Формальная верификация методом проверки модели

Захаров А.В.

Санкт-Петербургский государственный политехнический университет

10 ноября 2011

Методы обеспечения качества

- формальные инспекции кода
- тестирование
- статический анализ кода
- формальная верификация
  - дедуктивная верификация
  - *проверка моделей*

Реагирующие системы

Классы программ:

- программы преобразования данных
  - функция – преобразование информации
  - при получении результата завершают выполнение
  - примеры – обращение матрицы, сортировка
  - используется дедуктивная верификация
- реагирующие системы (reactive systems)
  - функция – поддержание взаимодействия с окружением
  - выполняются бесконечно
  - примеры – стеки протоколов, драйверы, планировщики
  - используется modelchecking

Идея modelchecking

- проверка соответствия модели системы частичной спецификации
- частичная спецификация – формула темпоральной логики
- является ли темпоральная формула моделью системы

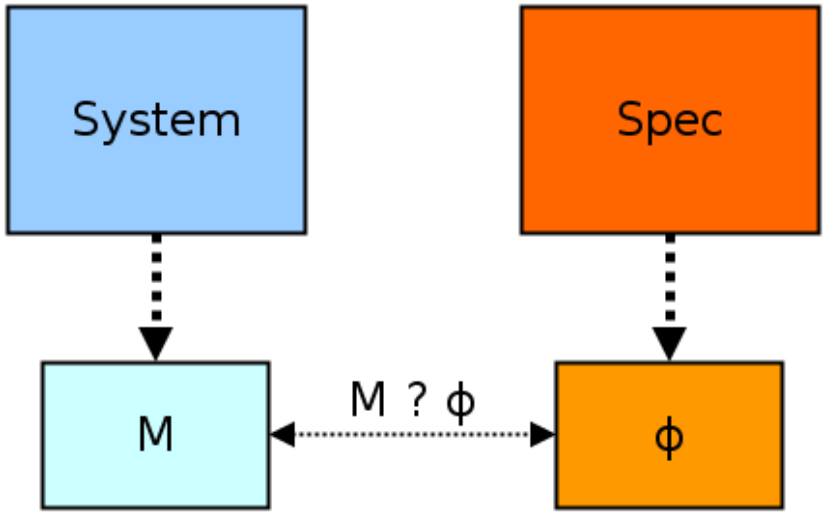


Figure: Проверка модели

Определения

- Предикат - высказывание, которое может быть истинно или ложно в зависимости от аргументов (p, $!p$, $p \vee q$, $p \wedge q$)
- Атомарный предикат - утверждение, которое может быть истинно или ложно и от структуры которого мы абстрагируемся. (p, q)
- Пусть X – множество предикатов, тогда 2^X – множество всех подмножеств множества X (powerset)

Модель Крипке: определение

$M = (S, S_0, R, L)$

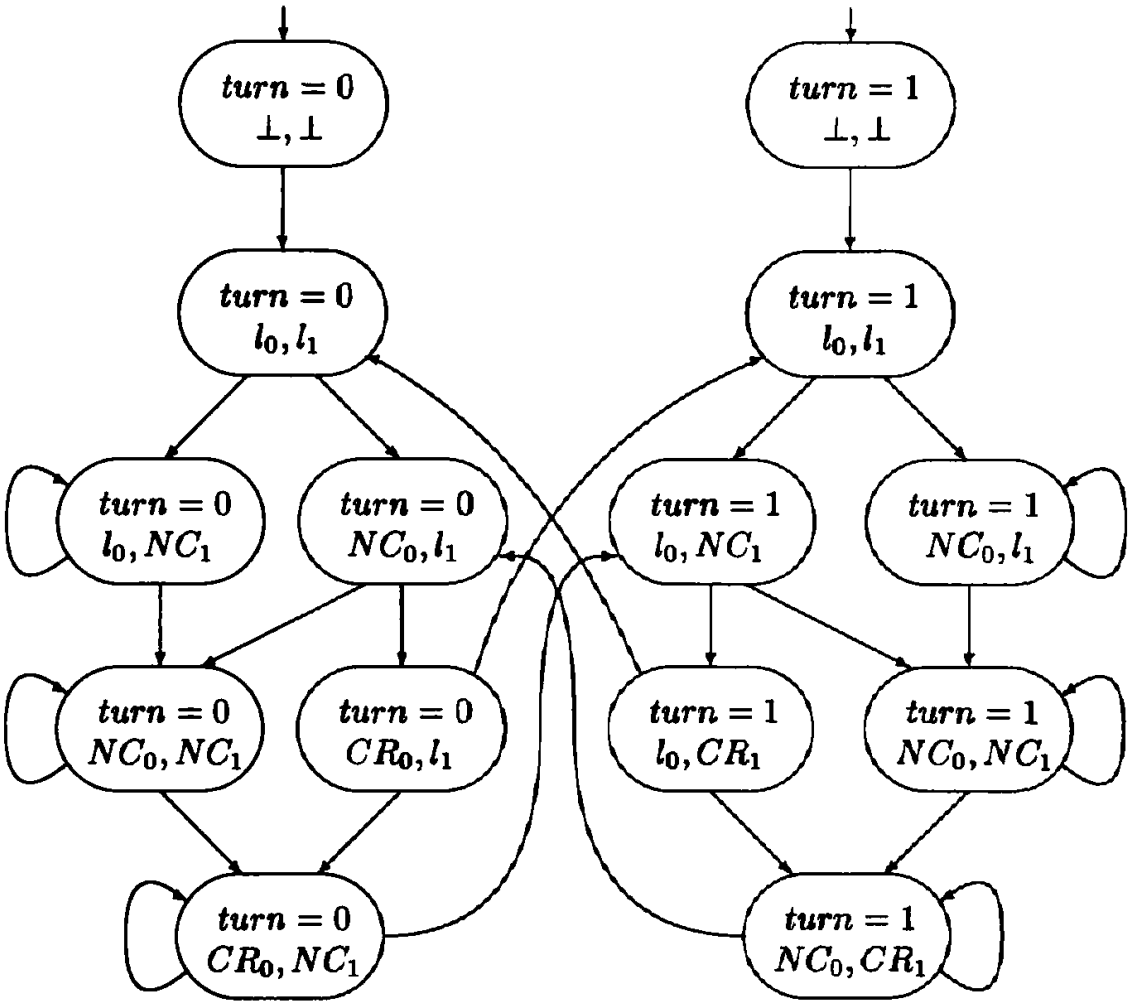
- S – множество состояний
- $S_0 \subseteq S$ – множество начальных состояний
- $R \subseteq S \times S$ – тотальное отношение переходов
  $\forall s \in S \quad \exists s' \quad (s, s') \in R$
- $L : S \to 2^{AP}$ – функция разметки

