Paradigmata programování 4

Přednáška 4. Kritická sekce

verze z 4. března 2025

Jan Laštovička

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

Je program správně?

```
(let ((x 1))
  (co
    (set! x 0)
    {(= x x)}))
```

Je program správně?

```
(let ((x 1))
  (co
    (set! x 0)
    {(= x x)}))
```

- Teoreticky ano: `(= x x)` je ekvivalentní `t`

Je program správně?

```
(let ((x 1))
  (co
    (set! x 0)
    {(= x x)}))
```

- Teoreticky ano: `(= x x)` je ekvivalentní `t`
- Prakticky ne: `(= x x)` se nevyhodnotí atomicky

Je program správně?

```
(let ((x 1))
  (co
    (set! x 0)
    {(= x x)}))
```

- Teoreticky ano: `(= x x)` je ekvivalentní `t`
- Prakticky ne: `(= x x)` se nevyhodnotí atomicky
- Řešení: podmínky tvrzení vyhodnocovat atomicky
- vede na změnu speciálního operátoru `assert`

Obsah

Musí program skončit?

let ((n 0) (flag t))	
p_1: (while flag p_2: (set! n (+ n 1)))	q_1: (set! flag nil)

Musí program skončit?

let ((n 0) (flag t))	
p_1: (while flag p_2: (set! n (+ n 1)))	q_1: (set! flag nil)

Nemusí. Uvažujme historii: p_1, p_2, p_1, p_2, ...

Musí program skončit?

let ((n 0) (flag t))	
p_1: (while flag p_2: (set! n (+ n 1)))	q_1: (set! flag nil)

Nemusí. Uvažujme historii: p_1, p_2, p_1, p_2, ...

Historie je (slabě) férová, pokud v každém jejím stavu platí:

- Jestliže se nějaký proces chystá vykonat příkaz, pak se dále v historii musí vykonat.

Musí program skončit?

let ((n 0) (flag t))	
p_1: (while flag p_2: (set! n (+ n 1)))	q_1: (set! flag nil)

Nemusí. Uvažujme historii: p_1, p_2, p_1, p_2, ...

Historie je (slabě) férová, pokud v každém jejím stavu platí:

- Jestliže se nějaký proces chystá vykonat příkaz, pak se dále v historii musí vykonat.

Plánovač je (slabě) férový, pokud

- historie všech programů jsou slabě férové.

Musí program skončit?

let ((n 0) (flag t))	
p_1: (while flag p_2: (set! n (+ n 1)))	q_1: (set! flag nil)

Nemusí. Uvažujme historii: p_1, p_2, p_1, p_2, ...

Historie je (slabě) férová, pokud v každém jejím stavu platí:

- Jestliže se nějaký proces chystá vykonat příkaz, pak se dále v historii musí vykonat.

Plánovač je (slabě) férový, pokud

- historie všech programů jsou slabě férové.

Je náš plánovač slabě férový?

Musí program skončit?

let ((n 0) (flag t))	
p_1: (while flag p_2: (set! n (+ n 1)))	q_1: (set! flag nil)

Nemusí. Uvažujme historii: p_1, p_2, p_1, p_2, ...

Historie je (slabě) férová, pokud v každém jejím stavu platí:

- Jestliže se nějaký proces chystá vykonat příkaz, pak se dále v historii musí vykonat.

Plánovač je (slabě) férový, pokud

- historie všech programů jsou slabě férové.

Je náš plánovač slabě férový? Ano.

Obsah

Vlastnosti programů

Vlastnost programu je

- výrok o historii programu.

Příklady:

- Proměnná `x` nemá nikdy hodnotu rovnu nule.
- Pokud `x` je nula, tak později `x` bude jedna.

Vlastnosti programů

Vlastnost programu je

- výrok o historii programu.

Příklady:

- Proměnná `x` nemá nikdy hodnotu rovnu nule.
- Pokud `x` je nula, tak později `x` bude jedna.

Program má jistou vlastnost, pokud

- je výrok pravdivý pro každou historii programu.

Vlastnosti programů

Vlastnost programu je

- výrok o historii programu.

Příklady:

- Proměnná `x` nemá nikdy hodnotu rovnu nule.
- Pokud `x` je nula, tak později `x` bude jedna.

