Модели на софтуерни системи

Лекция 8: **Паралелни софтуерни системи. Модел на поделяне на ресурсите. Еквивалентност на МС**

Олга Георгиева

СУ, Факултет по Математика и информатика,
Катедра СТ

Въведение: Паралелни процеси (Concurrency)

CONCURRENCE

1. съвпадение, стечение
2. съгласуване, координиране
3. съгласие, съдействие

Основни въпроси:

- Как се моделират паралелни системи?
- Какви характеристики, типични за паралелните системи, се очаква те да проявяват/описват?
- Как може да се гарантират тези характеристики и какви разсъждения могат да се правят за тях? Какви теории/логики се използват?

Същност на паралелните системи (The Essence of Concurrency)

- **Поведението на системата може да е много сложно.**

От какво се поражда тази сложност?

- Трудностите, свързани с паралелните системи, се решават чрез:

  А) Конструиране на **специфични езици и концепции**, който да се справят с присъщата сложност на изпълнението на системата;

  Б) **Разработване на теории за разсъждения** относно взаимодействащите си паралелни процеси;

- Проектирането на паралелни системи често изисква намиране на **надеждни техники за:**

  - ✓ координиране на изпълнението на примитивните процеси;
  - ✓ обмен на данни;
  - ✓ разпределение на паметта;
  - ✓ дефиниране на изпълнение, което минимизира времето за отговор (*response time) и максимизира производителността (throughput);*

Основни дефиниции

Какво е паралелност?

Дефиниция: **Паралелността** е свойство на компютърните системи, в които няколко изчислителни процеса се изпълняват едновременно и взаимодействат/комуникират помежду си.

Кога процесите са паралелни? Кога си взаимодействат?

Дефиниция: ***Паралелността е синхронизиран достъп до споделени ресурс(и).***

Какъв вид ресурс? …

Concurrency theory has been an active field of research in theoretical computer science since Carl Adam Petri's (1926-2010) seminal work on Petri Nets in the early 1960s.

In the years since, a wide variety of formalisms have been developed for modeling and reasoning about concurrency.

Паралелност (Concurrency)

- Споделеният ресурс може да бъде **споделена променлива** (**shared variable)** или **споделен комуникационен канал (shared message channel)**.

- **Споделена променлива:** след като процесът ***P*** запише стойност на променливата ***x***, то процесът ***Q*** може да я прочете**.**

