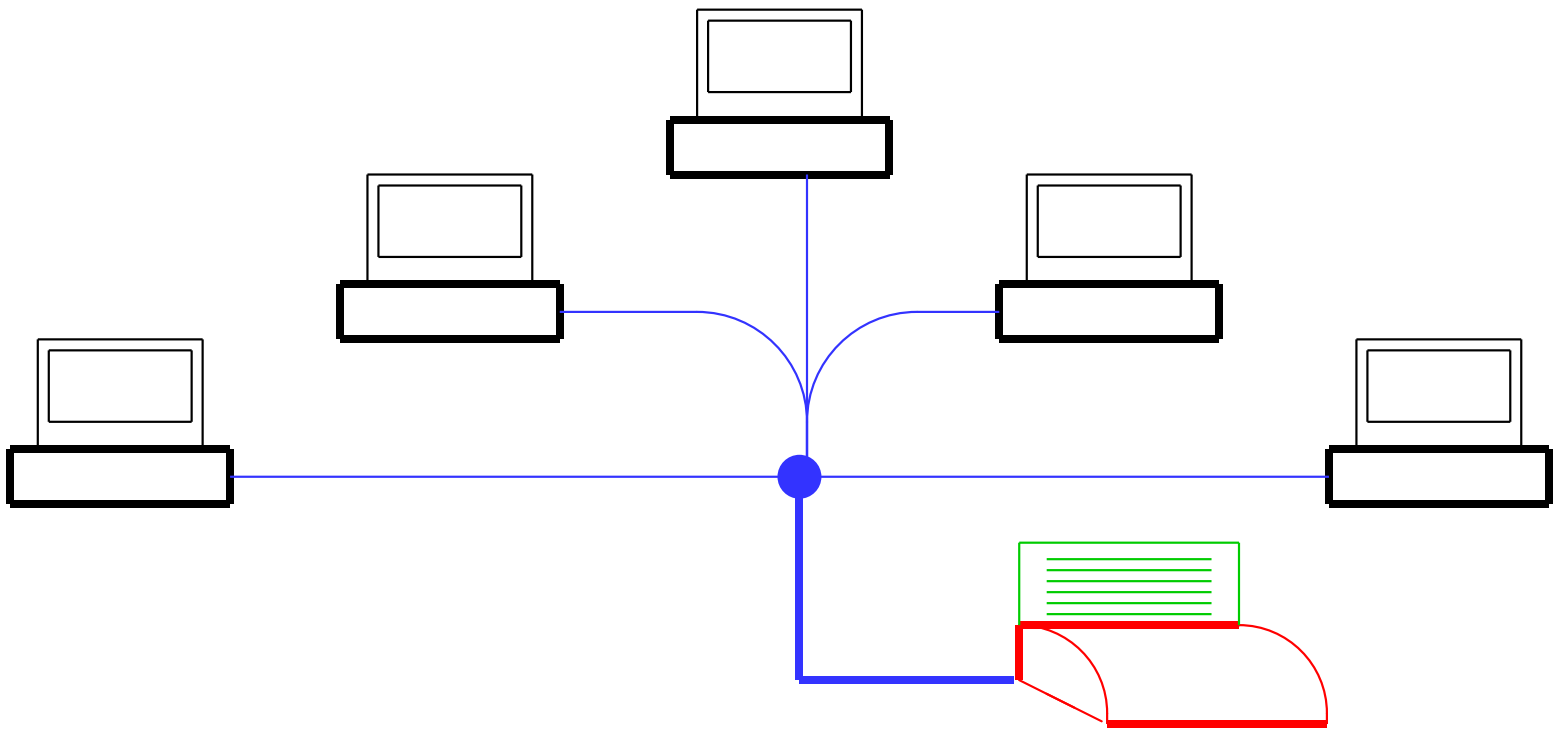
Оператор **wait**(b) периодически проверяет значение булевой переменной b, до тех пор пока он не обнаружит, что b обратилась в истину. Когда переменная b стала истинной, осуществляется переход к последующему оператору программы.
Формула $\mathcal{C}(\ell,\ \mathbf{wait}(b),\ \ell')$ представляет собой дизъюнкцию следующих двух подформул:

$$pc_i = \ell \ \wedge \ pc_i' = \ell \ \wedge \ \neg b \ \wedge \ same(V_i)\ ,$$
$$pc_i = \ell \ \wedge \ pc_i' = \ell' \ \wedge \ b \ \wedge \ same(V_i)\ .$$

