## Глава 2: Процеси и нишки

**Процес:** Единица на последователно изпълнение, която включва действия в определен ред.

**Нишка (Thread):** Лека версия на процес, която споделя памет с други нишки в рамките на един процес.

**Комплексните системи** се структурират като набор от по-прости дейности, всяка представена като последователен процес.

Процесите могат да се припокриват или да бъдат конкурентни, за да отразяват паралелизма в реалния свят или за управление на комуникации и други устройства.

Моделиране на процеси:

**Крайни автомати (Finite State Processes - FSP):** Моделират **процесите като поредици от действия**.

- превключвател: SWITCH = (on -> off -> SWITCH)

**Системи с етикетирани преходи** (Labelled Transition Systems- LTS):

**Графично представяне на FSP**, което показва състояния и преходи.

Моделите се описват с помощта на **крайни състояния**, известни като **LTS.** Тези системи се представят по два начина:

- **Текстов алгебричен вариант (FSP - Finite State Processes):** описание на системата чрез математически запис. Използва се за точно дефиниране на преходите между състоянията. Подходящ за прецизно моделиране на процеси
- **Графичен вариант (LTS):** Визуално предст. с-мата като граф:

  -Състоянията са представени като възли

-Преходите между състоянията - като стрелки с етикети

-Позволява по-лесно разбиране и визуализация на системата

Инструментът **LTSA** (Labelled Transition Systems Analyser) се използва за:

- Анализ на създадените модели
- Визуализация на графичното представяне
- Проверка на системните свойства
- Откриване на потенциални проблеми в модела

Този подход е особено полезен при моделиране на паралелни и разпределени системи, протоколи и софтуерни архитектури.

**Процесът** представлява изпълнение на последователна програма. Моделира се като **крайна машина на състоянията**, която преминава от едно състояние в друго чрез изпълнение на поредица от атомарни действия. Основни характеристики:

- Всяко състояние описва **текущото положение на изпълнението**
- **Преходите между състоянията стават чрез прости, неделими (атомарни) действия**
- Моделът позволява точно проследяване на логиката на изпълнение
- Крайният брой състояния осигурява възможн. за пълен анализ

**FSP- префикс на действие**: Когато **x** е действие, а **P** е процес, префиксът **(x-> P)** описва процес, който:

1. Първоначално извършва действието **x**

2. После се държи точно както е описан от процеса **P**

**ONESHOT = (once -> STOP) -** крайна машина на състояния (терминиращ процес). Извършва действието "once" и спира (STOP)

**Конвенции:**

- **Действията започват с малки букви**
- **Процесите започват с главни букви**

Същността на префиксирането е да се моделира последователност от действия, където всяко следващо действие зависи от предишното.

**Превключвател**: Пример за модел на превключвател, който преминава между състояния "включено" и "изключено". Използване на **префикс на действие** и **рекурсия** за моделиране на повтарящо се поведение: **SWITCH = (on->off->SWITCH)**

**Светофар**: Модел на светофар с последователност от действия: червено -> оранжево -> зелено -> оранжево:

TRAFFICLIGHT= **(red->orange->green->orange**-> TRAFFICLIGHT).

Модел на процес с **алтернативни действия**: **(x-> P | y-> Q)**

- Първоначално процесът може да извърши или действие **x**, или действие **y**
- Първото извършено действие определя по-нататъшното поведение
- Ако първото действие е **x**, процесът продължава като **P**
- Ако първото действие е **y**, процесът продължава като **Q**

Моделира недетерминистично поведение. Първоначален избор между две възможни действия. Различно продължение в зависимост от избраното първо действие

Автомат за напитки, който избира между кафе и чай:

**DRINKS = (red->coffee->DRINKS**

**|blue->tea->DRINKS).**

FSP модел на машината с четирите цветни бутона: Само жълтият бутон предизвиква действие "candy" Моделът е недетерминистичен - може да се избере произволен бутон при всяко състояние. След натискане на бутон, машината се връща в същото състояние:

**FAULTY = ({red,blue,green}-> FAULTY**

**|yellow -> candy -> FAULTY ).**

**Недетерминиран избор**: (x-> P | x -> Q) Моделиране на процес, който **Извършва действие x и след действие x може да премине или в състояние P, или в състояние Q** (хвърляне на монета):

**COIN = (toss->HEADS|toss->TAILS),**

**HEADS= (heads->COIN),**

**TAILS= (tails->COIN).**

Как да моделираме ненадежден комуникационен канал, който приема действия и ако възникне повреда, не произвежда изход, в противен случай извършва изходящо действие?