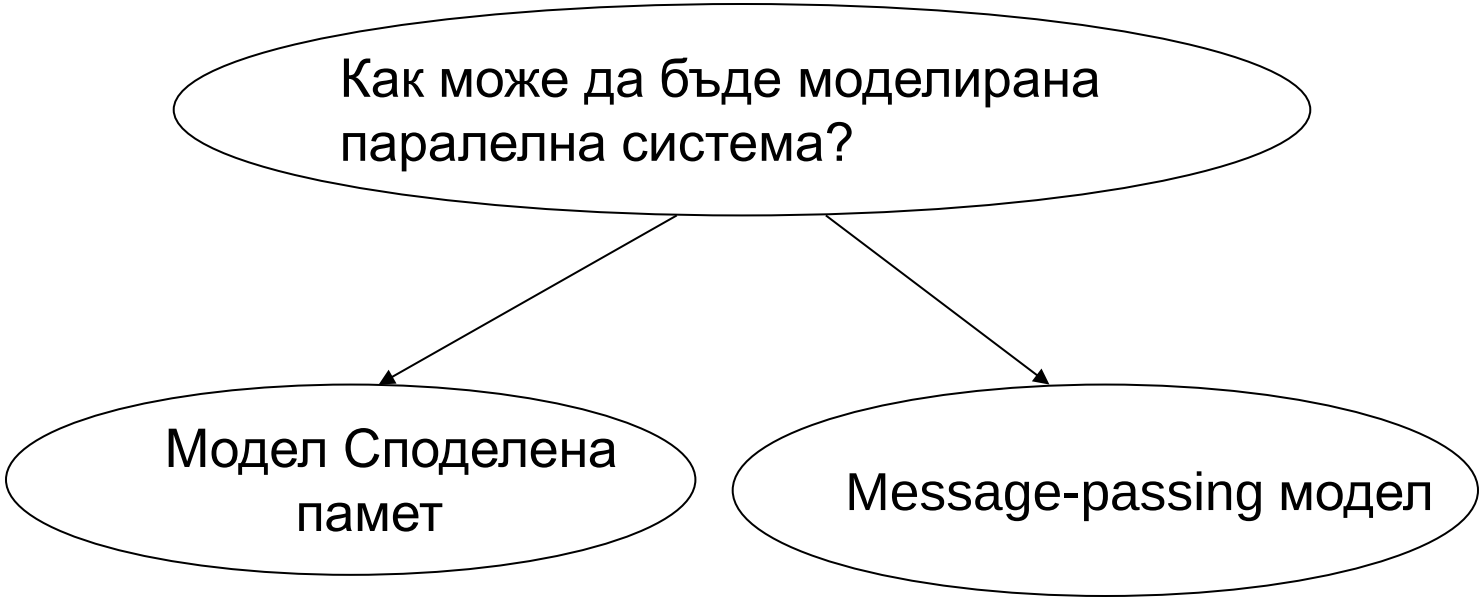
  - синхронизация *Пример?*

- **Споделен комуникационен канал** – каналът е среда, през която съобщенията се предават. Процесът ***P*** може да изпраща съобщения, които процесът ***Q*** да получава.

  - закъснение на разпространението, буфериране на съобщенията, ред на съобщенията, загуба на информация, дублиране, грешки.

Пример?

Моделиране на паралелни процеси



Изборът зависи от:

- характеристиките на системата;
- начинът за разсъждение;
- субективни преценки (вкус, предпочитание, умения).

Пример 1: Паралелни процеси (1)

x **–** *споделена променлива, инициализирана с* ***0***

P, ***Q*** – процеси, които се описват с код:

$x := 1$

$x := x+1$

Ако ***P*** и ***Q*** се изпълняват последователно **P; Q**, то x = ? след приключването им?

а **Q ; P** ?

$$P;\ Q \equiv$$

$$\begin{array}{l}
\{x = 0\} \\
\quad x := 1 \quad (P) \\
\{x = 1\} \\
\quad x := x + 1\ (P) \\
\{x = 2\} \\
\quad x := 1 \quad (Q) \\
\{x = 1\} \\
\quad x := x + 1\ (Q) \\
\{x = 2\}
\end{array}$$

Пример 1: Паралелни процеси (2)

А ако ***P*** и ***Q*** се изпълняват паралелно ***P*** || ***Q*** ?

$$
\begin{array}{l}
\{x = 0\} \\
\quad x := 1 \qquad (P) \\
\{x = 1\} \\
\quad x := 1 \qquad (Q) \\
\{x = 1\} \\
\quad x := x + 1 \, (P) \\
\{x = 2\} \\
\quad x := x + 1 \, (Q) \\
\{x = 3\}
\end{array}
$$

Summarizing, we have:

$$
\begin{array}{l}
\{x = 0\} \\
\quad P; \, Q \\
\{x = 2\}
\end{array}
$$

$$
\begin{array}{l}
\{x = 0\} \\
\quad P \parallel Q \\
\{x = 2 \vee x = 3\}
\end{array}
$$

- interleaved

- тагване (tagging)

Какво е поведението на последователния и на паралелния процес по отношение на получения резултат?

Основен въпрос

1. **Атомарна (*atomic*) операция (**strongly depends on the level of abstraction)**.**

Bank Account Example:

Level of abstraction	**Atomic**	**Non-atomic**
1) Bank account	withdrawing, depositing, ..	transferring
2) Bank‘s data base	accessing, updating file	withdrawing, ...
3) Physical disk	reading, writing at registers	accessing
4) Individual disk pages	reading, writing on individual pages	
5) Reading and writing to the bits	...	
6) Magnets, electrons,	...	

Основен въпрос: Дефиниране на атомарните операции

Атомарната операция е:

- *неразделима*:
- не може да бъде *наблюдавано* нейно междинно състояние.