Пример программы

Задача

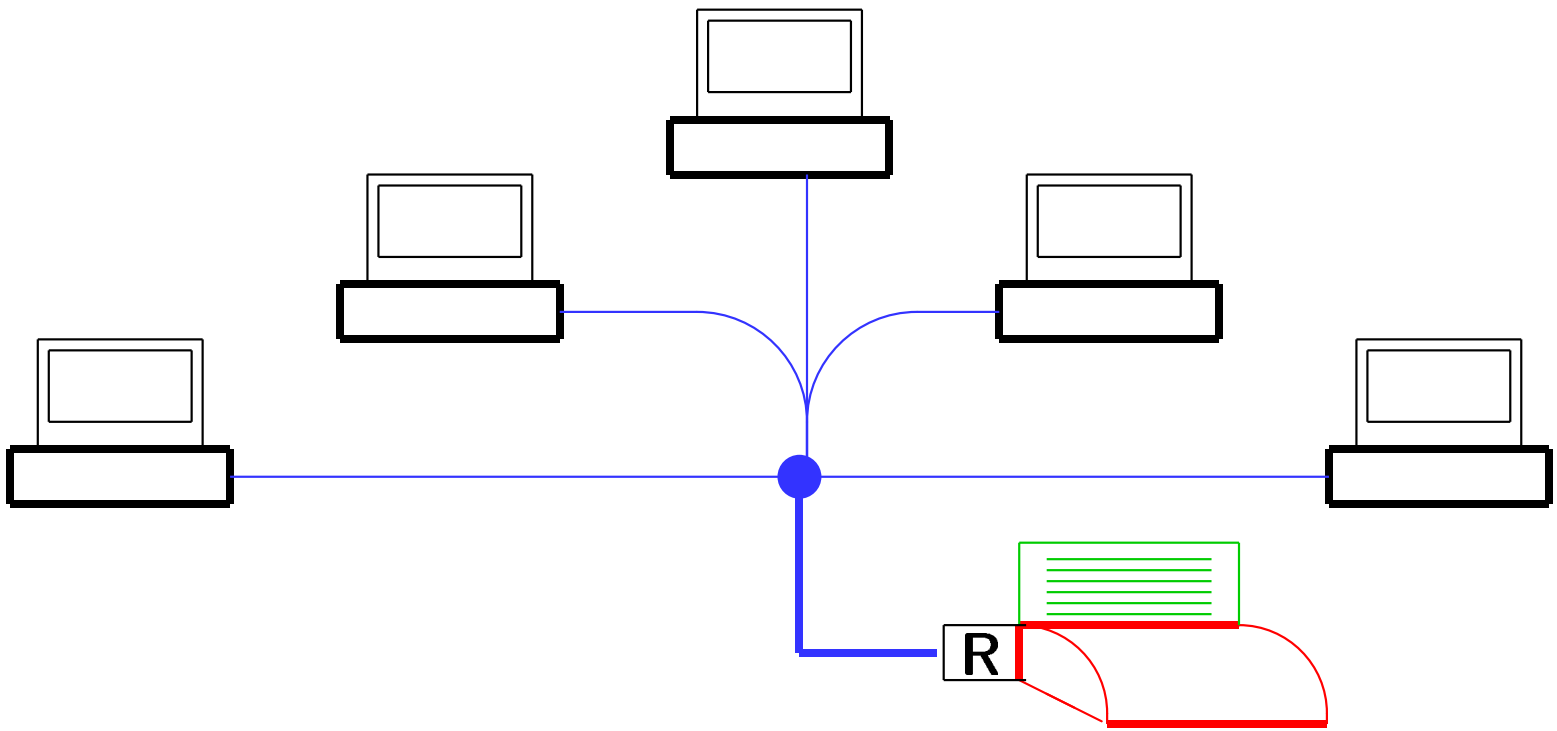
Имеется несколько компьютеров и только один принтер. Ни один компьютер не осведомлен о существовании других компьютеров. Как правильно организовать их взаимодействие, чтобы все они могли пользоваться этим принтером?