**CHAN = (in->CHAN**

**|in->out->CHAN ).**

Буфер: Моделиране на буфер, който приема стойности в диапазона 0 до 3 и след това ги извежда:

**BUFF = (in[i:0..3]->out[i]-> BUFF) or**

**BUFF(N=3) = (in[i:0..N]->out[i]-> BUFF).**

В контекста на FSP **индексите** се използват за **дефиниране на локални процеси и действия**, които позволяват гъвкаво моделиране чрез параметризация.

Индексни процеси и действия: Локалните процеси с индекси са еквивалентни на дефиниране на отделен процес за всяка стойност на индекса. Това означава, че можем да използваме диапазони и индекси, за да моделираме различни вариации на процесите и техните действия.

**Декларации на константи и диапазони**:

const N = 1 (N е константа със стойност 1)

range T = 0..N (T е диапазон от 0 до N (в този случай 0..1))

range R = 0..2*N (R е диапазон от 0 до 2*N (в този случай 0..2))

SUM = (in[a:T][b:T] -> TOTAL[a+b]),

TOTAL[s:R] = (out[s] -> SUM).

SUM: Процесът очаква две входни действия in[a][b], където a и b са стойности от диапазона T (0..1). След като получи тези стойности, изчислява тяхната сума a+b и преминава към състояние TOTAL[a+b].

TOTAL[s:R]: В това състояние, системата изпълнява действието out[s], където s е стойност от диапазона R (0..2), след което се връща към процеса SUM.

**Guarded actions** (охранявани действия) са механизъм в **FSP**, който използва условия (guards), за да управлява кои действия могат да бъдат изпълнени в даден момент. Това позволява моделирането на поведението на системата да бъде обвързано с текущото състояние и зададени условия.

**when B x -> P | y -> Q** означава следното:

- Ако условието B е **истинно** (true), тогава действията x и y са **достъпни** за избор.
- Ако условието B е **неистинно** (false), тогава действието x **не може да бъде избрано**. В този случай само y остава допустимо, ако няма друга охрана.

Брояч (COUNT):

COUNT (N=3) = COUNT[0],

COUNT[i:0..N] = (when(i<N) inc->COUNT[i+1]

|when(i>0) dec->COUNT[i-1] ).

**Деклариране на брояч**: COUNT представлява брояч, който може да се увеличава (inc) или намалява (dec). Нач. състояние е COUNT[0].

**Състояния на брояча**: Броячът може да има стойности i в диапазона 0..N, където N е зададена константа (в примера N=3).

**Охранявани действия**: when(i<N) inc->COUNT[i+1]: Действието inc (увеличаване) е достъпно само ако текущата стойност на брояча i е по-малка от максималната стойност N. След увеличаването броячът преминава в състояние COUNT[i+1]. when(i>0) dec->COUNT[i-1]: Действието dec (намаляване) е достъпно само ако текущата стойност

на брояча i е по-голяма от 0. След намаляването броячът преминава в състояние COUNT[i-1].

Guarded Actions (Охраняв. действия)-таймер за обратно отброяване:

1. Започва с определен брой стъпки (тикания).
2. Извършва тикания, докато достигне 0.
3. Издава звуков сигнал (beep), когато броячът стигне 0.
4. Може да бъде спрян по всяко време чрез действие stop.

COUNTDOWN (N=3) = (start->COUNTDOWN[N]),

COUNTDOWN[i:0..N] = (when(i>0) tick->COUNTDOWN[i-1]

|when(i==0) beep->STOP |stop->STOP ).

**Инициализация:**`COUNTDOWN(N=3)=(start- COUNTDOWN[N]):`

Таймерът започва с действие `start`, което го задава в състояние `COUNTDOWN[N]` (тук `N=3`).

**Състояние на отброяване:** `COUNTDOWN[i:0..N]:` Таймерът поддържа състояния, зависещи от текущата стойност на брояча `i`. Стойността на `i` е в диапазона `0..N`.

**Охранявани действия:** `when(i>0) tick->COUNTDOWN[i-1]:`

Ако `i > 0`, действието `tick` е позволено. То намалява стойността на брояча с 1 и преминава в състояние `COUNTDOWN[i-1]`.