Пример программы взаимного исключения

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

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

Достижимые состояния модели Крипке



Функция разметки

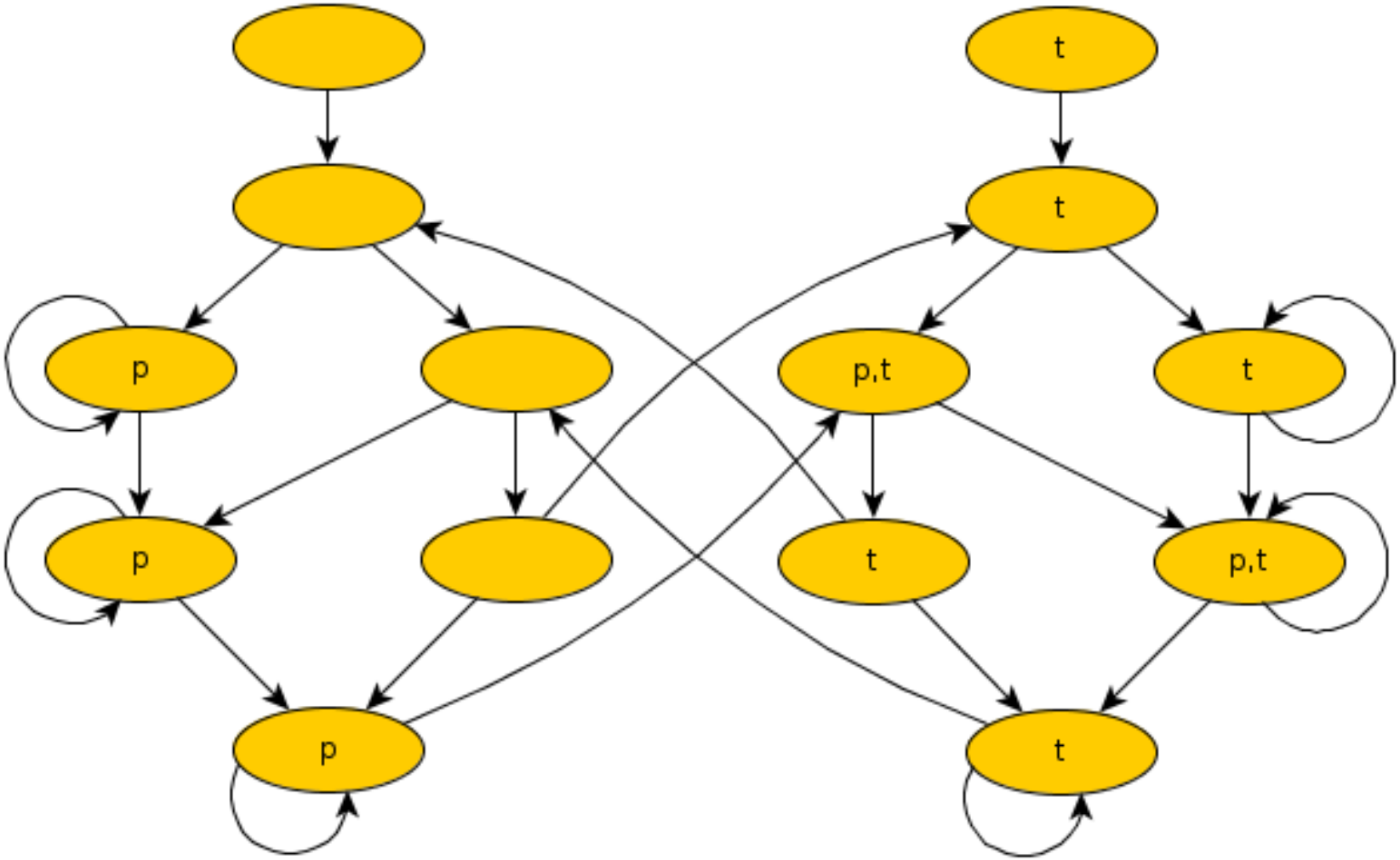
Состояния модели Крипке помечаются атомарными предикатами, которые истинны в данном состоянии.
Рассматриваемые атомарные предикаты