Program má jistou vlastnost, pokud

- je výrok pravdivý pro každou historii programu.

Vlastnost bezpečnosti

- má tvar: „V žádném stavu historie neplatí T.“

Vlastnosti programů

Vlastnost programu je

- výrok o historii programu.

Příklady:

- Proměnná `x` nemá nikdy hodnotu rovnu nule.
- Pokud `x` je nula, tak později `x` bude jedna.

Program má jistou vlastnost, pokud

- je výrok pravdivý pro každou historii programu.

Vlastnost bezpečnosti

- má tvar: „V žádném stavu historie neplatí T.“

Vlastnost živosti

- má tvar: „Za každým stavem historie, který splňuje T_1, se vyskytuje stav splňující T_2.“

Obsah

Problém kritické sekce

- Chceme jisté příkazy vykonat atomicky
- bez použití složených atomických příkazů.

`let ((x 0))`	

`(let ((tmp nil))`
`  (set! tmp (+ x 1))`
`  (set! x tmp))`
	`(let ((tmp nil))`
	`  (set! tmp (+ x 1))`
	`  (set! x tmp))`

Problém kritické sekce

- Chceme jisté příkazy vykonat atomicky
- bez použití složených atomických příkazů.

```
                    let ((x 0))
(let ((tmp nil))                  (let ((tmp nil))
  (loop                             (loop
    (set! tmp (+ x 1))                (set! tmp (+ x 1))
    (set! x tmp)))                    (set! x tmp)))
```

Problém kritické sekce

- Chceme jisté příkazy vykonat atomicky
- bez použití složených atomických příkazů.

```
                        let ((x 0))
(let ((tmp nil))                    (let ((tmp nil))
  (loop                               (loop
    nekritická sekce                    nekritická sekce
    vstupní protokol                    vstupní protokol
    (set! tmp (+ x 1))                  (set! tmp (+ x 1))
    (set! x tmp)))                      (set! x tmp)))
    výstupní protokol))                 výstupní protokol))
```

Problém kritické sekce

- Chceme jisté příkazy vykonat atomicky
- bez použití složených atomických příkazů.

```
let ((x 0))
```

(loop	(loop
nekritická sekce	*nekritická sekce*
vstupní protokol	*vstupní protokol*
(set! x (+ x 1))	(set! x (+ x 1))
výstupní protokol)	*výstupní protokol*)

Problém kritické sekce

- Chceme jisté příkazy vykonat atomicky
- bez použití složených atomických příkazů.

loop
  nekritická sekce
  vstupní protokol
  kritická sekce
  výstupní protokol
	loop
	  nekritická sekce
	  vstupní protokol
	  kritická sekce
	  výstupní protokol

Problém kritické sekce

- Chceme jisté příkazy vykonat atomicky
- bez použití složených atomických příkazů.

loop	loop
nekritická sekce	*nekritická sekce*
vstupní protokol	*vstupní protokol*
kritická sekce	*kritická sekce*
výstupní protokol	*výstupní protokol*

1. Proces musí někdy opustit kritickou sekci.

Problém kritické sekce

- Chceme jisté příkazy vykonat atomicky
- bez použití složených atomických příkazů.

loop	loop
nekritická sekce	*nekritická sekce*
vstupní protokol	*vstupní protokol*
kritická sekce	*kritická sekce*
výstupní protokol	*výstupní protokol*

1. Proces musí někdy opustit kritickou sekci.
2. Proces nemusí opustit nekritickou sekci.

Problém kritické sekce

- Chceme jisté příkazy vykonat atomicky
- bez použití složených atomických příkazů.

loop	loop
nekritická sekce	*nekritická sekce*
vstupní protokol	*vstupní protokol*
kritická sekce	*kritická sekce*
výstupní protokol	*výstupní protokol*

1. Proces musí někdy opustit kritickou sekci.
2. Proces nemusí opustit nekritickou sekci.
3. Proces může v nekritické sekci skončit.

Vlastnosti kritické sekce

Vlastnosti kritické sekce

1. Vzájemné vyloučení.

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci.

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí.

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí. Pokud se oba procesy snaží vstoupit do kritické sekce, aspoň jeden uspěje.

