Модели на софтуерни системи

доц. Олга Георгиева

СУ ФМИ

катедра “Софтуерни технологии”

Модели на софтуерни системи

Лекция 7: Машини на състоянието - варианти

Олга Георгиева

СУ, Факултет по Математика и информатика,
Катедра СТ

Развитие

- Състоянията могат да бъдат огромен брой дори и за много проста система;
- Често се налага по-компактно представяне на състоянията:
  - Да се опишат **като предикат** множеството от състояния, от които съществува преход;
  - Да се опише “целта” чрез промените на източника;
  - Използване **на текст**, а не илюстрации, с цел подпомагане на етапа на кодиране;
  - Фокусиране върху **специални аспекти** (като пътеки на събития).

Въведение

- За машината на състоянието **M = (S, I, A, δ)** могат да се разглеждат различни **варианти**, реализирани чрез:

  А) структуриране на състоянията **S** ;

  Б) структуриране на действията **A** ;

  В) обобщено описание на релацията/функцията на преходите δ ;

  Г) съвместно използване на случаите А), Б), В).

  Д) допълнителни случаи на пречистване, използвани в практиката.

- **Изборът на модел** зависи от:

  > Какво ще моделираме

  > Изисквания за *точност* и *простота* на описанието.

  > Понякога: ... и въпрос на вкус!

  Но винаги трябва да знаем защо сме направили съответния избор!

Примери!

А) Състоянията като функция

Всяко състояние на $\boldsymbol{M}$ е крайна функция от

крайно множество $\boldsymbol{Var}$ от променливи (“identifiers”, “names”) към (вероятно) *безкрайно* множество $\boldsymbol{Val}$ от *типови* стойности (крайна функция).

$$M = (\{S : Var \nrightarrow Val\}, I, A, \delta)$$

Пр.:Множеството от състояния на брояч (*примерът от миналата лекция*) е тотална функция:

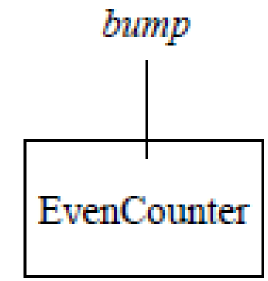
$$\{s : \{x\} \to \mathrm{int} \mid s(x) \geq 0\}$$

с функция на преходите:

$$\delta_{inc} = \{(s, a, s') : S \times \{inc\} \times S \mid s'(x) = s(x) + 1\}$$

Пример: Състоянията като функция

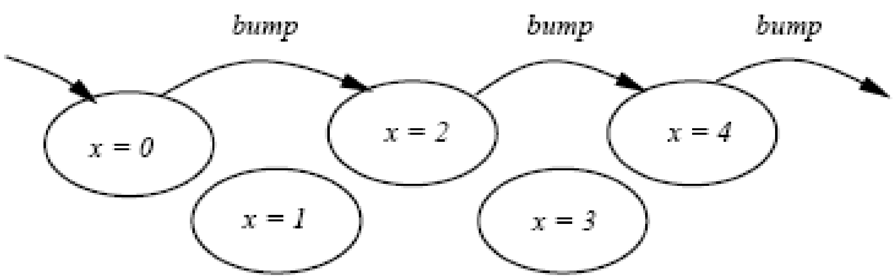
Задача: За брояч на четните положителни числа:



EvenCounter's Interface

1. Съставете граф на преходите
   (с етикет на действието "*bump*").
2. Дефинирайте **множеството на състоянията като функция**.
3. Дефинирайте функцията на преходите δ_{bump} .

Отг.:



Part of EvenCounter's State Transition Diagram

$$\{s : \{x\} \rightarrow \mathrm{int} \mid \; s(x) \geq 0 \cdots ?\}$$

$$\delta_{bump} = \{(s, a, s') : S \times \{bump\} \times S \mid \; even(s(x)) \wedge s'(x) = s(x) + 2\}$$

- Предикатът ***even*** е дефиниран предварително
- Недостижими състояния

Състоянията като функция

Функцията на преходите

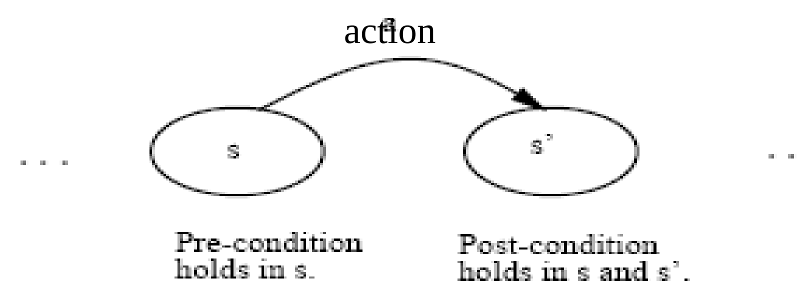
$$\delta_{action} = \{(s,a,s'): S \times \{action\} \times S \mid \ \Phi[s(v)/v]) \wedge \Psi[s'(v)/v', s(v)/v]\}$$

може да бъде представена по-кратко чрез **нотация “тип програмиране**”:

- ***pre-post*** спецификация:

За машината ***M* = (*S*, *I*, *A*, δ):** ***action***
pre Φ**(** v **)**
post Ψ**(** v, v' **)**

action (**header**) $\in$ ***A***, а Φ и Ψ са (**pre** and **post**) предикати върху вектора ***v*** на променливите на състоянието.



