FAKULTA
INFORMATIKY
Masarykova univerzita

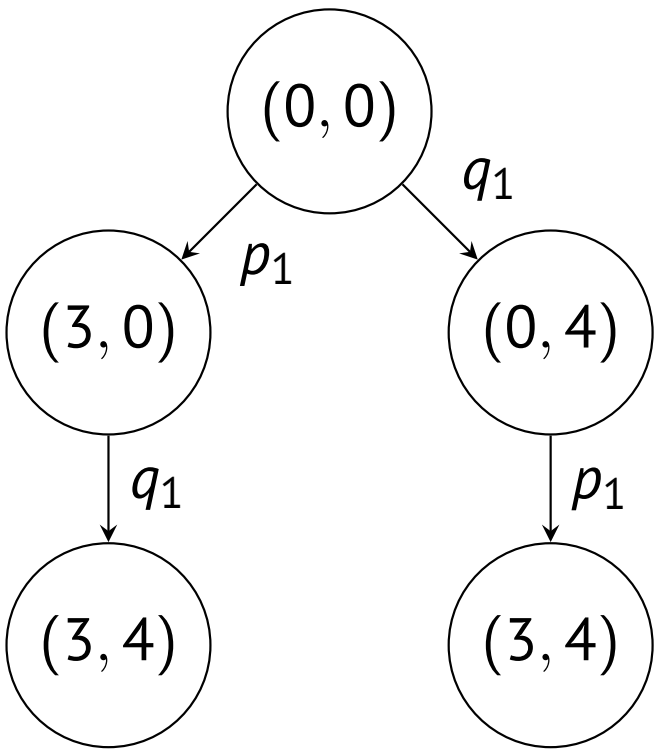
Context-Sensitive Dynamic Partial Order Reduction