Пример программы

Задача

Предполагается также, что у принтера есть единственный однобитовый регистр **R**, общедоступный для считывания и записи. Этот регистр может находится в одном из двух состояний — *busy* (принтер занят) и *free* (принтер свободен).



Пример программы

Прежде чем писать программу (драйвер), обеспечивающую взаимодействие каждого компьютера с принтером, нужно сформулировать требования, предъявляемые к этой программе.

Пример программы

Прежде чем писать программу (драйвер), обеспечивающую взаимодействие каждого компьютера с принтером, нужно сформулировать требования, предъявляемые к этой программе.

1. Всякий раз, когда принтер свободен и хотя бы один компьютер собирается отправить данные на печать, принтер будет рано или поздно занят;

Пример программы

Прежде чем писать программу (драйвер), обеспечивающую взаимодействие каждого компьютера с принтером, нужно сформулировать требования, предъявляемые к этой программе.

1. Всякий раз, когда принтер свободен и хотя бы один компьютер собирается отправить данные на печать, принтер будет рано или поздно занят;
2. Всякий раз, после того как принтер оказался занят, он должен когда-нибудь приступить к печати;

Пример программы

Прежде чем писать программу (драйвер), обеспечивающую взаимодействие каждого компьютера с принтером, нужно сформулировать требования, предъявляемые к этой программе.

1. Всякий раз, когда принтер свободен и хотя бы один компьютер собирается отправить данные на печать, принтер будет рано или поздно занят;
2. Всякий раз, после того как принтер оказался занят, он должен когда-нибудь приступить к печати;
3. Компьютер, завершивший печать, должен когда-нибудь освободить принтер;

Пример программы

Прежде чем писать программу (драйвер), обеспечивающую взаимодействие каждого компьютера с принтером, нужно сформулировать требования, предъявляемые к этой программе.

1. Всякий раз, когда принтер свободен и хотя бы один компьютер собирается отправить данные на печать, принтер будет рано или поздно занят;
2. Всякий раз, после того как принтер оказался занят, он должен когда-нибудь приступить к печати;
3. Компьютер, завершивший печать, должен когда-нибудь освободить принтер;
4. Данные на печать всегда передает не более чем один компьютер.

Пример программы

Прежде чем писать программу (драйвер), обеспечивающую взаимодействие каждого компьютера с принтером, нужно сформулировать требования, предъявляемые к этой программе.

1. Всякий раз, когда принтер свободен и хотя бы один компьютер собирается отправить данные на печать, принтер будет рано или поздно занят;
2. Всякий раз, после того как принтер оказался занят, он должен когда-нибудь приступить к печати;
3. Компьютер, завершивший печать, должен когда-нибудь освободить принтер;
4. Данные на печать всегда передает не более чем один компьютер.

А какие еще требования разумно предъявить к нашему драйверу?

Пример программы

Для связи с принтером программист предложил снабдить каждый компьютер одной и той же программой

π : **while** true **do**
wait (R=free);
R:=busy;
output(X,*printer*);
R:=free
od

Пример программы

Для связи с принтером программист предложил снабдить каждый компьютер одной и той же программой

π : **while** true **do**
wait (R=free);
R:=busy;
output(X,*printer*);
R:=free
od

Это простая и разумная программа.
Но будет ли система компьютеров, снабженных этой программой, вести себя в соответствии с указанными требованиями?

