

Математические методы верификации схем и программ

mk.cs.msu.ru → Лекционные курсы
→ Математические методы верификации схем и программ

Блок 9

Особенности моделирования систем

Лектор:

Подымов Владислав Васильевич

E-mail:

valdus@yandex.ru

ВМК МГУ, 2023/2024, осенний семестр

Моделирование программ

Рассмотрим **императивную программу** π , выполняющуюся в интерпретации \mathcal{I} на произвольном состоянии данных множества X

Модель $M_{\pi, \mathcal{I}, X} = (S, S_0, \rightarrow, L)$, отвечающая такому выполнению, может быть устроена так:

- ▶ S — это множество всех **состояний вычисления** программы
- ▶ $S_0 = \{\langle \pi \mid \sigma \rangle \mid \sigma \in X\}$
- ▶ \rightarrow — **отношение переходов программы**, в которое добавлены всевозможные пары вида $\langle \emptyset \mid \sigma \rangle \rightarrow \langle \emptyset \mid \sigma \rangle$
- ▶ AP — всевозможные связки x/d оценок переменных программы
- ▶ $x/d \in L(\langle \pi' \mid \sigma \rangle) \Leftrightarrow \sigma(x) = d$

Пути в такой модели Крипке являются **трассы программы** в \mathcal{I} на σ , $\sigma \in X$

Для других видов программ модель можно устроить точно так же, если определена **операционная семантика**

Моделирование программ

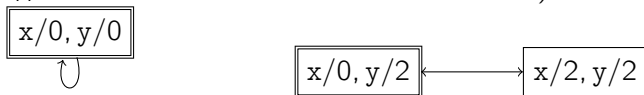
Для примера рассмотрим программу, в которой бесконечно (в цикле) выполняется присваивание

$$x := x + y;$$

в модели императивных программ с интерпретацией, задающей арифметику двухразрядных чисел с переполнением

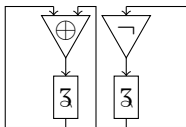
Пусть известно, что в начале работы программы $x = 0$, а значение y может быть любым

Тогда некоторые компоненты связности соответствующей модели Крипке устроены так (можете представить себе и остальные по аналогии):



Моделирование схем

(кто знает термин «последовательная схема» — можете представить её вместо схемы из функциональных элементов с задержкой, СФЭЗ)



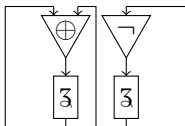
Модель $M = (S, S_0, \rightarrow, L)$, отвечающая СФЭЗ

(последовательной схеме), может быть устроена так:

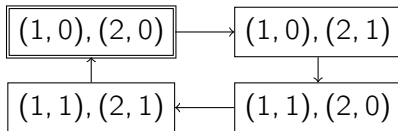
- ▶ Все элементы задержки пронумерованы: $1, 2, \dots, n$
- ▶ $S = \{0, 1\}^n$ (все состояния схемы)
- ▶ $S_0 = \{(0, 0, \dots, 0)\}$ (состояние схемы после сброса)
- ▶ $s \rightarrow s' \Leftrightarrow$ при переходе к следующему моменту времени (по переднему фронту тактового сигнала) возможна такая смена состояния схемы
- ▶ $AP = \{1, 2, \dots, n\} \times \{0, 1\}$ (номер и состояние регистра)
- ▶ $(i, b) \in L(b_0, b_1, \dots, b_n) \Leftrightarrow b_i = b$

Моделирование схем

(кто знает термин «последовательная схема» — можете представить её вместо схемы из функциональных элементов с задержкой, СФЭЗ)



Например, модель Крипке для этой схемы может быть устроена так:



Моделирование параллелизма

Современные вычислительные системы зачастую состоят из набора компонентов, исполняющихся одновременно (параллельно) и взаимодействующих друг с другом

В зависимости от природы системы, при построении модели используется один из двух видов параллелизма (или их комбинация):

- ▶ **Асинхронное исполнение** (чередующееся исполнение; семантика чередующихся вычислений; *interleaving*): шаг вычисления системы отвечает одному шагу выполнения **одного** компонента, а остальные компоненты не выполняются
- ▶ **Синхронное исполнение**: шаг вычисления системы отвечает одновременному выполнению шага вычисления **всех** компонентов

Моделирование параллелизма

Параллельная композиция моделей Крипке $M = (S, S_0, \rightarrow, L)$ и $M' = (S', S'_0, \mapsto, L')$ над непересекающимися множествами атомарных высказываний — это модель $M|M' = (S \times S', S_0 \times S'_0, \rightsquigarrow, \mathcal{L})$, где:

- ▶ $\mathcal{L}(s, s') = L(s) \cup L'(s')$
- ▶ Отношение переходов \rightsquigarrow определяется видом параллелизма

Синхронное исполнение характерно для аппаратных систем, и других имеющих встроенные средства синхронизации компонентов