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí. Pokud se oba procesy snaží vstoupit do kritické sekce, aspoň jeden uspěje. (vlastnost živosti)

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí. Pokud se oba procesy snaží vstoupit do kritické sekce, aspoň jeden uspěje. (vlastnost živosti)
3. Absence zbytečného čekání.

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí. Pokud se oba procesy snaží vstoupit do kritické sekce, aspoň jeden uspěje. (vlastnost živosti)
3. Absence zbytečného čekání. Pokud se proces snaží vstoupit do kritické sekce a druhý je v nekritické sekci nebo skončil, pak musí ihned uspět.

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí. Pokud se oba procesy snaží vstoupit do kritické sekce, aspoň jeden uspěje. (vlastnost živosti)
3. Absence zbytečného čekání. Pokud se proces snaží vstoupit do kritické sekce a druhý je v nekritické sekci nebo skončil, pak musí ihned uspět. (vlastnost bezpečnosti)

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí. Pokud se oba procesy snaží vstoupit do kritické sekce, aspoň jeden uspěje. (vlastnost živosti)
3. Absence zbytečného čekání. Pokud se proces snaží vstoupit do kritické sekce a druhý je v nekritické sekci nebo skončil, pak musí ihned uspět. (vlastnost bezpečnosti)
4. Zaručení vstupu.

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí. Pokud se oba procesy snaží vstoupit do kritické sekce, aspoň jeden uspěje. (vlastnost živosti)
3. Absence zbytečného čekání. Pokud se proces snaží vstoupit do kritické sekce a druhý je v nekritické sekci nebo skončil, pak musí ihned uspět. (vlastnost bezpečnosti)
4. Zaručení vstupu. Pokud se proces snaží vstoupit do kritické sekce, jednou musí uspět.

Vlastnosti kritické sekce

1. Vzájemné vyloučení. Nejvýše jeden proces je v kritické sekci. (vlastnost bezpečnosti)
2. Absence uváznutí. Pokud se oba procesy snaží vstoupit do kritické sekce, aspoň jeden uspěje. (vlastnost živosti)
3. Absence zbytečného čekání. Pokud se proces snaží vstoupit do kritické sekce a druhý je v nekritické sekci nebo skončil, pak musí ihned uspět. (vlastnost bezpečnosti)
4. Zaručení vstupu. Pokud se proces snaží vstoupit do kritické sekce, jednou musí uspět. (vlastnost živosti)

Náčrtek

loop
  nekritická sekce
  vstupní protokol
  kritická sekce
  výstupní protokol
	loop
	  nekritická sekce
	  vstupní protokol
	  kritická sekce
	  výstupní protokol

Náčrtek

let ((in1 nil) (in2 nil))	
loop *nekritická sekce* *vstupní protokol* *kritická sekce* *výstupní protokol*	loop *nekritická sekce* *vstupní protokol* *kritická sekce* *výstupní protokol*

Náčrtek

let ((in1 nil) (in2 nil))	
loop *nekritická sekce* (set! in1 t) *kritická sekce* (set! in1 nil)	loop *nekritická sekce* (set! in2 t) *kritická sekce* (set! in2 nil)

Náčrtek

let ((in1 nil) (in2 nil))	
loop *nekritická sekce* (set! in1 t) *kritická sekce* (set! in1 nil)	loop *nekritická sekce* (set! in2 t) *kritická sekce* (set! in2 nil)

Vzájemné vyloučení: $M =$ (not (and in1 in2))

Řešení nahrubo

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  nekritická sekce
  `(set! in1 t)`
  kritická sekce
  `(set! in1 nil)`
	`loop`
	  nekritická sekce
	  `(set! in2 t)`
	  kritická sekce
	  `(set! in2 nil)`

Řešení nahrubo

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
`nekritická sekce`
`(set! in1 t)`
`kritická sekce`
`(set! in1 nil)`
	`loop`
	`nekritická sekce`
	`(set! in2 t)`
	`kritická sekce`
	`(set! in2 nil)`

```
{?}
(set! in1 t)
{(not (and in1 in2))}
```

