EVENTO, COMUNICAÇÃO, ALFABETO, PROCESSO, E OPERADORES BÁSICOS PARA PROJETO DE PROCESSOS

Alexandre Mota & Augusto Sampaio

CSP

- Concebida por Tony Hoare [85]
- Atualizada por Bill Roscoe [97] para fins de automação (Verificação de Modelos)
  - FDR (Failures-Divergences Refinement)
- Foi projetada visando modelar sistemas concorrentes/distribuídos de forma simples e elegante

Elementos básicos de CSP

- A linguagem é composta por:
  - Eventos (canais)
  - Processos (equações)
  - Operadores sobre processos e/ou eventos
  - Estruturas de dados baseadas no paradigma funcional

- Por isto é considerada uma **álgebra de processos**

Processos

- Processos correspondem a componentes de um sistema distribuído
- Cada processo possui seu próprio estado
  - Semelhante a um objeto
- Processos **comunicam-se** com outros

Comunicação

- Realizada através de **eventos** de comunicação
- Eventos são **atômicos, instantâneos**
- A comunicação **só ocorre** através da **sincronização** entre dois ou mais processos:
  - Mesmo evento ocorre em dois processos ao mesmo tempo

Comunicações

- Em CSP, a partir de
  - channel ch: T
- Podemos ter as comunicações
  - ch?var para uma entrada de dados
  - ch!exp para uma saída de dados
  - ch.expS para sincronização simples e/ou saída de dados
- Assim, evite ch?(1+x) e ch.(1+x*2), mas ch.x é válido

Comunicações restritas

- Dissemos anteriormente que
  - channel ch: Int
- era muito custoso
- Uma forma de usar a declaração acima sem introduzir tanto custo é
  - ch?x:A
- Onde A é um sub-conjunto de Int

Exemplo de comunicação restrita

- Dado
  - channel ch: Int
- Então, ch?x:{0,1,2} só permite a leitura de 3 valores apesar do tipo do canal ser inteiro
- Podemos também usar
  - ch?x:{x<=10}
- ou mesmo
  - ch?x?y:{(x-1)..(x+1)}

Eventos e alfabetos

- O alfabeto de um processo (αP) é o conjunto de todos os seus eventos:
- O alfabeto de uma especificação (Σ) é a união dos alfabetos de todos os processos (Events em FDR)
- A escolha de eventos deve ser cuidadosa
  - O modelo pode tornar-se não é analisável
- Suponha a ocorrência do pressionamento de qualquer tecla (1 evento) versus teclas específicas (N eventos)

Eventos

- Denotam acontecimentos do mundo real que são relevantes para o modelo

- Exemplos:
  - Chamadas de métodos
  - Fatos/ações perceptíveis (computador travou, pressionamento de uma tecla, etc.)
  - Sincronizações
    - Só continuo se…

Canais

- A diferença básica de canais para eventos é que canais comunicam dados
- Assim
  - channel ev (É um evento)
  - channel ch: Int (É um canal)

No final, só temos eventos…

- Canais são meros açúcares sintáticos para coleções de eventos
- Assim
  - channel ch: Int
- Produz
  - Os eventos {ch.0, ch.1, …, ch.N}
- Particularmente
  - {|ch|} = {ch.0, ch.1, …, ch.N}

Mas atenção…

- Uma declaração como
  - channel ch: Int
- É muito custosa porque produz um conjunto muito grande de eventos
- Será que você precisará realmente de todo esse conjunto?
- Tente simplificar sempre que possível!!!

STOP

- Processo sem comportamento
  - Representa um estado terminal, de onde não é possível **sair nunca mais**
- Este processo representa estado (processo)
  - Em *deadlock,* ou
  - Sem funcionar (quebrado)

Deadlock

- Em CSP, o deadlock pode ocorrer em duas situações
- Com o uso do processo STOP
  - É estático e deve ser propositalmente (falha)
- Decorrente de impasse
  - É dinâmico e deve ser evitado (erro na modelagem)

SKIP

- Também é um processo básico, mas denota comportamento de término com sucesso
- Sucesso significa ter o evento especial ($\surd$) no trace do processo
- Note que $\surd$ não é um evento sintático/declarável (channel)
  - $\Sigma^{\surd} = \Sigma \cup \{\surd\}$
- P = SKIP é um processo CSP válido!

