



Context-Sensitive Dynamic Partial Order Reduction

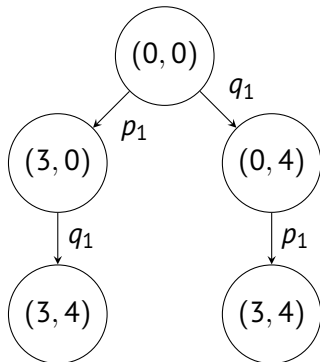
Jan Tužil

26. března 2018



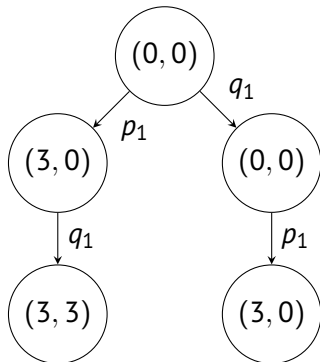
Example 1

$p: x := 3$ $q: y := 4$



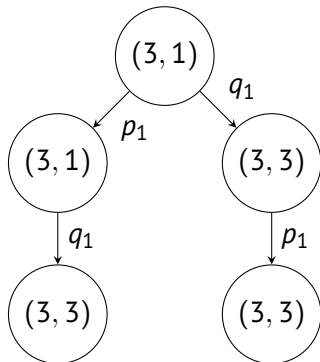
Example 2

$p: x := 3$ $q: y := x$



Example 3

$p: x := 3$ $q: y := x$





Pojmy

- Od naivního k Source-POR
- Od Source-POR k Context-Sensitive (Source-) POR



Notace

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$

▷ Iniciální stav

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$

val $s[_] : \text{Trace} \rightarrow \text{State}$

▷ Iniciální stav

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$

val $s[_] : \text{Trace} \rightarrow \text{State}$

▷ Iniciální stav

▷ Stav, do něhož doběhne stopa z s

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$

▷ Iniciální stav

val $s[_] : \text{Trace} \rightarrow \text{State}$

▷ Stav, do něhož doběhne stopa z s

val $\text{enabled} : \text{State} \rightarrow \text{Set}\langle \text{Process} \rangle$

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$

▷ Iniciální stav

val $s[_] : \text{Trace} \rightarrow \text{State}$

▷ Stav, do něhož doběhne stopa z s

val $\text{enabled} : \text{State} \rightarrow \text{Set}\langle \text{Process} \rangle$

var $\text{Sleep} : \text{Trace} \rightarrow \text{Set}\langle \text{Trace} \rangle$

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$ ▷ Iniciální stav

val $s[_] : \text{Trace} \rightarrow \text{State}$ ▷ Stav, do něhož doběhne stopa z s

val $\text{enabled} : \text{State} \rightarrow \text{Set}\langle \text{Process} \rangle$

var $\text{Sleep} : \text{Trace} \rightarrow \text{Set}\langle \text{Trace} \rangle$ ▷ Navazující stopy jež netřeba řešit

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$ ▷ Iniciální stav

val $s[_] : \text{Trace} \rightarrow \text{State}$ ▷ Stav, do něhož doběhne stopa z s

val $\text{enabled} : \text{State} \rightarrow \text{Set}\langle \text{Process} \rangle$

var $\text{Sleep} : \text{Trace} \rightarrow \text{Set}\langle \text{Trace} \rangle$ ▷ Navazující stopy jež netřeba řešit

var $\text{backtrack} : \text{Trace} \rightarrow \text{Set}\langle \text{Process} \rangle$

Notace

val $\text{Event} \leftarrow \text{Process} \times \mathbb{N}$

val $s : \text{State}$ ▷ Iniciální stav

val $s[_] : \text{Trace} \rightarrow \text{State}$ ▷ Stav, do něhož doběhne stopa z s

val $\text{enabled} : \text{State} \rightarrow \text{Set}\langle \text{Process} \rangle$

var $\text{Sleep} : \text{Trace} \rightarrow \text{Set}\langle \text{Trace} \rangle$ ▷ Navazující stopy jež netřeba řešit

var $\text{backtrack} : \text{Trace} \rightarrow \text{Set}\langle \text{Process} \rangle$

val $\text{dom} : \text{Trace} \rightarrow \text{Set}\langle \text{Event} \rangle$

Source DPOR (1)

```
function ExploreSource( $E : \text{Trace}$ ,  $Sleep : \text{Set}<\text{Trace}>$ )  
   $sleep(E) \leftarrow Sleep$   
  choose process  $p \in enabled(s[e]) \setminus Sleep$  or return  
   $backtrack(E) \leftarrow \{p\};$   
  while  $\exists p \in backtrack(E) \setminus sleep(E)$  do  
    DetectRacesSource( $E, p$ )  
     $Sleep' \leftarrow \{v \mid v \in sleep(E) \wedge E \models p \diamond v\}$   
    Explore( $E.p, Sleep'$ )  
     $sleep(E) \leftarrow sleep(E) \cup \{p\}$   
  end while  
end function
```

Source DPOR (2)

```

function DetectRacesSource( $E$  : Trace,  $p$ : Process)
  val  $e_p$  : Event  $\leftarrow next_E(p)$ 
  for all  $e \in dom(E)$  such that  $e$  is in reversible race with  $e_p$  do
    val  $E' \leftarrow prefixBefore(E, e)$  ▷ does not include  $e$ 
    val  $v$  : Trace  $\leftarrow indepSuffixAfter(e, E).p$  ▷ does not include  $e$ 
    if the first event of  $v$  is not in  $backtrack(E')$  then
      add it there
    end if
  end for
end function
  