- $turn = 1$ – обозначим t
- $pc_1 = NC_1$ – обозначим p

Разметка состояний

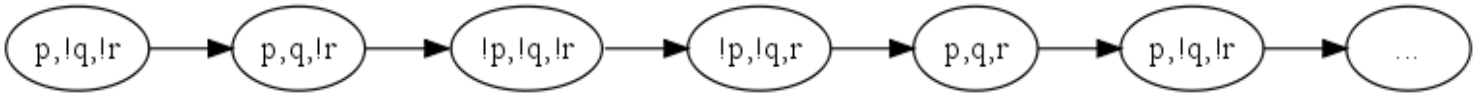
- в состоянии s_0 $(turn = 0, pc_0 = \perp, pc_1 = \perp)$ $L(s_0) = \emptyset$
- в состоянии s_1 $(turn = 1, pc_0 = \perp, pc_1 = \perp)$ $L(s_1) = \{t\}$
- в состоянии s_2 $(turn = 0, pc_0 = l_0, pc_1 = NC_1)$ $L(s_2) = \{p\}$
- в состоянии s_3 $(turn = 1, pc_0 = CR_0, pc_1 = NC_1)$ $L(s_3) = \{t, p\}$

Модель Крипке для взаимодействующих процессов



Темпоральные логики

- Формулы классической логики статичны — истинность не зависит от времени
- Состояние технической системы изменяется со временем
- Темпоральная логика — истинность формул зависит от момента времени, в который вычисляются значения формул



Примеры

- Любое отправленное сообщение рано или поздно будет получено
- Пока ключ зажигания не вставлен, машина не поедет
- Всегда верно, что если производитель остановился, потребитель рано или поздно запустится

Деревья вычислений

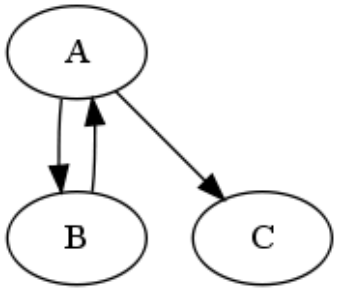


Figure: Система переходов

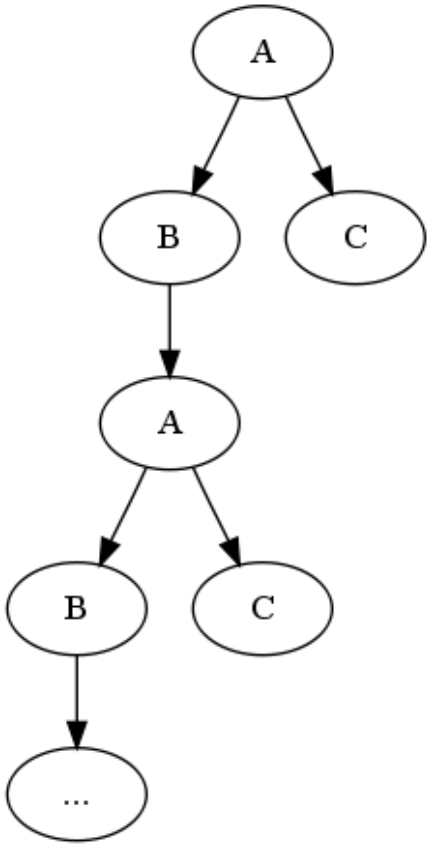


Figure: Дерево вычислений

Виды темпоральных логик

Виды темпоральных логик:

- Линейная темпоральная логика, Linear Time Logic (LTL)
- Темпоральные логики деревьев вычислений, Computation Tree Logic (CTL, CTL*)

Виды темпоральных логик

LTL:

- линейная темпоральная логика
- (не)выполнение свойств *на всех* путях в дереве вычислений

CTL, CTL*:

- темпоральные логики деревьев вычислений
- (не)выполнение свойств *на всех* или *на каком-либо* пути

Формализация LTL: синтаксис

Состав формул:

- атомарные предикаты (Atomic Predicate, AP)
- булевские операторы (&&, ||, !)
- темпоральные операторы
  - **G** (или []) – Globally, «всегда», «повсюду»
  - **F** (или <>) – Future, «рано или поздно», «когда-то в будущем»
  - **X** – neXttime, «в следующий момент времени»
  - **U** – Until, «до тех пор пока»

Формализация LTL: синтаксис

```
f ::= p
    | true
    | false
    | (f)
    | f binop g
    | unop f
```

```
unop ::= G
       | F
       | X
       | !

binop ::= U
        | &&
        | ||
        | ->
        | <->
```