Prefixo

- Operador usado para modelar comportamento linear (->)
  - ev -> Proc
- O processo a -> P espera indefinidamente por a, e então comporta-se como P
- Exemplo:
  - TwoSteps = leftFoot -> rightFoot -> SKIP
- Formas inválidas: `a -> b, P -> Q`

Definindo Processos

- Declarando os eventos (ou canais) de comunicação:

```
channel up, down
```

- Definindo (nomeando) um processo cujo alfabeto é determinado pelos eventos acima:

```
P0 = up -> down ->
          up -> down -> STOP
```

Definições Recursivas

```
P1 = up -> down -> P1
P2 = up -> down -> up -> down -> P2
```

- Através de equações recursivas (guardadas)
- Utilizadas para processos que
  - tem um comportamento repetitivo
  - normalmente não terminam

Processo *vs* definição de processo

- Operador de recursão:

  $\mu X.F(X)$

  representa "o processo" definido por

  $P = F(P)$

  para uma expressão F (guardada)

- $P = F(P)$ equivale a $P = \mu X.F(X)$
- Isto é, podemos ter definições (equações) diferentes, mas processos iguais

Definições mutuamente recursivas

```
Pu = up -> Pd
Pd = down -> Pu
```

- Processo *versus* Sistema:
  - Dois processos, um único sistema
  - Um processo não “chama” um outro processo, mas “comporta-se como” um outro processo

Escolha prefixada

- Operador para construir processos:

  `?x:A -> P(x)`

  onde `x` é uma variável, `A` um conjunto de eventos, e `P(x)` um processo

- O conjunto `A` pode ser infinito (Cuidado)

- `STOP` equivale a `?x:{} -> P(x)`

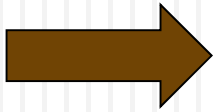
Escolha Determinística

- O processo P [] Q oferece a possibilidade tanto de se comportar quanto P quanto como Q
- A decisão sobre P ou Q não pertence a P [] Q, mas de quem interage com ele

Entrada e escolha

Seja c um canal do tipo $\{e_0, \ldots, e_N\}$, então

c?x -> P



c.e_0 -> P[e_0 / x]
[ ]
...
[ ]
c.e_N -> P[e_N / x]

Escolha prefixada

- Abreviação para o operador indexado de escolha externa:

```
?x:A -> P(x)
```

é equivalente a

```
[] x:A @ x -> P(x)
```

- Exemplo:

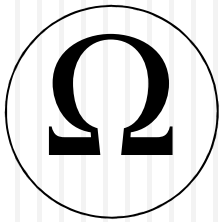
```
Sigma = {a,b,up,down}
REPEAT =
   [] x:Sigma @ x -> x -> REPEAT
```

Máquinas de estados e CSP

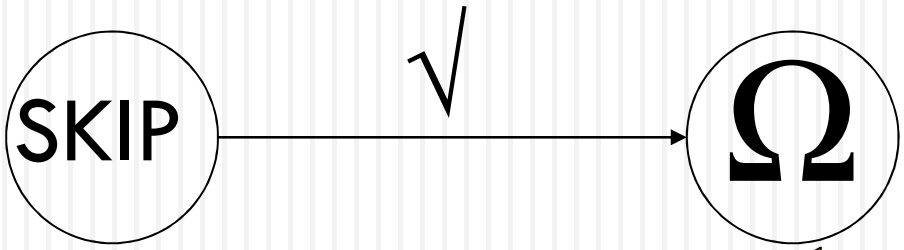
- Todo processo em CSP corresponde a uma "máquina de estados" (ou LTS)
  - **L**abelled **T**ransition **S**ystems

- Ferramentas de verificação de modelos (*model checking)*, como FDR, verificam processos com um **número finito de estados**