`when(i==0) beep->STOP`: Ако `i == 0`, действието `beep` е позволено. То сигнализира края на отброяването и преминава в състояние `STOP`.

`stop->STOP`: Действието `stop` може да бъде изпълнено във всяко състояние (независимо от стойността на `i`). То прекратява таймера и го поставя в състояние `STOP`.

**Състояние STOP:** В състояние `STOP` таймерът спира, като не извършва никакви действия повече.

**Process Alphabets (Азбука на процеса): множеството от действия**, в които процесът може да участва. Тези действия дефинират взаимодействията на процеса с външната среда.

**Азбука на процеса- списъкът с действия**, които са част от дефиницията на процеса. Например, ако процесът съдържа действия като start, tick, stop и beep, тези действия формират неговата азбука.

**Имплицитно дефиниране**: Азбуката на процеса **не се дефинира ръчно**, а автоматично се извежда от действията, които са описани в дефиницията на процеса.

**Извеждане на азбуката**: Азбуката на даден процес може да бъде визуализирана с помощта на **LTSA** в прозореца "Alphabet"

**Process Alphabet Extension (Разширение на азбуката на процеса):** позволява да се добавят допълнителни действия към **имплицитната азбука** на процеса, които иначе не са включени в неговата дефиниция. По подразбиране, азбуката на процеса се състои от действията, дефинирани в него. **Разширението на азбуката** позволява **ръчно добавяне** на допълнителни действия, които не са изрично описани в логиката на процеса- с оператора **+{}.**

WRITER = (write[1]->write[3]->WRITER)

**+{write[0..3]}.**

**Имплицитна азбука**: Без разширение, азбуката на процеса включва само действията, дефинирани в логиката на процеса:write[1], write[3]

**Разширение на азбуката**: С добавянето на **+{write[0..3]},** азбуката на процеса се разширява така, че да включва **всички действия в диапазона write[0] до write[3]**, дори ако някои от тях не са дефинирани в логиката на процеса.

След разшир., азбуката на п-са: { write[0], write[1], write[2], write[3] }

**FILTER** в FSP - обработва стойности **v** в диапазона **0 до 5**.:

FILTER = (in[v:0..5] -> DECIDE[v]),

DECIDE[v:0..5] =

(when (v <= 2) out[v] -> FILTER

|when (v > 2) FILTER).

**Приема входна стойност** v чрез действието in[v:0..5]. **Решава дали да изведе стойността или да я отхвърли**, в зависимост от условието:

Ако v <= 2, стойността се извежда чрез действието out[v].

Ако v > 2, ст-та се **отхвърля** и процесът се връща към нач. състояние.

**Основният процес: FILTER -**Процесът започва с действието in[v:0..5], което приема стойност **v** в диапазона от **0 до 5**. След приемане на стойността, той преминава в състоянието DECIDE[v].

**Решение в DECIDE[v]**: **Когато v <= 2:** Стойността се извежда чрез действието out[v]. След това процесът се връща в начално състояние (FILTER), готов да приеме нова стойност. **Когато v > 2:** Стойността се **отхвърля** (не се извежда никакво действие). Процесът директно се връща в състояние FILTER.

**Моделиране на процеси като крайни автомати**

**Процесът** е единица на конкурентност, която представлява изпълнението на дадена програма. Процесът е независима единица, която може да се изпълнява паралелно с други процеси.

**LTS (Labelled Transition Systems)**: Използва се за моделиране на процеси като **крайни автомати**, където: 1/Състоянията представляват различни етапи на процеса. 2/**Преходите** между състоянията са последователности от **атомарни действия**.

**FSP** : Език за специфициране на процеси с помощта на:

- **Префикс "->"**: Определя следващото действие.
- **Избор " | "**: Позволява избор между няколко действия.
- **Рекурсия**: Позволява повтарящо се поведение.

**Нишки в Java**: Нишките (threads) се използват за имплементиране на процеси. Нишката е лека единица на изпълнение, която може да се създава, стартира и спира динамично. Thread Lifecycle:

-**CREATED** (Създадена): създадена, но все още не е стартирана.

-**RUNNING** (Изпълнява се): Нишката изпълнява задачата си.

-**RUNNABLE** (Готова за изпълнение): Нишката е готова за изпълнение, но изчаква ресурс от процесора.

-**NON-RUNNABLE** (Неизпълнима): Нишката временно е в изчакване

-**TERMINATED** (Прекратена): Нишката е приключила изпълнението си и не може да бъде рестартирана.