- ***Формализъм***: Нека ***f*** е не атомарна операция, която е резултат на последователност от изпълнение на две атомарни операции ***g*** и ***h***:

$$f = \ll g \gg ; \ll h \gg$$

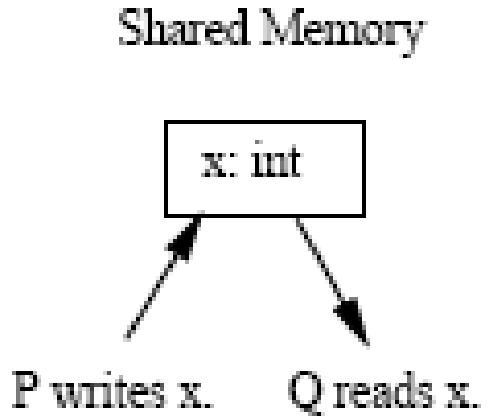
Тогава, друг процес може да прекъсне ***f*** . Това може да промени състоянието на системата. Така post условията на ***g*** може да не се изпълняват, когато ***h*** стартира своето изпълнение.

- *Пример*: Сумата на 2 банкови сметки може да доведе до разлика в стойността.

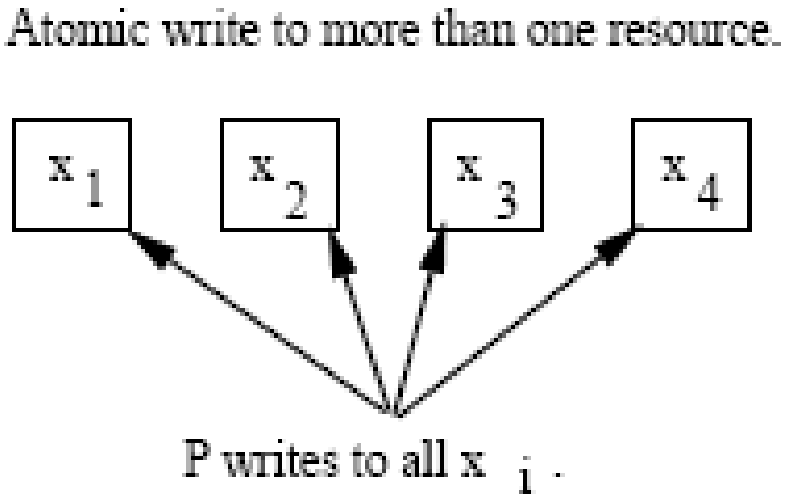
Начини за комуникация

Модел Споделена памет

1) Споделена променлива (или ресурс)



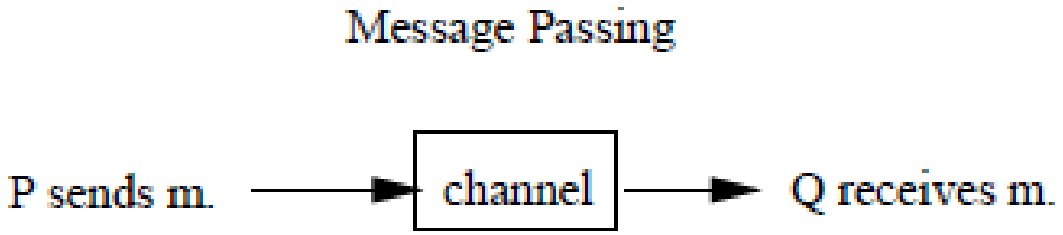
2) *Множество* от споделени променливи (*пр*. replicated databases, multiprocessor with main memory and local caches per processor).