Řešení nahrubo

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
`nekritická sekce`
`(set! in1 t)`
`kritická sekce`
`(set! in1 nil)`
	`loop`
	`nekritická sekce`
	`(set! in2 t)`
	`kritická sekce`
	`(set! in2 nil)`

```
{(not (and t in2))}
(set! in1 t)
{(not (and in1 in2))}
```

Řešení nahrubo

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
`nekritická sekce`
`(set! in1 t)`
`kritická sekce`
`(set! in1 nil)`
	`loop`
	`nekritická sekce`
	`(set! in2 t)`
	`kritická sekce`
	`(set! in2 nil)`

```
{(not in2)}
(set! in1 t)
{(not (and in1 in2))}
```

Řešení nahrubo

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  `nekritická sekce`
  `(set! in1 t)`
  `kritická sekce`
  `(set! in1 nil)`
	`loop`
	  `nekritická sekce`
	  `(set! in2 t)`
	  `kritická sekce`
	  `(set! in2 nil)`

```
{(not in2)}
(set! in1 t)
{(not (and in1 in2))}
```

$\Rightarrow$

```
{t}
[(await (not in2))
 (set! in1 t)]
{(not (and in1 in2))}
```

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  `nekritická sekce`
  `[(await (not in2))`
  `(set! in1 t)]`
  `kritická sekce`
  `(set! in1 nil)`
	`loop`
	  `nekritická sekce`
	  `[(await (not in1))`
	  `(set! in2 t)]`
	  `kritická sekce`
	  `(set! in2 nil)`

Hrubé řešení

$M =$ (not (and in1 in2))

let ((in1 nil) (in2 nil))	
loop *nekritická sekce* {(and M (not in1))} [(await (not in2)) (set! in1 t)] {(and M in1)} *kritická sekce* (set! in1 nil) {(and M (not in1))}	loop *nekritická sekce* {(and M (not in2))} [(await (not in1)) (set! in2 t)] {(and M in2)} *kritická sekce* (set! in2 nil) {(and M (not in2))}

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
`  `*nekritická sekce*
`  {(and M (not in1))}`
`  [(await (not in2))`
`  (set! in1 t)]`
`  {(and M in1)}`
`  `*kritická sekce*
`  (set! in1 nil)`
`  {(and M (not in1))}`
	`loop`
	`  `*nekritická sekce*
	`  {(and M (not in2))}`
	`  [(await (not in1))`
	`  (set! in2 t)]`
	`  {(and M in2)}`
	`  `*kritická sekce*
	`  (set! in2 nil)`
	`  {(and M (not in2))}`

- Vzájemné vyloučení:

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
`  `*`nekritická sekce`*
`  {(and M (not in1))}`
`  [(await (not in2))`
`  (set! in1 t)]`
`  {(and M in1)}`
`  `*`kritická sekce`*
`  (set! in1 nil)`
`  {(and M (not in1))}`
	`loop`
	`  `*`nekritická sekce`*
	`  {(and M (not in2))}`
	`  [(await (not in1))`
	`  (set! in2 t)]`
	`  {(and M in2)}`
	`  `*`kritická sekce`*
	`  (set! in2 nil)`
	`  {(and M (not in2))}`

- Vzájemné vyloučení: `(not (and in1 in2))`

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  nekritická sekce
  `{(and M (not in1))}`
  `[(await (not in2))`
  `(set! in1 t)]`
  `{(and M in1)}`
  kritická sekce
  `(set! in1 nil)`
  `{(and M (not in1))}`
	`loop`
	  nekritická sekce
	  `{(and M (not in2))}`
	  `[(await (not in1))`
	  `(set! in2 t)]`
	  `{(and M in2)}`
	  kritická sekce
	  `(set! in2 nil)`
	  `{(and M (not in2))}`

- Vzájemné vyloučení: `(not (and in1 in2))`
- Absence uváznutí:

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  nekritická sekce
  `{(and` M `(not in1))}`
  `[(await (not in2))`
  `(set! in1 t)]`
  `{(and` M `in1)}`
  kritická sekce
  `(set! in1 nil)`
  `{(and` M `(not in1))}`
	`loop`
	  nekritická sekce
	  `{(and` M `(not in2))}`
	  `[(await (not in1))`
	  `(set! in2 t)]`
	  `{(and` M `in2)}`
	  kritická sekce
	  `(set! in2 nil)`
	  `{(and` M `(not in2))}`