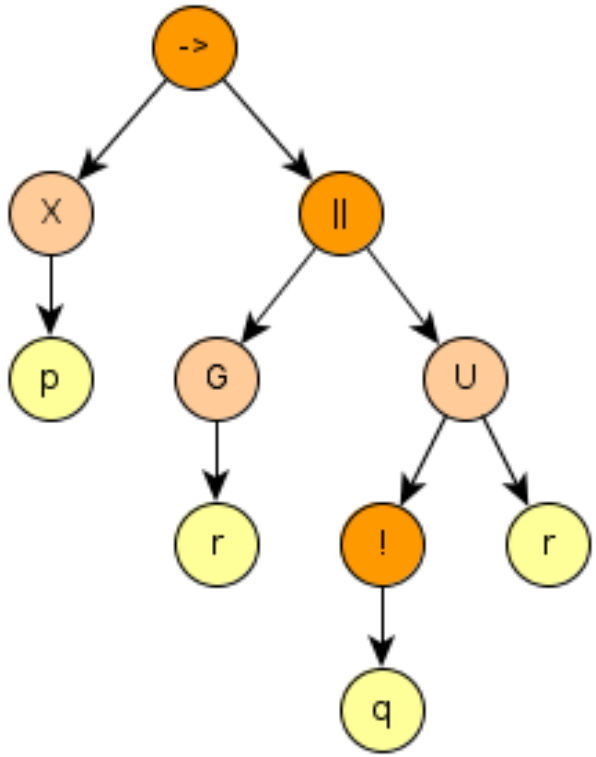
Формализация LTL: приоритет операций

Приоритет операций

- `true, false`
- `!, X, G, F`
- `U`
- `&&, ||`
- `->`
- `<->`

Пример

```
X p -> G r || ! q U r
```



Формализация LTL: семантика

Обозначения

- Модель Крипке $M =< S, S_0, R, L >$
- Бесконечный путь в модели Крипке: $\pi =< s_0, s_1, s_2, ... >$, где $s_i \in S$
- Суффикс π^i пути π: $\pi^i =< s_i, s_{i+1}, s_{i+2}, ... >$
- Формула f выполняется на пути π модели Крипке M: $M, \pi \models f$

Формализация LTL: семантика

Семантика формул LTL

- $M, \pi \models p \iff p \in L(s_0)$
- $M, \pi \models \neg p \iff M, \pi \not\models p$
- $M, \pi \models \phi_1 \vee \phi_2 \iff$
  - $M, \pi \models \phi_1$, or
  - $M, \pi \models \phi_2$
- $M, \pi \models X(\phi) \iff |\pi| > 1$ and $M, \pi^1 \models \phi$
- $M, \pi \models F(\phi) \iff \exists k : 0 \leq k < |\pi|$ and $M, \pi^k \models \phi$
- $M, \pi \models G(\phi) \iff \forall k : 0 \leq k < |\pi|$ and $M, \pi^k \models \phi$
- $M, \pi \models \phi U \psi \iff$
  - $\exists k : 0 \leq k < |\pi|, M, \pi^k \models \psi$, and
  - $\forall i : 0 \leq i < k, \pi^i \models \phi$

Семантика LTL. Примеры

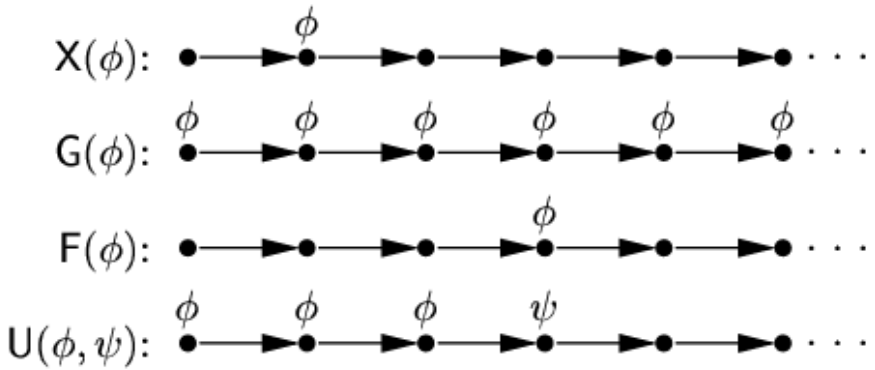


Figure: Примеры путей, на которых выполняются формулы

Основные типы свойств

- безопасность, safety – "something bad will not happen"
- живость, liveness – "something good will happen"
- справедливость, fairness – "if something is attempted/requested infinitely often, then it will be successful/allocated infinitely often"

Неоднозначность формализации свойств

После начала лекции она будет прочитана до конца (в середине лекции должен быть перерыв)

Неоднозначность формализации свойств

После начала лекции она будет прочитана до конца (в середине лекции должен быть перерыв)

- $start \rightarrow (F(break \wedge F\ finish))$

Неоднозначность формализации свойств

После начала лекции она будет прочитана до конца (в середине лекции должен быть перерыв)

- $start \rightarrow (F(break \wedge F\ finish))$
- $start \rightarrow teaching\ U\ finish$

Неоднозначность формализации свойств

После начала лекции она будет прочитана до конца (в середине лекции должен быть перерыв)

- $start \rightarrow (F(break \wedge F\ finish))$
- $start \rightarrow teaching\ U\ finish$
- $start \rightarrow (teaching\ U\ (break\ U\ (teaching\ U\ finish)))$

Что делать?!