Переходы синхронной композиции моделей

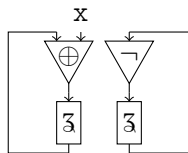
(без взаимодействия компонентов) определяются так:

$$(s_1, s'_1) \rightsquigarrow (s_2, s'_2) \Leftrightarrow s_1 \rightarrow s_2 \text{ и } s'_1 \mapsto s'_2$$

Моделирование параллелизма

$$M = (S, S_0, \rightarrow, L) \quad M' = (S', S'_0, \mapsto, L') \quad M|M' = (S \times S', S_0 \times S'_0, \rightsquigarrow, \mathcal{L})$$

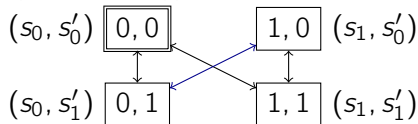
Например:



Модели Крипке, описывающие поведение левой и правой задержек:



Синхронная композиция этих моделей:



Моделирование параллелизма

Асинхронное исполнение характерно для систем без встроенных средств синхронизации компонентов, в том числе (с «примесью» синхронности) для программных систем

Переходы асинхронной композиции моделей (без взаимодействия компонентов) определяются так:

$$(s_1, s'_1) \rightsquigarrow (s_2, s'_2) \Leftrightarrow (s_1 \rightarrow s_2 \text{ и } s'_1 = s'_2) \text{ или } (s_1 = s_2 \text{ и } s'_1 \mapsto s'_2)$$

Моделирование параллелизма

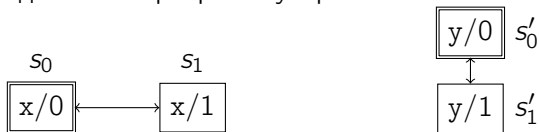
Для примера рассмотрим две параллельно работающие программы, в цикле выполняющие одно присваивание:

$$\pi_1 : x := x + 1;$$

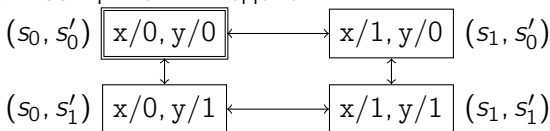
$$\pi_2 : y := y + 1;$$

Для простоты будем считать, что эти программы выполняются в условиях арифметики одноразрядных чисел с переполнением

Модели Крипке для этих программ устроены так:

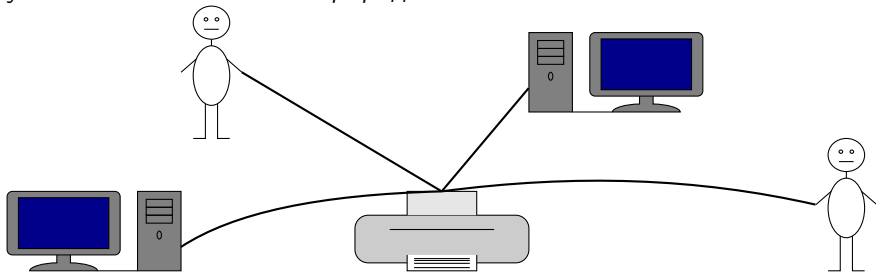


Асинхронная композиция этих моделей:



Моделирование взаимодействия

Пример: с сетевым принтером пытаются взаимодействовать участники *неизвестной природы*



Принтер работает последовательно: принимает информацию и производит печать согласно содержащейся в нём *программе*

Программы остальных участников, *если они есть*, неизвестны

Моделирование взаимодействия

Предположим, что в контроллере принтера есть однокбитовый регистр R , доступный на чтение и запись всем желающим послать запрос на печать:

$$R = \text{t} \Leftrightarrow \text{принтер свободен для печати}$$

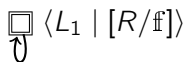
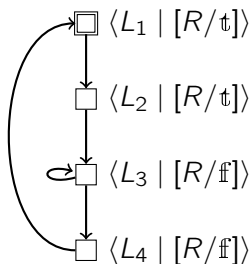
Тогда программу π , посредством которой можно организовать взаимодействие участника с принтером, можно устроить так:

```
while t do  
  L1 : while  $\neg R$  do  $\emptyset$  od  
  L2 :  $R := \text{f}$ ;  
  L3 : послать данные для печати  
  L4 :  $R := \text{t}$ ;  
od
```

Моделирование взаимодействия

Модель Крипке для π :

(функция разметки опущена)



Моделирование взаимодействия

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

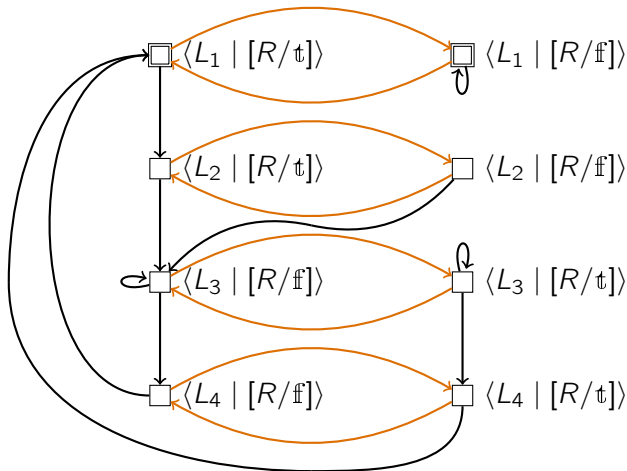
Такое взаимодействие выражается в том, что состояние вычисления программы может измениться под воздействием её **окружения**