- Vzájemné vyloučení: `(not (and in1 in2))`
- Absence uváznutí: `(and (not in1) in2 (not in2) in1)`

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  nekritická sekce
  `{(and M (not in1))}`
  `[(await (not in2))`
  `(set! in1 t)]`
  `{(and M in1)}`
  kritická sekce
  `(set! in1 nil)`
  `{(and M (not in1))}`
	`loop`
	  nekritická sekce
	  `{(and M (not in2))}`
	  `[(await (not in1))`
	  `(set! in2 t)]`
	  `{(and M in2)}`
	  kritická sekce
	  `(set! in2 nil)`
	  `{(and M (not in2))}`

- Vzájemné vyloučení: `(not (and in1 in2))`
- Absence uváznutí: `(and (not in1) in2 (not in2) in1)`
- Absence zbytečného čekání:

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
`  `*`nekritická sekce`*
`  {(and M (not in1))}`
`  [(await (not in2))`
`  (set! in1 t)]`
`  {(and M in1)}`
`  `*`kritická sekce`*
`  (set! in1 nil)`
`  {(and M (not in1))}`
	`loop`
	`  `*`nekritická sekce`*
	`  {(and M (not in2))}`
	`  [(await (not in1))`
	`  (set! in2 t)]`
	`  {(and M in2)}`
	`  `*`kritická sekce`*
	`  (set! in2 nil)`
	`  {(and M (not in2))}`

- Vzájemné vyloučení: `(not (and in1 in2))`
- Absence uváznutí: `(and (not in1) in2 (not in2) in1)`
- Absence zbytečného čekání: `(and (not in2) in2)` a `(and (not in1) in1)`

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  nekritická sekce
  `{(and M (not in1))}`
  `[(await (not in2))`
  `(set! in1 t)]`
  `{(and M in1)}`
  kritická sekce
  `(set! in1 nil)`
  `{(and M (not in1))}`
	`loop`
	  nekritická sekce
	  `{(and M (not in2))}`
	  `[(await (not in1))`
	  `(set! in2 t)]`
	  `{(and M in2)}`
	  kritická sekce
	  `(set! in2 nil)`
	  `{(and M (not in2))}`

- Vzájemné vyloučení: `(not (and in1 in2))`
- Absence uváznutí: `(and (not in1) in2 (not in2) in1)`
- Absence zbytečného čekání: `(and (not in2) in2)` a `(and (not in1) in1)`
- Zaručení vstupu:

Hrubé řešení

$M =$ `(not (and in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  nekritická sekce
  `{(and` M `(not in1))}`
  `[(await (not in2))`
  `(set! in1 t)]`
  `{(and` M `in1)}`
  kritická sekce
  `(set! in1 nil)`
  `{(and` M `(not in1))}`
	`loop`
	  nekritická sekce
	  `{(and` M `(not in2))}`
	  `[(await (not in1))`
	  `(set! in2 t)]`
	  `{(and` M `in2)}`
	  kritická sekce
	  `(set! in2 nil)`
	  `{(and` M `(not in2))}`

- Vzájemné vyloučení: `(not (and in1 in2))`
- Absence uváznutí: `(and (not in1) in2 (not in2) in1)`
- Absence zbytečného čekání: `(and (not in2) in2)` a `(and (not in1) in1)`
- Zaručení vstupu: Ne. Proč?

Zjednodušení

- `<->` ... logická spojka právě když
- `(<-> lock (or in1 in2))`

`let ((in1 nil) (in2 nil))`	

`loop`
  nekritická sekce
  `[(await (not in1))`
  `(set! in1 t)]`
  kritická sekce
  `(set! in1 nil)`
	`loop`
	  nekritická sekce
	  `[(await (not in2))`
	  `(set! in2 t)]`
	  kritická sekce
	  `(set! in2 nil)`

Zjednodušení

- <-> ... logická spojka právě když
- (<-> lock (or in1 in2))

let ((in1 nil) (in2 nil) (lock nil))	

loop
  nekritická sekce
  [(await (not in1))
  (set! in1 t)
  (set! lock t)]
  kritická sekce
  [(set! in1 nil)
  (set! lock nil)]
	loop
	  nekritická sekce
	  [(await (not in2))
	  (set! in2 t)
	  (set! lock t)]
	  kritická sekce
	  [(set! in2 nil)
	  (set! lock nil)]