Jan Tušil

26. března 2018

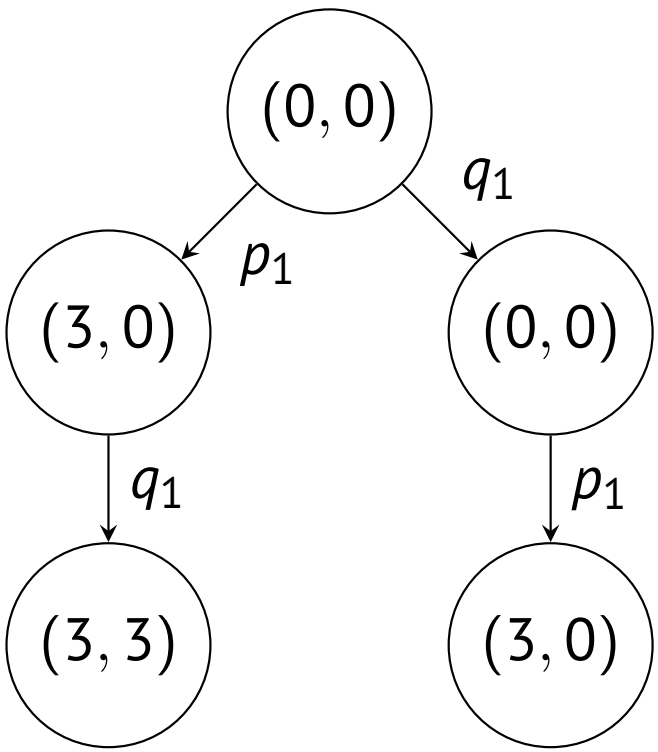
Example 1

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



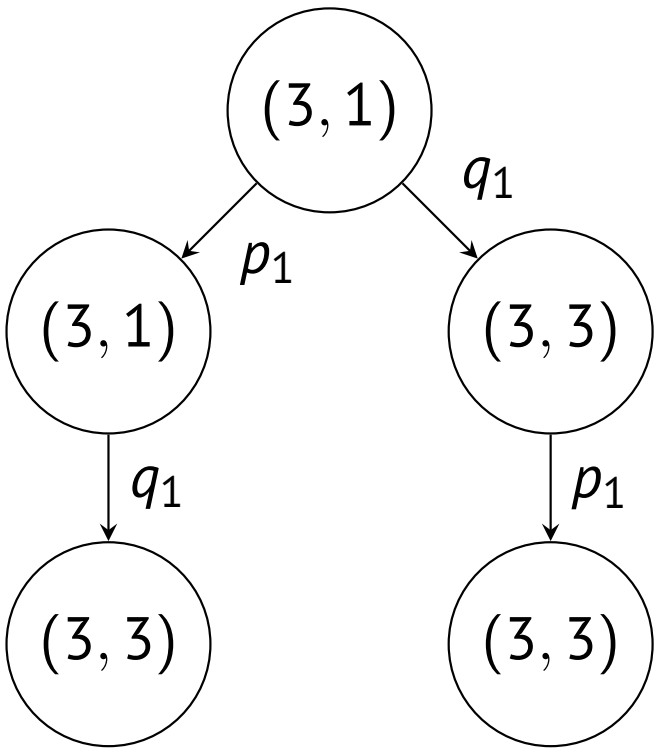
Example 2

```
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 Event $\leftarrow$ Process $\times\mathbb{N}$

Notace

val Event $\leftarrow$ Process $\times\mathbb{N}$
val s : State

Notace

val Event $\leftarrow$ Process $\times \mathbb{N}$
val s : State ▷ Iniciální stav

Notace

val Event $\leftarrow$ Process $\times \mathbb{N}$
val s : State ▷ Iniciální stav
val s[_] : Trace -> State

Notace

val Event $\leftarrow$ Process $\times\mathbb{N}$
val s : State ▷ Iniciální stav
val s[_] : Trace -> State ▷ Stav, do něhož doběhne stopa z s

Notace

val Event $\leftarrow$ Process $\times \mathbb{N}$
val s : State ▷ Iniciální stav
val s[_] : Trace -> State ▷ Stav, do něhož doběhne stopa z s
val enabled : State -> Set<Process>

Notace

val Event $\leftarrow$ Process $\times \mathbb{N}$
val s : State ▷ Iniciální stav
val s[_] : Trace -> State ▷ Stav, do něhož doběhne stopa z s
val enabled : State -> Set<Process>
var Sleep : Trace -> Set<Trace>

Notace

val Event $\leftarrow$ Process $\times\mathbb{N}$
val s : State ▷ Iniciální stav
val s[_] : Trace -> State ▷ Stav, do něhož doběhne stopa z s
val enabled : State -> Set<Process>
var Sleep : Trace -> Set<Trace> ▷ Navazující stopy jež netřeba řešit

Notace

val Event $\leftarrow$ Process $\times\mathbb{N}$
val s : State ▷ Iniciální stav
val s[_] : Trace -> State ▷ Stav, do něhož doběhne stopa z s
val enabled : State -> Set<Process>
var Sleep : Trace -> Set<Trace> ▷ Navazující stopy jež netřeba řešit
var backtrack : Trace -> Set<Process>

Notace

val Event $\leftarrow$ Process $\times\mathbb{N}$
val s : State ▷ Iniciální stav
val s[_] : Trace -> State ▷ Stav, do něhož doběhne stopa z s
val enabled : State -> Set<Process>
var Sleep : Trace -> Set<Trace> ▷ Navazující stopy jež netřeba řešit
var backtrack : Trace -> Set<Process>
val dom : Trace -> Set<Event>

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)
        $Sleep' \leftarrow \{v \mid v \in sleep(E) \wedge E \vDash p \diamond v\}$
        $Sleep' \leftarrow Sleep' \cup \{v \mid p.v \in sleep(E)\}$
        Explore($E.p$, $Sleep'$)
        $sleep(E) \leftarrow sleep(E) \cup \{p\}$
    end while
end function

DetectRaces

function DetectRaces(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 $\leftarrow$ $indepSuffixAfter(e, P).p$ ▷ does not include e
        if the first event of v is not in $backtrack(E')$ **then**
            add it there
        end if
        sleepSomething(E, p, e, E', v)
    end for
end function

DetectRaces

function DetectRaces(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 $\leftarrow$ $indepSuffixAfter(e, P).p$ ▷ 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
    sleepSomething(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.

sleepSomething

function sleepSomething(E, p, e : Event, E' : Trace, v : Trace)
    val $u \leftarrow depSuffixFrom(e, E)$

        if $s[E.p] = s[E'.v.u]$ **then** ▷ commutativity
            $sleep(E) \leftarrow sleep(E) \cup \{v.u\}$
        end if

end function

sleepSomething

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

Relace happened-before