Например, регистр R в примере может быть изменён любым участником

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

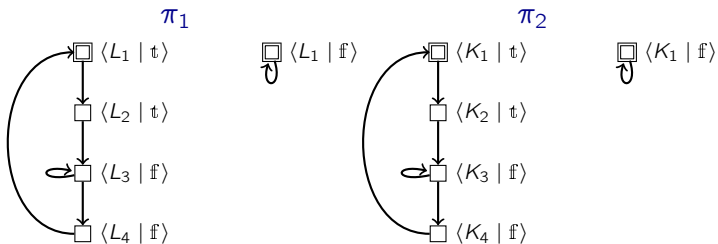
Моделирование взаимодействия

Модель Крипке для программы π с окружением, способным произвольно переключать значение регистра R:



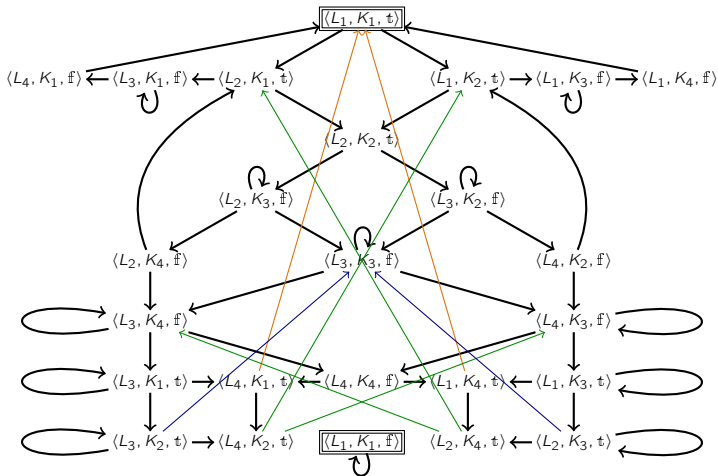
Моделирование взаимодействия

Рассмотрим две (одинаковые) программы взаимодействия с сетевым принтером, выполняющиеся согласно следующим моделям Крипке:



Моделирование взаимодействия

Модель Крипке, описывающая асинхронное исполнение π_1 и π_2 с общим регистром R:



Гранулярность и атомарность в моделях

Переход t в модели отвечает выполнению

атомарного действия системы:

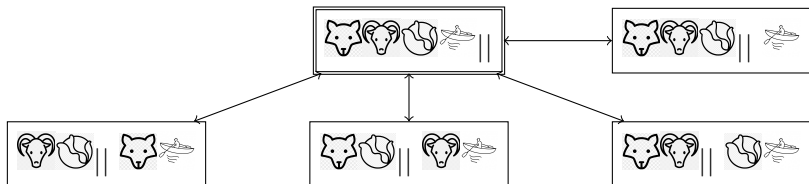
- ▶ в выполнение t не могут вмешаться другие действия
- ▶ t невозможно или неразумно разделять на более простые действия
- ▶ при выполнении t не наблюдаются промежуточные состояния

Выбор **гранулярности** действий в модели: того, какие именно (насколько детальные или абстрактные) действия будут считаться атомарными — играет важную роль при разработке модели:

- ▶ Если действия модели слишком абстрактны, то в модели могут отсутствовать некоторые ошибки, наблюдающиеся при частичном выполнении и «перекрытии» реальных действий
- ▶ Если действия модели слишком детальны, то это может
 - ▶ существенно увеличить размер модели за счёт несуществующих или «неважных» состояний и из-за этого
 - ▶ понизить эффективность верификации

Гранулярность и атомарность в моделях

Например,



Нужно ли рассматривать отдельное действие «плавание по реке»?

А «посадка в лодку» и «высадка из лодки»?

Можно ли посчитать атомарным плавание туда и обратно?

Гранулярность и атомарность в моделях

Другой пример

Рассмотрим две параллельно выполняющиеся программы, каждая из которых выполняет одну команду

$$\pi_1 : x := x + y;$$
$$\pi_2 : y := x + y;$$

Устроит ли нас, если эти две команды будут считаться атомарными?

Тогда в вычислении системы на $[x/2, y/3]$ будут достигаться только состояния данных $[x/5, y/3]$, $[x/5, y/8]$, $[x/2, y/5]$ и $[x/7, y/5]$

Но реализация таких присваиваний на языке ассемблера может содержать и более одной команды:

```
load $1, x
```

```
load $2, y
```

```
add $1, $2
```

```
store $1, x
```

```
load $3, x
```

```
load $4, y
```

```
add $3, $4
```

```
store $3, y
```

Если атомарными считать ассемблерные команды, то достижимы и другие состояния данных — например, $[x/5, y/5]$ — что может оказаться нежелательным (ошибкой)