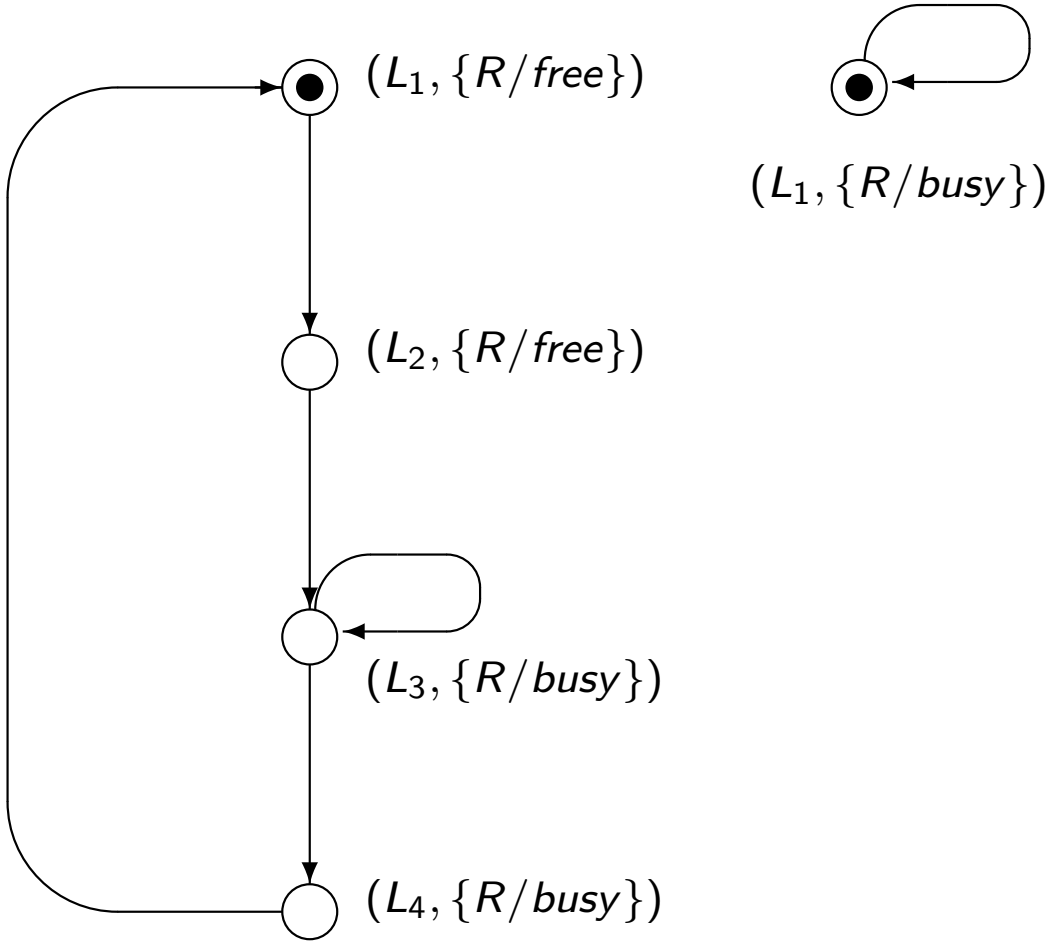
Пример программы

Рассмотрим параллельную композицию этих программ

cobegin
π' : **while** true
do wait (R=free); R:=busy; **skip**; R:=free; **od**
$\|$
π'' : **while** true
do wait (R=free); R:=busy; **skip**; R:=free; **od**
coend