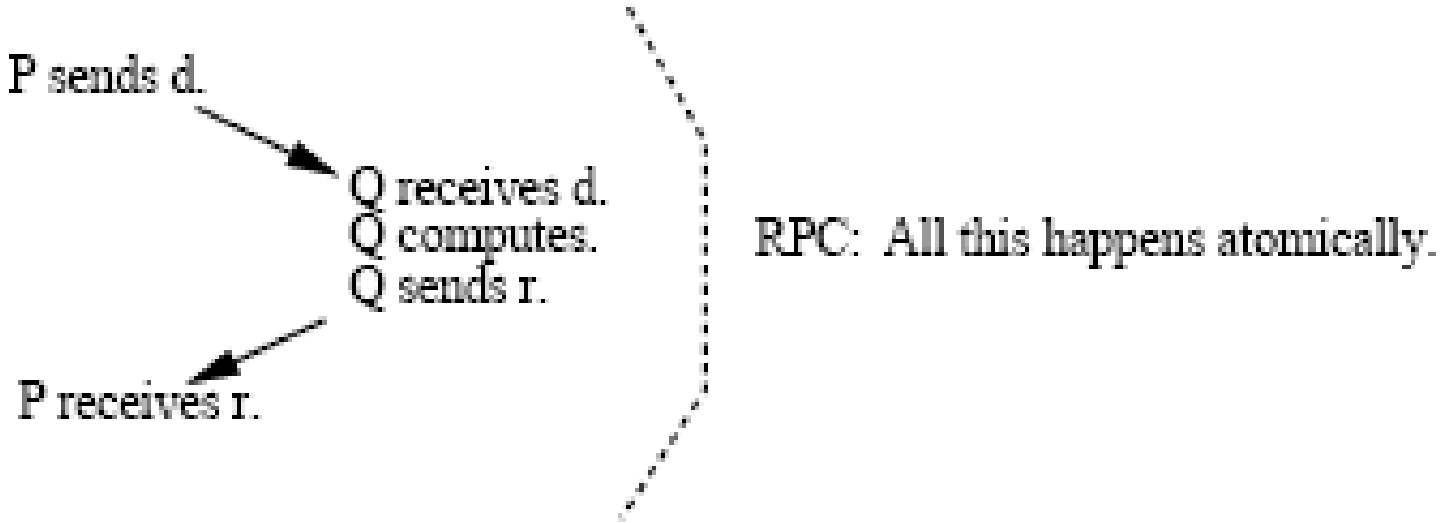
Комуникация

Message passing модел

1) Комуникационен канал (буфер). Кои са атомарните операции?



2) Атомарно разпространение (remote procedure call …)



Синхронизиране

Модел Споделена памет

1) Безопасност**: mutual exclusion**

- синхронизиращи конструкции (semaphores, condition variables, monitors, … more ?)

2) Прочитане на **последната записана стойност или**

- по определен ред: - partial order or (rarely)
- real time order.

- *Пример*: Highway control system.

Синхронизиране

Message passing модел

- Координиране на изпращането и получаването на съобщения

- Видове:
  - *asynchronous* : non-blocking sender (example?)
  - *synchronous* : blocking sender and receiver (example?)

- Буфериране (*Buffered message passing* - channel capacity)

- Communicating Sequential Processes (CSP), a model of concurrent systems is based on *synchronous* message passing (C.A.R. Hoare, 1985)

Модел Споделена памет
Shared-Memory Model by state machines

x **–** споделена променлива, инициализирана с ***0.***

P, ***Q*** – процеси с код:

$x := 1$

$x := x + 1$

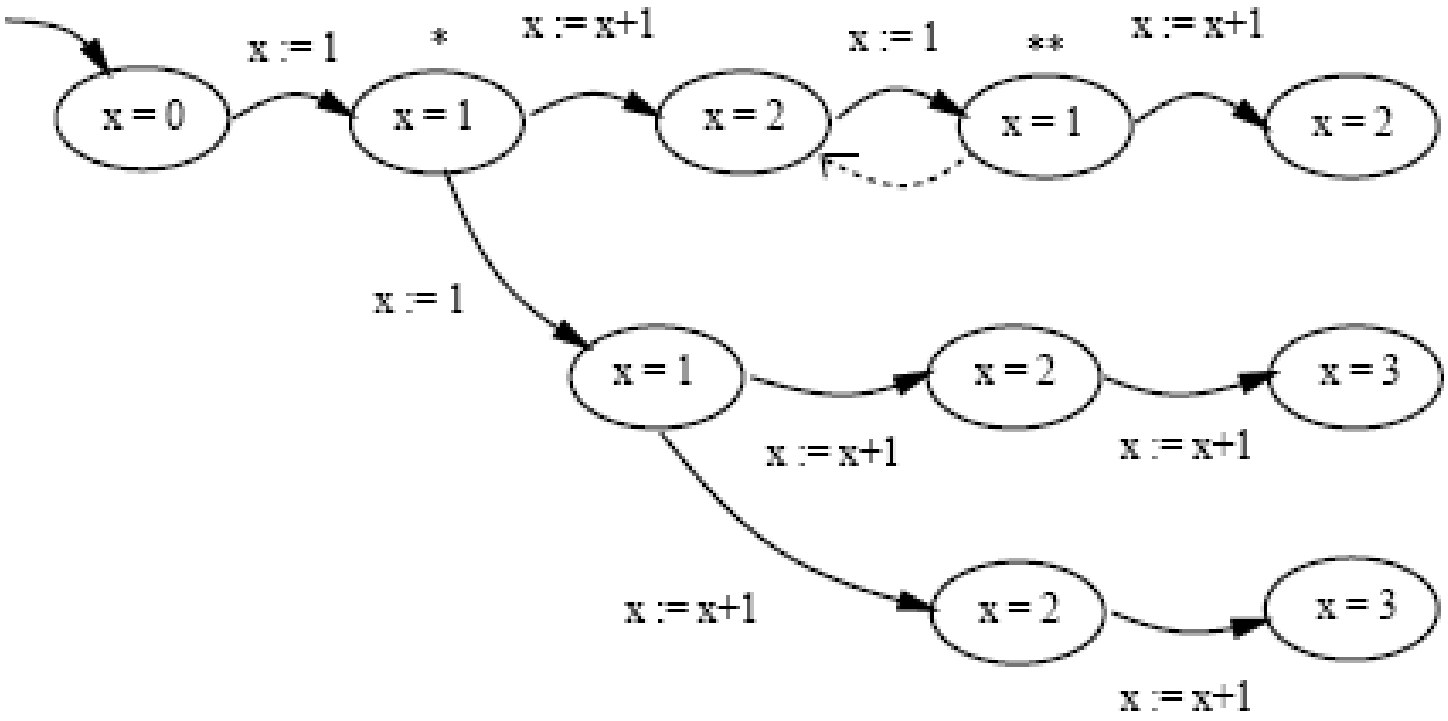
При паралелни процеси с взаимни превключвания се получават следните пътеки:

$x := 1 \quad (P)$	$x := 1 \quad (P)$	$x := 1 \quad (P)$
$x := x + 1 \quad (P)$	$x := 1 \quad (Q)$	$x := 1 \quad (Q)$
$x := 1 \quad (Q)$	$x := x + 1 \quad (P)$	$x := x + 1 \quad (Q)$
$x := x + 1 \quad (Q)$	$x := x + 1 \quad (Q)$	$x := x + 1 \quad (P)$
$\{x = 2\}$	$\{x = 3\}$	$\{x = 3\}$

и ! още 3 симетрични случая.

Модел Споделена памет, моделиран с машина на състоянието (1)

- Граф на паралелното изпълнение на P и Q:



Important ! We need to keep track of what P and Q have done already.

Модел Споделена памет, моделиран с машина на състоянието (2)

Трябва да се следи изпълнението на отделните паралелни примитивни процеси. За целта се въвежда:

> програмен брояч (**program counter)** (**pc)** за всеки примитивен процес. Той се прибавя към променливите, които дефинират състоянието на съответния примитив.

> Начин за различаване на преходи между състоянията, който си приличат изисква таг (**to tag)** с името на процеса към всяко действие, чиято реализация променя състоянието.

Модел Споделена памет, моделиран с машина на състоянието (3)

Combining State Machines

- Моделират се поотделно всички примитивни процеси:

$$P = (S_P, I_P, A_P, \delta_P)$$
$$Q = (S_Q, I_Q, A_Q, \delta_Q)$$

- За МС на ***P*** **||** ***Q*** = ? Вярно е, че:

$$traces(P \,||\, Q) \neq traces(P) \cap traces(Q)$$

- Има трасета в ***P*** **||** ***Q*** , които не са трасета нито в ***P*** нито в ***Q***

traces**(** ***P*** **||** ***Q*****)** $\notin$ (***traces***(***P***) or ***traces***(***Q***)).

Модел Споделена памет, моделиран с машина на състоянието (4)

Combining State Machines

1) Разделят се променливите на отделните примитиви на **локални и глобални**.

(Divide the state variables ***vars*(*P*$_i$)** of each process ***P***$_i$ into *locals* and *globals.* The locals are those that are affected by only that process the globals are those that are shared with all other processes and *can be changed by any of them.)*

2) Към локалните променливи на всеки процес ***P*** се прибавя програмния му брояч ***pc*$_P$** .

Remind:

Each state is a mapping from variables to values. For simplicity, we will assume the same value set, ***Val***, over which all the different types of variables range.

s | ***V*** to stand for ***dom*(*s*** | ***V*** restriction of the domain of state **s** to just the variables in **V** .