LTS de STOP

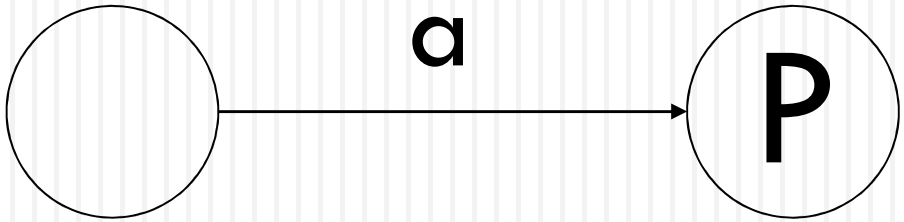


LTS de SKIP



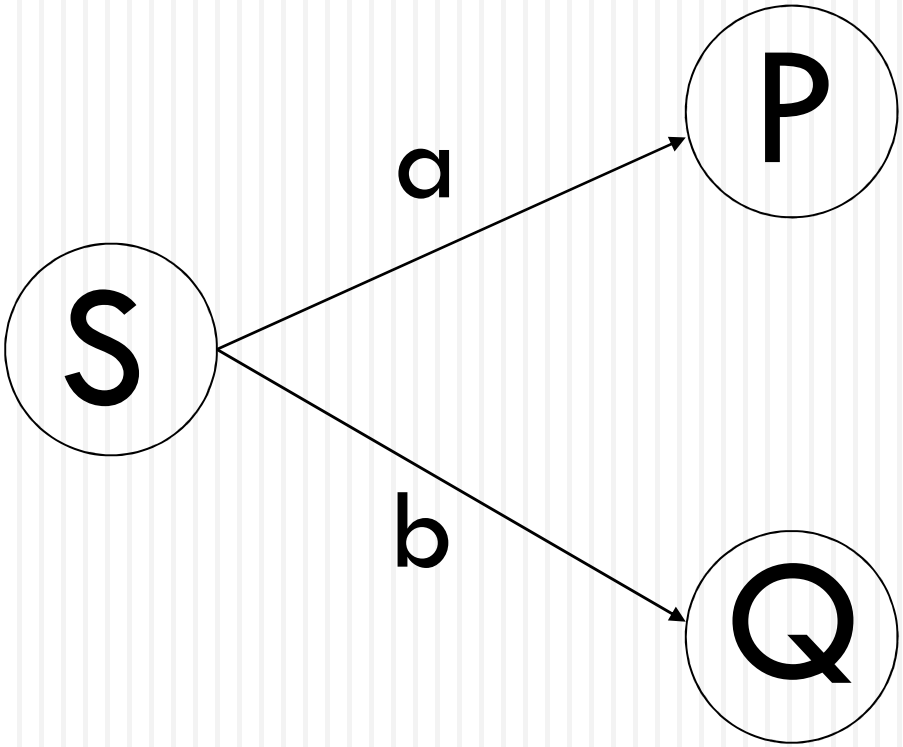
Por que se analisarmos o processo P = SKIP em FDR sobre deadlock, haverá uma confirmação dessa propriedade indesejável?

LTS de Prefixo (a -> P)



LTS da escolha determinística

Seja S = (a -> P) [] (b -> Q), então



Definições parametrizadas

- Corresponde a uma família, possivelmente **infinita**, de definições:

```
COUNT(n)=
   if n==0 then
      (up -> COUNT(1))
   else
      (up -> COUNT(n+1) []
       down -> COUNT(n-1))
```

Definições parametrizadas

- O número de estados pode ser limitado:

```
LCOUNT(L,n) =
    (n<L & up -> LCOUNT(L,n+1))
  [](n>0 & down -> LCOUNT(L,n-1))
```

onde

```
(cond & P)
```

corresponde a

```
if cond then P else STOP
```

Exemplo

```
DATA = {0,1}
channel left, right:DATA
COPY = left?x -> right!x -> COPY

Binf(s) =
 if s==<> then
    left?x -> Binf(<x>)
 else  (left?x -> Binf(s^<x>)
      []right!head(s) -> Binf(tail(s)))
```

Exemplo

```
ATM1 = incard?c -> pin.fpin(c) ->
       req?n -> dispense!n ->
       outcard.c ->

channel incard, outcard:CARD
channel pin:PINs
channel req, dispense:WA
```

Exemplo

```
CARD = {0..9}
datatype pinnumbers = PIN.Int
fpin(c) = PIN.c
PINs = {fpin(c) | c <- CARD}
WA = {10,20,30,40,50}
```

Ferramentsa

- ProBE
  - Animador para CSP
- FDR
  - Verificador de modelos para CSP
- Casper
  - Projetar protocolos com geração de especificação CSP automaticamente

Exercícios

- Do livro texto
  - Essenciais: 1.1.1, 1.1.2, 1.1.3
  - Opcionais: 1.1.5, 1.1.4