Импликация или конюнкция?

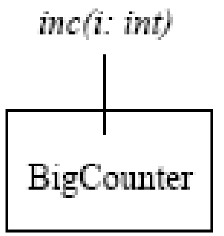
Зад. Дефинирайте действията *inc* и *bump* на двата брояча.

Б) Действия с аргументи

Нека броячът (BigCounter) да дефинира преходи

в резултат на действие ***inc*** , която е дефинирана като

функция с аргумент цяло число:



BigCounter's Interface

Много кратко и точно описание чрез ***lambda abstraction*** на ***фамилия* от действия**.

$inc(\ i\!: int)$

pre *true*

post $x' = x+i$

и вариант с допълнително ограничение в pre-condition за *FatCounter* (защо?):

$inc(\ i\!: int)$

pre $i > 0$

post $x' = x+i$

Зад. Представете част от графа на преходите *на FatCounter* с $inc(i\!: int)$ за първите 4 последователни състояния.

Действия с резултати

Допълнителна структура на действията се дефинира, когато резултатът от действията са необходими на външния наблюдател. Преходът на състоянията е *случай* (***action instance***), който се описва чрез **двойка събития**:

а) *извикване (invocation event)*: името на действието и стойностите на неговите входни аргументи;

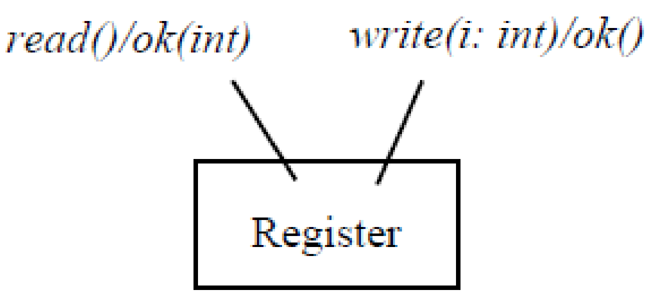
б) *отговор (response event)*: името на условието за приключване и стойността на неговия резултат.

Такава спецификация е удобна в случаите на:

> нормално приключване - ***ok( )*** и

> извънредно (exceptional) приключване;

Пр. Регистър с действия *read* и *write*.



Register's Interface

Пример: Действия с аргументи и с резултати

- Запишете граф на преходите на регистъра за първите три поредни състояния ($x=0$, $x=1$, $x=2$) като началното състояние е $x = 0$.
- Специфицирайте двете действия *read* и *write* :

read( ) / *ok*(*int*)
pre *true*
post *result* = *x*

write(*i*:*int*) / *ok*( )
pre *true*
post *x'* = *i*

- Запишете някои от изпълненията на тази машина:

$$\langle x = 0, \mathit{write}(1)/\mathit{ok}(), x = 1, \mathit{read}()/\mathit{ok}(1), x = 1\rangle$$
$$\langle x = 0, \mathit{write}(1)/\mathit{ok}(), x = 1, \mathit{read}()/\mathit{ok}(1), x = 1, \mathit{read}()/\mathit{ok}(1), x = 1, \mathit{write}(5)/\mathit{ok}(), x = 5, \mathit{read}()/\mathit{ok}(5), x = 5\rangle$$
$$\langle x = 0, \mathit{write}(1)/\mathit{ok}(), x = 1, \mathit{write}(7)/\mathit{ok}(), x = 7, \mathit{write}(9001)/\mathit{ok}(), x = 9001\rangle$$

- За горните изпълнения запишете някои от event-based пътеките:

$$\langle \mathit{write}(1)/\mathit{ok}(), \mathit{read}()/\mathit{ok}(1)\rangle$$
$$\langle \mathit{write}(1)/\mathit{ok}(), \mathit{read}()/\mathit{ok}(1), \mathit{read}()/\mathit{ok}(1), \mathit{write}(5)/\mathit{ok}(), \mathit{read}()/\mathit{ok}(5)\rangle$$
$$\langle \mathit{write}(1)/\mathit{ok}(), \mathit{write}(7)/\mathit{ok}(), \mathit{write}(9001)/\mathit{ok}()\rangle$$

- и state-based пътеки …

Спецификация

- Записва се **резултата** от всяко действие (ако има такъв);
- Дава **типа** (като стойност) на резултата – *int*, …;
- Може да използва специални (запазени) думи (“*result*”, “*even*”, ...)