Also **V =V′** stands for $\forall x \in V. x = x'$ (all variables in **V** stay the same in value).

Модел Споделена памет, моделиран с машина на състоянието (5)

Composite State Machines

Тогава

$$P \,||\, Q = (S_{P||Q}, I_{P||Q}, A_{P||Q}, \delta_{P||Q})$$

дефинира **нова МС** както следва:

• Състояния:

$S_{P||Q} = \{ (locals(P) \times (locals(Q) \times (globals(P, Q)) \nrightarrow Val \}$

where

$globals(P, Q) = vars(P) \cap vars(Q)$

$locals(P) = vars(P) \setminus globals(P, Q)$

$locals(Q) = vars(Q) \setminus globals(P, Q)$

$pc_P \in locals(P) \wedge pc_Q \in locals(Q)$

! Всеки примитивен процес свободно променя собствените си променливи, но не и други променливи. Всеки примитивен процес може да промени всяка глобална променлива.

Composite State Machines

- Начални състояния

$$I_{P||Q} = \{i \in S_{P||Q} \mid i \restriction vars(P) \in I_P \wedge i \restriction vars(Q) \in I_Q\}$$

- Действия

$A_{P||Q} = A_P \cup A_Q$ *защо?*

Note 1 !: If we *tagging* each action with a process name, then:

$A_P \cap A_Q = \emptyset$

Note 2 !: If we *tagging* each action with a process name, then If we want shared actions, then we would not need to pull this trick of tagging each action.

Composite State Machines

- **Функция на преходите:**

$$\delta_{P||Q} = \{(s, a, s') \in S_{P||Q} \times A_{P||Q} \times S_{P||Q} \mid$$

$$(a \in A_P \wedge (s \restriction vars(P), a, s' \restriction vars(P)) \in \delta_P \wedge pc'_P = pc_P + 1) \wedge locals(Q) = locals'(Q)$$