- использовать шаблоны спецификации – temporal specification patterns
  - http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml
- проверять формулу на конкретных путях
  - вручную или автоматически
- сравнивать разные варианты формализации
  - mcheck

Исходные идеи для алгоритма верификации LTL

- LTL-формула описывает множество путей, удовлетворяющих этой формуле
- Представим, что путь – это некоторая бесконечная входная последовательность
- Конечная входная последовательность может быть распознана конечным автоматом
- Бесконечная входная последовательность может быть распознана автоматом Бюхи
- Формула ϕ выполняется на модели M, если
  - пересечение языков автоматов для модели и для отрицания формулы пусто
  - $L_M \cap L_{\neg\phi} = \emptyset$

Конечный автомат над конечными словами

Конечный автомат над конечными словами $A =< \Sigma, Q, \Delta, Q^0, F >$, где

- Σ – конечный алфавит
- Q – конечное множество состояний
- $\Delta \subseteq Q \times \Sigma \times Q$ – отношение переходов
- $Q^0 \subseteq Q$ – множество начальных состояний
- $F \subseteq Q$ - множество заключительных состояний

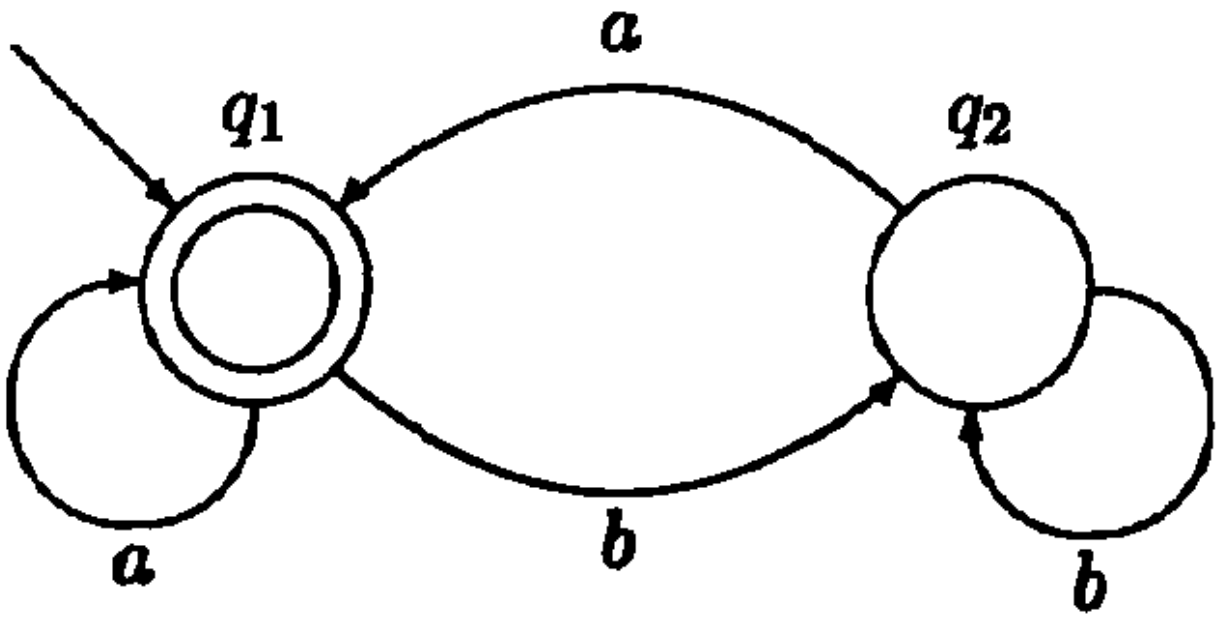
Определения

- Проход автомата A над $v \in \Sigma*$ – отображение $\rho : \{0, 1, ..., |v|\} \to Q$
  - $\rho(0) \in Q^0$
  - $(\rho(i), v(i), \rho(i+1)) \in \Delta$
- Допускающий проход – $\rho(|v|) \in F$
- Автомат A допускает v тогда и только тогда, когда существует проход A на v
- Язык $L(A) \subseteq \Sigma*$ – множество слов, которые допускает A

Конечный автомат над бесконечными словами

- Реагирующие системы выполняются бесконечно
- Формализация таких систем – автомат Бюхи
- Автомат Бюхи $A =< \Sigma, Q, \Delta, Q^0, F >$
  - F – допускающее, а не заключительное
- Допускающий проход над бесконечными словами
  - $inf(\rho) \cap F \neq \emptyset$, где
  - $inf(\rho)$ – множество состояний, которые встечаются в ρ бесконечно часто

Пример автомата Бюхи



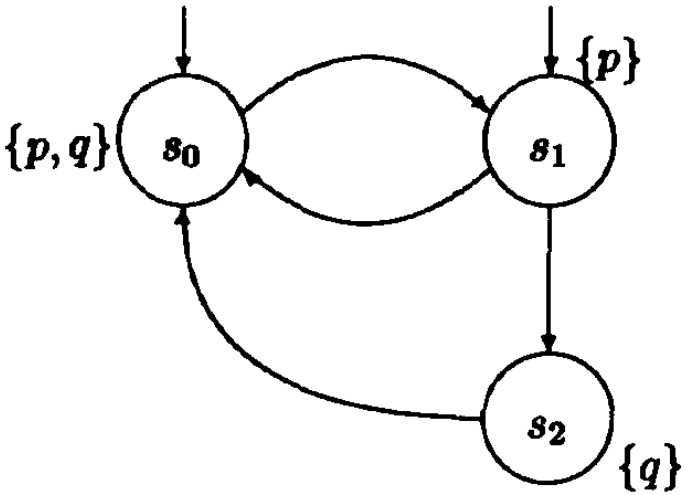
Преобразование модели Крипке в автомат Бюхи