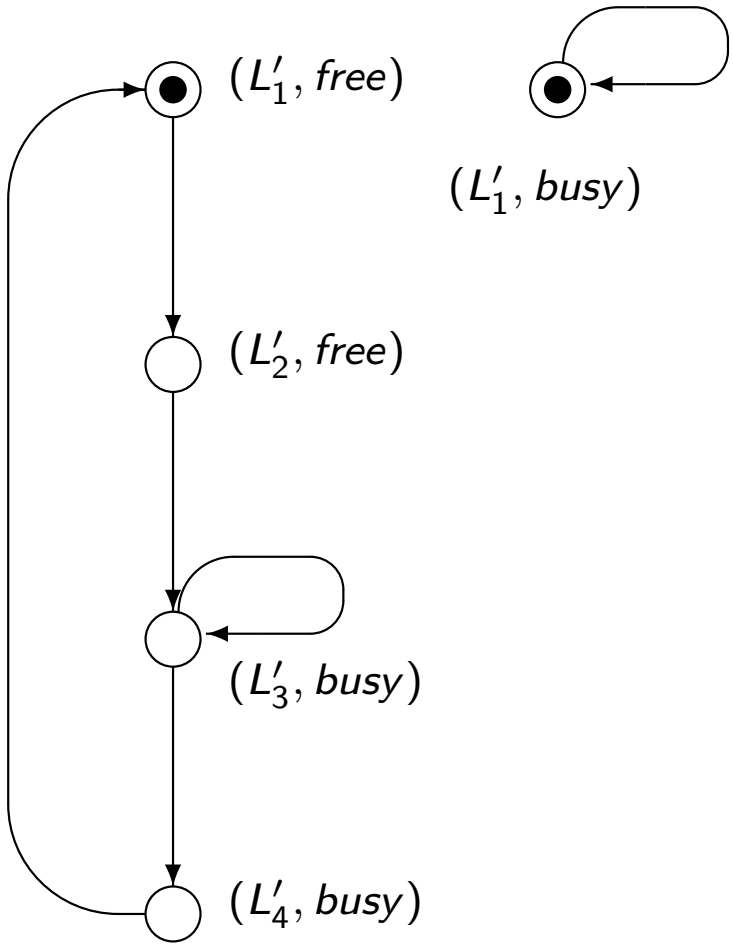
Пример программы

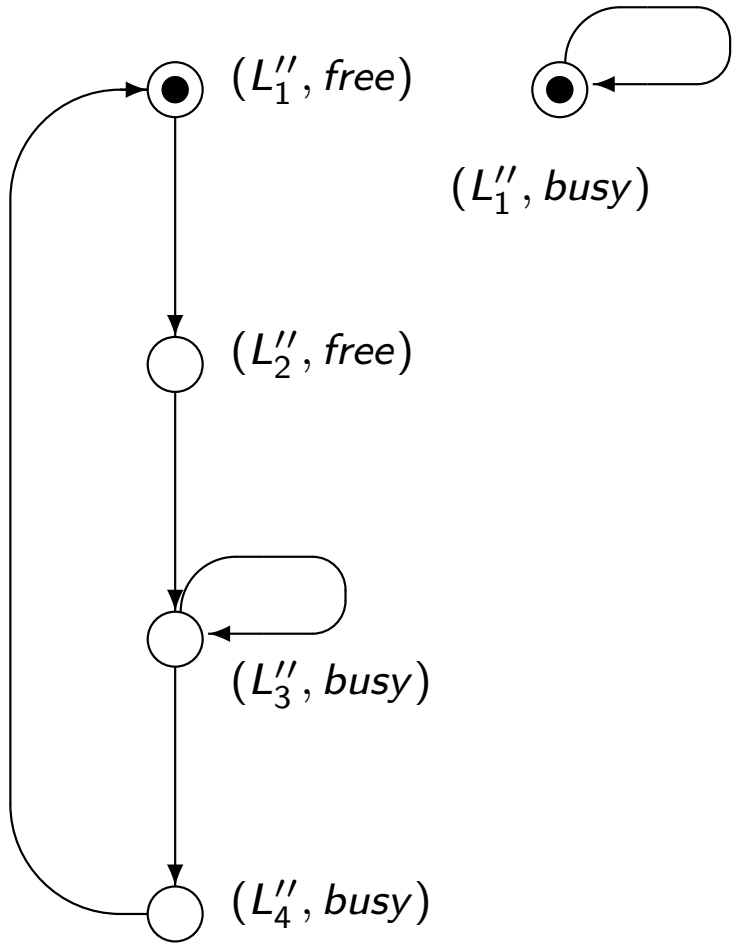
Модель Крипке для программы π'