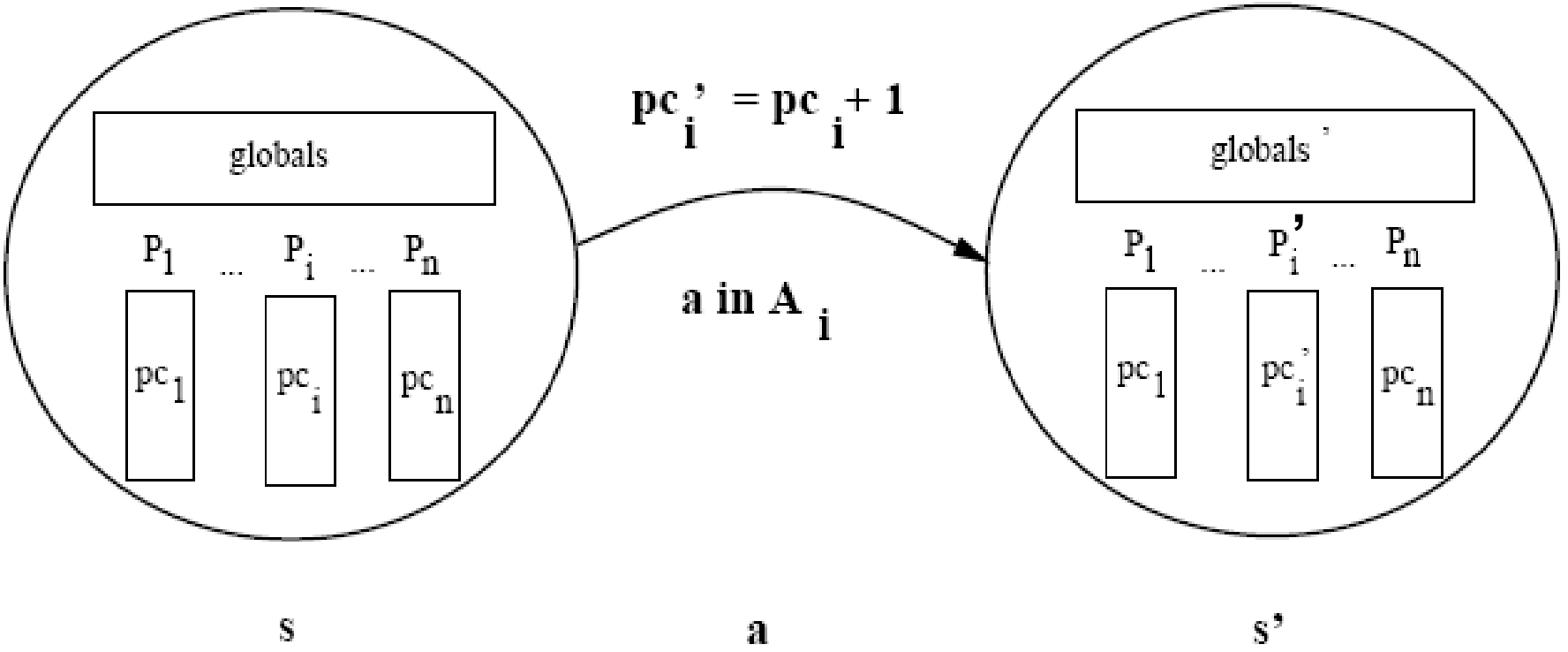
V

$$(a \in A_Q \wedge (s \restriction vars(Q), a, s' \restriction vars(Q)) \in \delta_Q \wedge pc'_Q = pc_Q + 1 \wedge locals(P) = locals'(P))\}$$

A step **(*s, a, s'*)** is a step of one of the two machines: if for ***P***, then its ***pc*** gets bumped, and all locals of ***Q*** stay same; similarly, if for ***Q*.**

Composite State Machines

- **A single step transition of a generalized composite state machine with *n* processes:**



Обобщение

• Моделът комбинира МС, но не е композитен, защото няма характеристика на еквивалентост:

$$traces(P \,||\, Q) \neq traces(P) \cap traces(Q)$$

$$traces(P \,||\, Q) \neq traces(P) \cup traces(Q)$$

• Характеристиката (предикат) *Inv* , който е валиден и за P и за Q не задължително е валиден за паралелния процес ***P || Q***.

$$Inv(P) \wedge Inv(Q) \not\Rightarrow Inv(P \,||\, Q)$$

• Характеристика, която е валидна и за ***P || Q*** е *глобална инвариантa*.

Приложения на модела

Този модел е общовалиден за програмни езици, които поддържат модел споделена памет.

Примери:

- any language that has a **Threads** interface such as C-Threads, Modula-2++, Modula-3,

- or any *operating system* that supports lightweight threads of control (e.g. Mach is based on this model).

Много системи се описват с този модел:

- People design systems with this model in mind.
- Most transactional systems assume either physically centralized database or a globally-viewed distributed DB.

Еквивалентност на МС

В каква връзка могат да бъдат две МС ?

> еквивалентност
> удовлетворяване на характеристики (в някакъв смисъл)

***Полза от изследване за еквивалентност*:**

Машина М1 може **да замести** машина М2 без това да променя цялата система (*пр*.: компилатор, модулна архитектура). Спазва се принципът **referential transparency**: при модулярност при програмирането, функционално програмиране, доказателство с уравнения ...);

Причина:

> Ефективност – намаляване на броя на състоянията, преходите, променливите ...

> Също – цена, поддръжка, портабилност, естетика ...

Кога две машини са еквивалентни?

- ***Практически смисъл на въпроса***

Отговорът е възможен в **определен контекст** според искането заместване на МС*;*

> *интуитивно определение*: еквивалентност при наблюдение (според външен наблюдател).

> *научен отговор*: обект на дебат и изследване

- ***Еквивалентност в термините на поведение вход – изход*:**

- *тесен* смисъл - еднаквост на реализирания изход;

- *широк* смисъл - сложните процеси изискват да се следят и *междинните преходи и взаимодействия*, а не само крайното състояние/изход

- ***еднаквост на*** поведението; ... пътеките; ... състоянията/действията... променливите на състоянието ...

Пр. Light example with *flick* action and three positional light with the same action

Какво означава еквивалентност ?