Модель Крипке $M =< S, S_0, R, L >$ преобразуется в автомат Бюхи $A =< \Sigma, S \cup \{i\}, \Delta, \{i\}, S \cup \{i\} >$, где

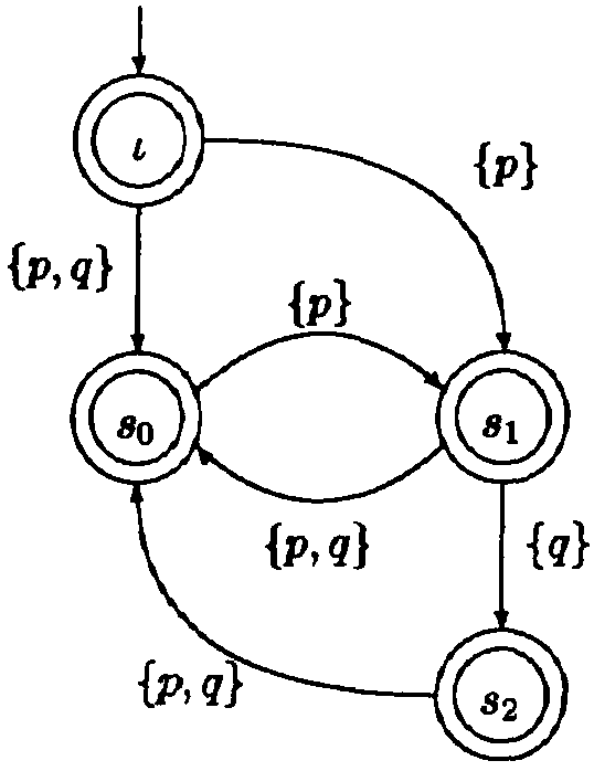
- $\Sigma = 2^{AP}$
- i – новое начальное состояние
- $(s, \alpha, s') \in \Delta \Leftrightarrow (s, s') \in R \wedge \alpha = L(s')$
- $(i, \alpha, s) \in \Delta \Leftrightarrow s \in S_0 \wedge \alpha = L(s)$

Получаемый автомат может быть недетерминированным

Преобразование модели Крипке в автомат Бюхи



Модель Крипке



Автомат

Языки автоматов и поведение

- Система соответствует спецификации, если множество всех поведений системы входит в множество допустимых поведений
- Спецификация описывает множество допустимых поведений
- Поведение системы может быть описано автоматом Бюхи
- Система соответствует спецификации, если $L(A) \subseteq L(S)$
- или
- Система соответствует спецификации, если $L(A) \cap \overline{L(S)} = \emptyset$

Описание недопустимых поведений

Дополнение языка спецификаций

$$\overline{L(S)} = \Sigma^{\omega} \setminus L(S)$$

Автомат $\overline{S}$

- автомат, распознающий язык $\overline{L(S)}$
- явно задан пользователем
- получен на основе языка темпоральных формул LTL

Трансляция LTL в автомат Бюхи

Используется алгоритм Герта, Пеледа, Варди и Волпера.

Пересечение языков

- Автоматы Бюхи замкнуты относительно операций пересечения и дополнения
- Для распознавания пересечения языков двух автоматов можно построить автомат Бюхи

Автомат Бюхи для пересечения языков

Даны автоматы Бюхи B_1 и B_2 над алфавитом Σ:

- $B_1 =< \Sigma, Q_1, \Delta_1, Q_1^0, F_1 >$
- $B_2 =< \Sigma, Q_2, \Delta_2, Q_2^0, F_2 >$
- распознают языки $L(B_1)$ и $L(B_2)$

Требуется построить автомат $B_1 \cap B_2$, допускающий язык $L(B_1) \cap L(B_2)$

Автомат Бюхи для пересечения языков

Автомат

$$B_1 \cap B_2 =< \Sigma, Q_1 \times Q_2 \times \{0,1,2\}, \Delta, Q_1^0 \times Q_2^0 \times \{0\}, Q_1 \times Q_2 \times \{2\} >$$

- $(< r_i, q_j, x >, a, < r_m, q_n, y >) \in \Delta \iff$
- $(r_i, a, r_m) \in \Delta_1$ и $(q_j, a, q_n) \in \Delta_2$
- x, y подчиняются правилам
  - если $(x = 0) \wedge (r_m \in F_1)$, то $y = 1$, иначе
  - если $(x = 1) \wedge (q_n \in F_2)$, то $y = 2$, иначе
  - если $(x = 2)$, то $y = 0$, иначе
  - $y = x$