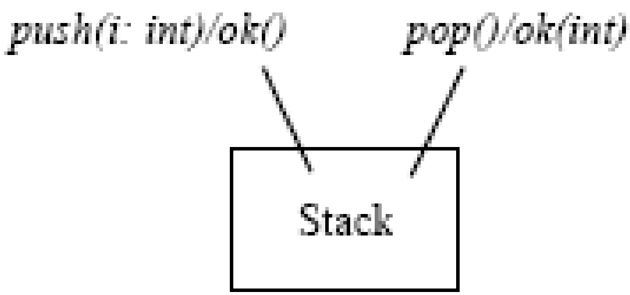
Технически бележки

- съществува разлика (малка) между действие, елемент от крайното м-во на действията *A* и извършване на действието – елемент на м-вото (може би безкрайно) на преходите δ. Това различие е аналогично на деклариране на процедура и извикване на процедура.

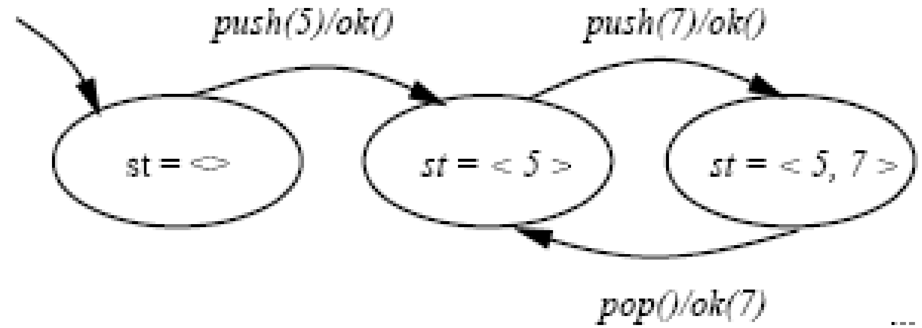
- някои машини третират *invocation event* като входно действие и *response event* като изходно действие - за паралелни и разпределени системи.

Пример: Действия с аргументи и с резултати

Интерфейс с извънредни приключвания: нормални или изключения:



Stack's Interface



Part of Stack's State Transition Diagram

push(*i:int*) / *ok*( )

pre *true*

post $st' = st \;\hat{}\; \langle i \rangle$

или по-робастен вариант:

pop() /*ok*(*int)*

pre $st \neq \langle\,\rangle$

post $st = st' \;\hat{}\; \langle result \rangle$

pop() /*ok*(*int), empty*( )

pre *true*

post $st \neq \langle\,\rangle \Rightarrow (st = st' \;\hat{}\; \langle result \rangle \wedge terminates = ok) \wedge$

$st = \langle\,\rangle \Rightarrow (st = st' \wedge terminates = empty)$

В) Функция на преходите (обобщение) - недетерминизъм

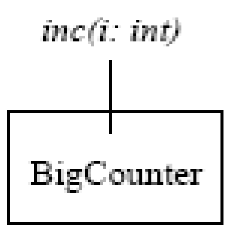
Когато има повече от едно възможно следващо състояние при еднакво действие и еднакво изходно състояние, то функцията на преходите трябва да свързва множество от състояния:

δ (s_k, *action*) = {s_1', s_2',…, s_n'}

Недетерминизмът се представя с **дизюнкция** в post състоянието.

Пр.: Random Counter

1) Представете граф на машината за частта на начално състояние $x = 3$ и действие *inc(4)* при следната спецификация:

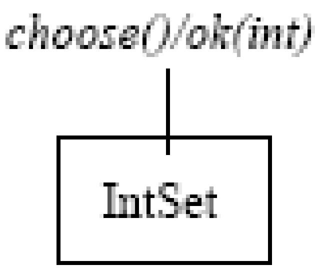


BigCounter's Interface

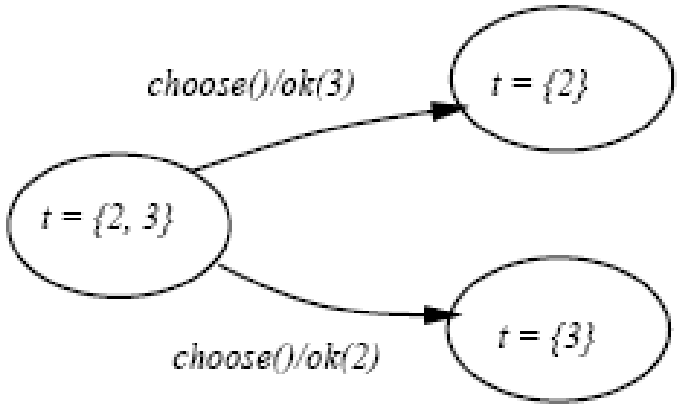
inc(*i*:*inc*)
pre $i > 0$
post $x' = x + i \lor x' = 2i$

Пример: Съвместно използване на разгледаните случаи

Пр. Машината *IntSet* съдържа множество от цели числа *t*. Интерфейсът включва действие *choose* без аргумент и което премахва и връща елемент от *t.*



IntSet's Interface



Part of IntSet's State Transition Diagram

Запишете IntSet в pre-post нотация:

Каква е машината –

детерминистична или недетерминистична?