Пример программы

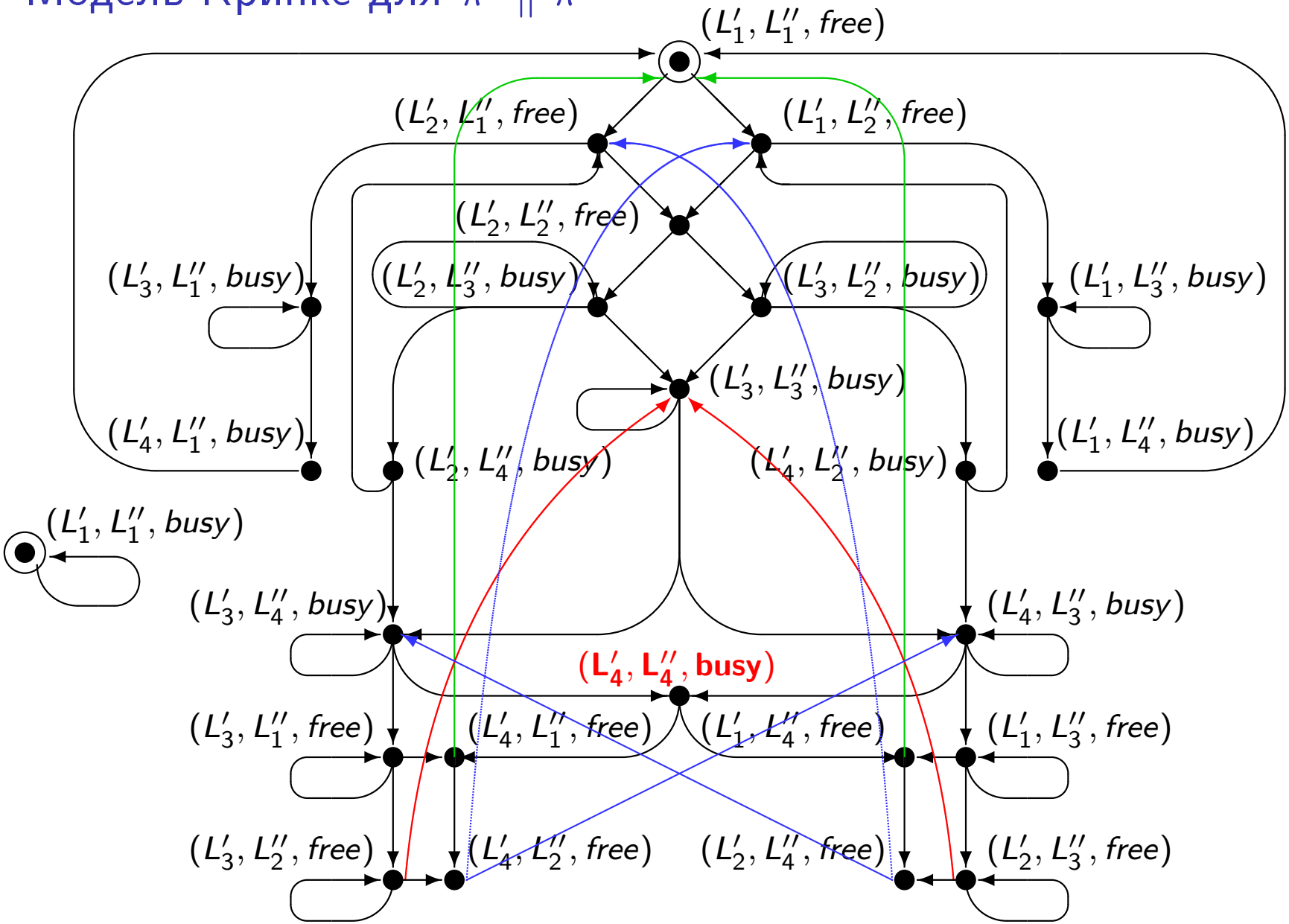
Модель Крипке для программы π' и программы π''





Пример программы

Модель Крипке для $\pi' \parallel \pi''$



Алгоритм Петерсона

А посмотрите на параллельную композицию программ

cobegin
π_1 : **while** true
do $\langle$ b_1:=true; x:=2$\rangle$;
wait (x=1 $\vee$ $\neg b_2$); **skip**; b_1:=false;
od
$\|$
π_2 : **while** true
do $\langle$ b_2:=true; x:=1$\rangle$;
wait (x=2 $\vee$ $\neg b_1$); **skip**; b_2:=false;
od
coend

Алгоритм Петерсона

А посмотрите на параллельную композицию программ

cobegin
π_1 : **while** true
do $\langle$ b_1:=true; x:=2$\rangle$;
wait (x=1 $\vee$ $\neg b_2$); **skip**; b_1:=false;
od
$\|$
π_2 : **while** true
do $\langle$ b_2:=true; x:=1$\rangle$;
wait (x=2 $\vee$ $\neg b_1$); **skip**; b_2:=false;
od
coend

А могут ли здесь два процесса одновременно оказаться в критической секции?

Пример программы

А как сформулировать требования правильности?

Пример программы

Задача об обедающих философах

КОНЕЦ ЛЕКЦИИ 3.