Допускающие состояния нового автомата

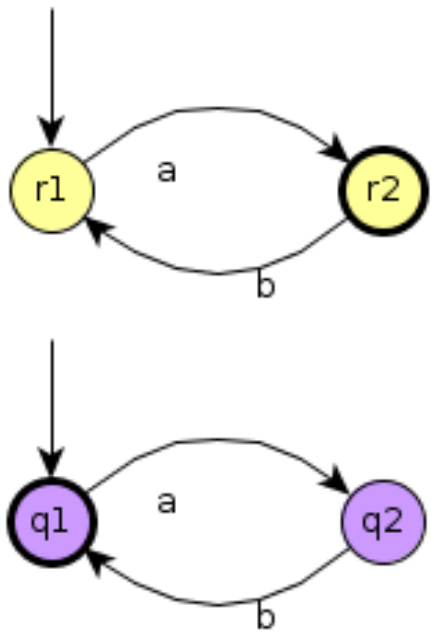
Допускающий проход над бесконечными словами

- $inf(\rho) \cap F \neq \emptyset$, где
- $inf(\rho)$ – множество состояний, которые встечаются в ρ бесконечно часто

Почему используются *такие* допускающие состояния?

- $Q_1 \times Q_2 \times \{2\}$, а не
- $F_1 \times F_2$

Пример с допускающими состояниями



Пример с допускающими состояниями

Попробуем второй вариант...