Г) Съвместно използване на разгледаните случаи

Обобщение: Общият шаблон, прилаган при описване на всяко действие *action* в **A** от **M = (S, I, A, δ),** е:

$$\textbf{\textit{action}}(\ \textbf{\textit{inputs}}\ )/\ \textbf{\textit{term}}_1(\ \textbf{\textit{output}}_1\ ),\ \ldots,\ \textbf{\textit{term}}_n(\ \textbf{\textit{output}}_n\ )$$

pre $\Phi(v)$

post $\Psi(v, v')$

където ***inputs*** е списък на аргументите и техният тип, $\textbf{\textit{term}}_i$ - името на ***i***-тото условие за приключване и $\textbf{\textit{output}}_i$ е типът и резултатът, съответстващ на $\textbf{\textit{term}}_i$.

Φ и Ψ са предикати на състоянието върху вектора $\boldsymbol{v}$.

Използваните в това изложение определители:

- *ok* – използва се в *header* за нормално прекъсване
- *result* – използва се в *post-condition* за върнатата стойност
- *terminates* – използва се в *post-condition* за стойност на условието за приключване. Стойността е указана в *header*.
- *empty*

Други машини на състоянието, често използвани в практиката

- Крайни автомати

Детерминистични крайни автомати (DFSA) $\boldsymbol{M = (S, I, F, A, \delta)}$:

S е **крайно (ограничено)** множество от възможни състояния;

I, I $\subseteq$ **S** е множество от начални състояния (**singleton set**);

F $\subseteq$ **S** е крайно множество от **крайни/последни** състояния;

A е крайно множество от действия;

$\delta \subseteq$ **S x A** $\nrightarrow$ **S** е детерминистична, частична функция.

FSA terminology	State Machine terminology
alphabet	actions
string	trace†
language L(M)	behavior Beh(M)

trace † - пътека, която завършва с последно състояние.

L(M) може да бъде безкраен (безкраен брой стрингове на M).

Крайни изпълнения и безкрайно поведение

• Крайни изпълнения и безкрайно поведение:

– модел, включващ (само) ограничени пътеки (***finite-trace model***).

Пример CSP. Машините са с поведение, съставено с **безкрайно** множество от **крайни** пътеки:

➢ Опростява разсъжденията;

➢ Практическа резонност – безкрайното изпълнение не може да се види ...;

➢ Недостатък – невъзможност да опише “deadlocks”, “ fairness”. За тази цел се изисква усложняване на структурата на пътеките и поведението.

Разсъждения върху МС

- **Необходимост**
- **Най-важната характеристика на МС е инвариантността:** предикати Q, които са верни за всички достижими състояния. (*Пр*. Броячите)

Случаи за разсъждения върху МС

- **Крайни МС (софтуерен инструмент)**
- **Безкрайни МС**

***А) Индукция върху състоянията от изпълненията*. Удобна е когато има рекурсивна структура на домейна.**

Нека има изпълнение: $\langle s_0,\ a_1,\ s_1,\ a_2,\ \ldots,\ s_{i-1},\ a_i,\ s_i,\ \ldots\rangle$. За да се докаже, че характеристиката Θ е инварианта, необходимо за всяко изпълнение:

1. ***Основен случай:*** Показва се валидност за началното състояние $\mathbf{s_0}$
2. ***Индуктивна стъпка:*** Приема се валидност за състояние $\boldsymbol{s_{i-1}}$ и се доказва за състояние $\boldsymbol{s_i}$

Случаи за разсъждения върху МС - Proving an Invariant

B) Предикатът, който формира на състоянията, е по-силен от инвариантността, която се стремим да докажем

$$P \Rightarrow \Theta$$

C) Д-во чрез правило върху *pre/post спецификацията* (най-често използвана):

(Case analysis of all actions)

1. Показва се, че Θ е истинен за всички начални състояния
2. За всяко действие
   > Приема се, че
   - pre-условието Φ е в сила в pre-състоянието,
   - инварианта Θ е валиден за pre-състоянието
   - post-условието Ψ е в сила в pre и post състоянията

   > показва се, че
   - Θ е валиден и в post - състоянието
3. Следователно Θ е инварианта