Еквивалентността се дефинира трудно дори в рамките на еднакъв вид пътеки:

Пр. Регистър

$$\langle x = 0, write(1)/ok(), x = 1, read()/ok(1), x = 1, read()/ok(1), x = 1, write(5)/ok(), x = 5 \rangle$$

Дефинирайте пътека на състоянията:

A state-based trace of this execution is:

$$\langle x = 0, x = 1, x = 1, x = 1, x = 5 \rangle$$

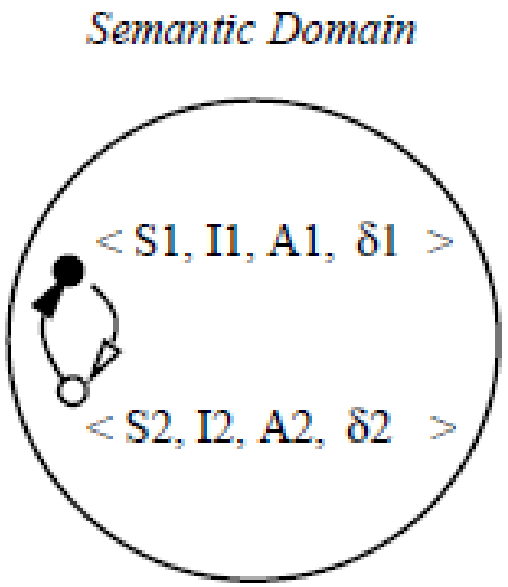
Is it equivalent to the following?

$$\langle x = 0, x = 1, x = 5 \rangle$$

А ако е безкрайна ?!...

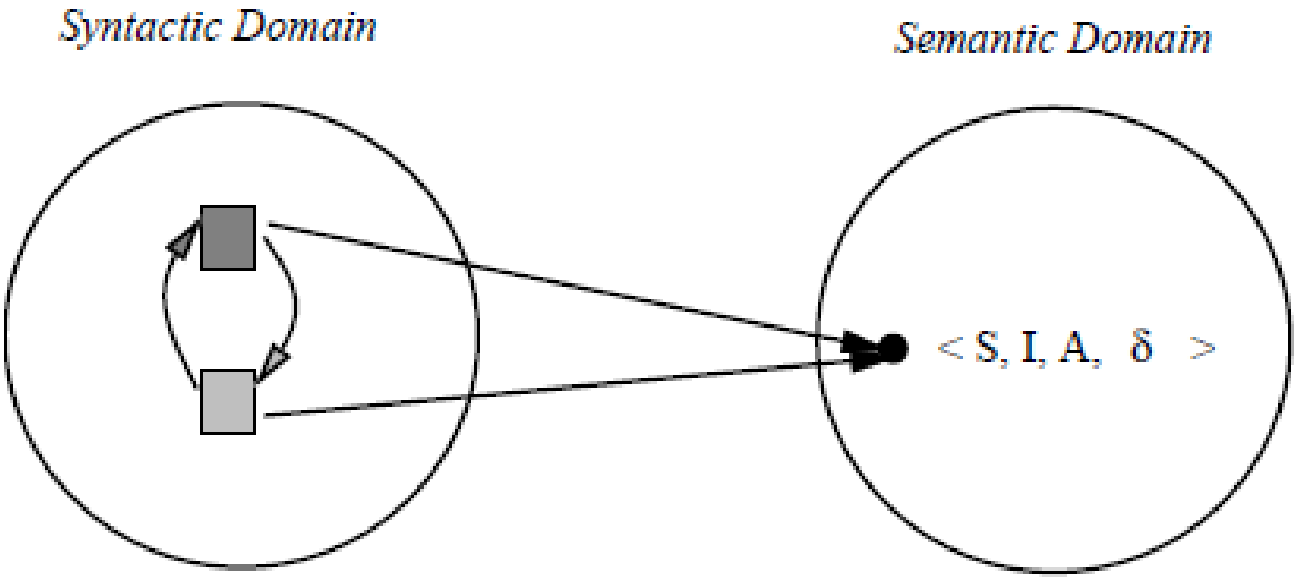
Показване на еквивалентност на две МС

Два възможни подхода: а) семантичен – за множествата в четворката на всяка МС или относно поведението на машините



Илюстриране на еквивалентността на две МС

или синтактичен: б) поради еквивалентност на синтаксиса се дефинира и еквивалентност на семантиката (Z, pre-post, CSP описания)



в) да се докаже еквивалентност от първи принцип чрез мат. логика, множества...
г) показва се, че едната “симулира” другата и обратно или
д) едната се трансформира в другата (*пр*. компилатор).