- $A =< \Sigma, Q_1 \times Q_2, \Delta, Q_1^0 \times Q_2^0, F_1 \times F_2 >$

Пример с допускающими состояниями

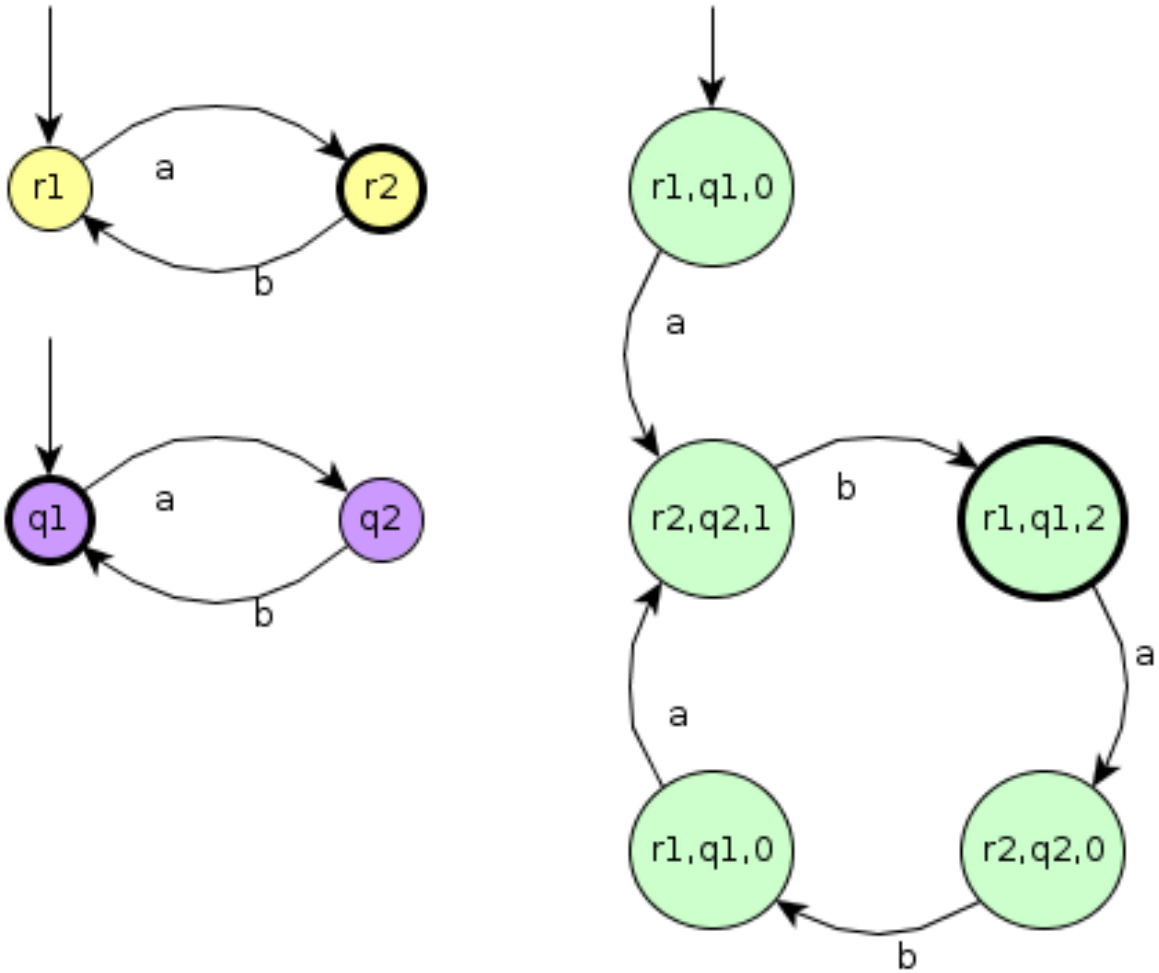
Второй вариант

- $A =< \Sigma, Q_1 \times Q_2, \Delta, Q_1^0 \times Q_2^0, F_1 \times F_2 >$

Автомат A ничего не принимает

$L(A) = \emptyset$

Автомат для пересечения языков



Алгоритм проверки пустоты языка

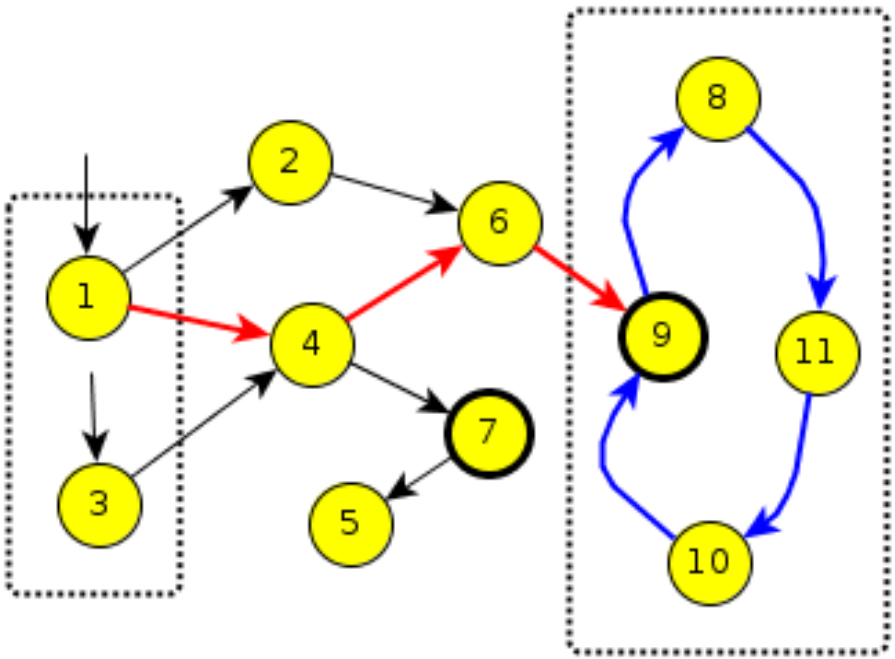
- Допускающий проход ρ: $inf(\rho) \cap F \neq \emptyset$
- Множество Q конечно
- Есть суффикс ρ': $\forall s \in \rho' inf(\rho') \cap s \neq \emptyset$
- Этот суффикс является сильносвязной компонентой графа переходов

Алгоритм проверки пустоты

Идея алгоритма

- найти все пути из всех начальных состояний во все допустимые состояния
- найти циклы, содержащие найденные допустимые состояния
- в результате, либо докажем, что нет такого прохода, либо предъявим контр-пример

Иллюстрация



Поиск допускающего состояния

```
for all q_0 ∈ Q^0 do
        dfs1(q_0);
terminate(False)
```

Поиск допускающего состояния

```
procedure dfs1(q)
        add(statespace, {q, 0})
        push(stack1, q)
        for all q' ∈ next(q) do
                if {q', 0} ∉ statespace
                        dfs1(q')
        if accept(q)
                dfs2(q)
        pop(stack1)
```

Поиск цикла

```
procedure dfs2(q)
        add(statespace, {q, 1})
        push(stack2, q)
        for all q' ∈ next(q) do
                if q' ∈ stack1
                        terminate(True)
                else if {q', 1} ∉ statespace
                        dfs2(q')
        pop(stack2)
```

Результат алгоритма

- *False* – язык автомата пуст
- *True* – язык не пуст, есть *контрпример*
  - *stack*1 содержит конечный префикс контр-примера
  - *stack*2 содержит путь из конечной вершины q_1 префикса в некоторую вершину q_2 префикса
  - из q_2 есть путь в q_1 (часть префикса из *stack*1)

Вычислительная сложность

Если автомат A имеет n состояний

- алгоритм поиска допускающего состояния линеен $-O(n)$
- алгоритм поиска ССК также линеен $O(n)$
- алгоритм проверки пустоты языка имеет сложность $O(n^2)$

Верификация "на-лету"

- можно не строить оба автомата A и S полностью
- достаточно иметь полностью автомат S и выбирать из автомата A только те состояния, переходы которых согласованы с переходами из текущего состояния S
- если язык пуст, то проделаем существенно меньше операций

Заключение

- Линейная темпоральная логика позволяет специфицировать поведение систем с *конечным числом состояний*
- Автоматы Бюхи позволяют распознавать бесконечные входные последовательности
- По темпоральной формуле может быть построен автомат Бюхи
- Проверка выполнимости спецификации S на модели A формулируется как $L(A) \cap \overline{L(S)} = \emptyset$
- Существует способ построения автомата Бюхи для $L(A) \cap L(B)$
- Рассмотрен алгоритм проверки пустоты языка, распознаваемого автоматом Бюхи

Источники

- Верификация моделей программ: Model Checking [Текст] / Кларк Э.М. Мл., Грамберг О., Пелед Д. - М. : МЦНМО, 2002. - 416 с. - ISBN 5-94057-054-2
- Ю.Г. Карпов Model Checking. Верификация параллельных и распределенных программных систем [Текст]. - СПб.: БХВ-Петербург, 2010. - 552 с. - ISBN 978-5-9775-0404-1
- G. Holzmann and D. Peled, M. Yannakakis On Nested Depth First Search // In The Spin Verification System. – 1996. – pp 23-32