```

Source DPOR (2)

```

function DetectRacesSource( $E$  : Trace,  $p$ : Process)
  val  $e_p$  : Event  $\leftarrow next_E(p)$ 
  for all  $e \in dom(E)$  such that  $e$  is in reversible race with  $e_p$  do
    val  $E' \leftarrow prefixBefore(E, e)$  ▷ does not include  $e$ 
    val  $v$  : Trace  $\leftarrow indepSuffixAfter(e, E).p$  ▷ does not include  $e$ 
    if  $I_{E'}(v) \cap backtrack(E') \neq \emptyset$  then
      add some  $q' \in I_{E'}(v)$  to  $backtrack(E')$ 
    end if
  end for
end function
  
```

Výraz $I_{E'}(v)$ označuje procesy, které mají v posloupnosti v nějakou událost, která nenastává po (ve smyslu happens-before) žádné jiné události ve v .

Source DPOR - příklad

```
p: write(x);
q: read(y); read(x);
r: read(z); read(x);
```

Source DPOR - příklad

```
p: write(x);
q: read(y); read(x);
r: read(z); read(x);
```

Na tabuli

Algoritmus

```

function ExploreCS( $E$  : Trace,  $Sleep$  : Set<Trace>)
     $sleep(E) \leftarrow Sleep$ 
    choose process  $p \in enabled(s[e]) \setminus Sleep$  or return
     $backtrack(E) \leftarrow \{p\}$ ;
    while  $\exists p \in backtrack(E) \setminus sleep(E)$  do
        DetectRaces( $E, p$ )
         $Sleep' \leftarrow \{v \mid v \in sleep(E) \wedge E \models p \diamond v\}$ 
         $Sleep' \leftarrow Sleep' \cup \{v \mid p.v \in sleep(E)\}$ 
        Explore( $E.p, Sleep'$ )
         $sleep(E) \leftarrow sleep(E) \cup \{p\}$ 
    end while
end function
    
```

DetectRaces

```

function DetectRacesCS( $E$  : Trace,  $p$ : Process)
  val  $e_p$  : Event  $\leftarrow next_E(p)$ 
  for all  $e \in dom(E)$  such that  $e$  is in reversible race with  $e_p$  do
    val  $E' \leftarrow prefixBefore(E, e)$  ▷ does not include  $e$ 
    val  $v$  : Trace  $\leftarrow indepSuffixAfter(e, E).p$  ▷ does not include  $e$ 
    if the first event of  $v$  is not in  $backtrack(E')$  then
      add it there
    end if
    suspendSomething( $E, p, e, E', v$ )
  end for
end function
  
```


DetectRaces

```

function DetectRacesCS( $E$  : Trace,  $p$ : Process)
  val  $e_p$  : Event  $\leftarrow next_E(p)$ 
  for all  $e \in dom(E)$  such that  $e$  is in reversible race with  $e_p$  do
    val  $E' \leftarrow prefixBefore(E, e)$  ▷ does not include  $e$ 
    val  $v$  : Trace  $\leftarrow indepSuffixAfter(e, E).p$  ▷ does not include  $e$ 
    if  $I_{E'}(v) \cap backtrack(E') \neq \emptyset$  then
      add some  $q' \in I_{E'}(v)$  to  $backtrack(E')$ 
    end if
    suspendSomething( $E, p, e, E', v$ )
  end for
end function
  
```

Výraz $I_{E'}(v)$ označuje procesy, které mají v posloupnosti v nějakou událost, která nenastává po (ve smyslu happens-before) žádné jiné události ve v .

suspendSomething

function suspendSomething(E , p , e : Event, E' : Trace, v : Trace)

val $u \leftarrow \text{depSuffixFrom}(e, E)$

if $s[E.p] = s[E'.v.u]$ **then**

▷ commutativity

$\text{sleep}(E) \leftarrow \text{sleep}(E) \cup \{v.u\}$

end if

end function

suspendSomething

```
function suspendSomething( $E, p, e : \text{Event}, E' : \text{Trace}, v : \text{Trace}$ )  
  val  $u \leftarrow \text{depSuffixFrom}(e, E)$   
  if  $\nexists w \in \text{sleep}(E')$  where  $w \leq v.u$  then                                 $\triangleright$  redundancy check  
    if  $s[E.p] = s[E'.v.u]$  then                                            $\triangleright$  commutativity  
       $\text{sleep}(E) \leftarrow \text{sleep}(E) \cup \{v.u\}$   
    end if  
  end if  
end function
```



Relace happened-before



Proč to funguje

Proč to funguje

Definice

Mazurkiewiczova stopa je happened-before relace úplné výpočetní posloupnosti.

Proč to funguje

Definice

Mazurkiewiczova stopa je happened-before relace úplné výpočetní posloupnosti.

Tvrzení

Pro každou Mazurkiewiczovu stopu T , $\text{ExploreSourcePOR}(\epsilon, \emptyset)$ prozkoumá nějakou výpočetní sekvenci E , která stopu T implementuje.

Proč to funguje

Definice

Mazurkiewiczova stopa je happened-before relace úplné výpočetní posloupnosti.

Tvrzení

Pro každou Mazurkiewiczovu stopu T , $\text{ExploreSourcePOR}(\epsilon, \emptyset)$ prozkoumá nějakou výpočetní sekvenci E , která stopu T implementuje.

Tvrzení

Pro každou Mazurkiewiczovu stopu T , $\text{ExploreCS}(\epsilon, \emptyset)$ prozkoumá nějakou výpočetní sekvenci E , která buď implementuje stopu T , nebo dosáhne stejného stavu jako nějaká jiná sekvence E' , která stopu T implementuje.