Zjednodušení

- `<->` ... logická spojka právě když
- `(<-> lock (or in1 in2))`

`let ((in1 nil) (in2 nil) (lock nil))`	

`loop`
`  `*`nekritická sekce`*
`  [(await (not lock))`
`   (set! in1 t)`
`   (set! lock t)]`
`  `*`kritická sekce`*
`  [(set! in1 nil)`
`   (set! lock nil)]`
	`loop`
	`  `*`nekritická sekce`*
	`  [(await (not lock))`
	`   (set! in2 t)`
	`   (set! lock t)]`
	`  `*`kritická sekce`*
	`  [(set! in2 nil)`
	`   (set! lock nil)]`

Zjednodušení

- `<->` ... logická spojka právě když
- `(<-> lock (or in1 in2))`

`let ((lock nil))`	

`loop`
`  nekritická sekce`
`  [(await (not lock))`
`   (set! lock t)]`
`  kritická sekce`
`  (set! lock nil)`
	`loop`
	`  nekritická sekce`
	`  [(await (not lock))`
	`   (set! lock t)]`
	`  kritická sekce`
	`  (set! lock nil)`

Test-and-Set

Výraz:

`(ts` *var*`)`

Atomicky nastaví *var* na *Pravdu* a vrátí původní hodnotu *var*.

Jemné částečné řešení

let ((lock nil))	

loop
  nekritická sekce
  (while (ts lock)
  nil)
  kritická sekce
  (set! lock nil)
	loop
	  nekritická sekce
	  (while (ts lock)
	  nil)
	  kritická sekce
	  (set! lock nil)

Implementace obecných složených atomických příkazů

Implementace obecných složených atomických příkazů

- Lze použít libovolné řešení kritické sekce.

Implementace obecných složených atomických příkazů

- Lze použít libovolné řešení kritické sekce.
- Najdeme složené atomické příkazy používající stejné proměnné a

Implementace obecných složených atomických příkazů

- Lze použít libovolné řešení kritické sekce.
- Najdeme složené atomické příkazy používající stejné proměnné a
- hranaté závorky nahradíme vstupním a výstupním protokolem.

Implementace obecných složených atomických příkazů

- Lze použít libovolné řešení kritické sekce.
- Najdeme složené atomické příkazy používající stejné proměnné a
- hranaté závorky nahradíme vstupním a výstupním protokolem.

`let ((n 0))`	
`dotimes (i 100)` `  [(set! n (+ n 1))]`	`dotimes (i 100)` `  [(set! n (+ n 1))]`

Implementace obecných složených atomických příkazů

- Lze použít libovolné řešení kritické sekce.
- Najdeme složené atomické příkazy používající stejné proměnné a
- hranaté závorky nahradíme vstupním a výstupním protokolem.

```
let ((n 0) (lock nil))
```

```
dotimes (i 100)
  (while (ts lock)
    nil)
  (set! n (+ n 1))
  (set! lock nil)
```

```
dotimes (i 100)
  (while (ts lock)
    nil)
  (set! n (+ n 1))
  (set! lock nil)
```

Obsah

Můžeme zrušit atomičnost?

let ((in1 nil) (in2 nil))	
loop *nekritická sekce* [(await (not in2)) (set! in1 t)] *kritická sekce* (set! in1 nil)	loop *nekritická sekce* [(await (not in1)) (set! in2 t)] *kritická sekce* (set! in2 nil)

Můžeme zrušit atomičnost?

let ((in1 nil) (in2 nil))	
loop	loop
nekritická sekce	*nekritická sekce*
(await (not in2))	(await (not in1))
(set! in1 t)	(set! in2 t)
kritická sekce	*kritická sekce*
(set! in1 nil)	(set! in2 nil)

Můžeme zrušit atomičnost?

let ((in1 nil) (in2 nil))	
loop	loop
nekritická sekce	*nekritická sekce*
(await (not in2))	(await (not in1))
(set! in1 t)	(set! in2 t)
kritická sekce	*kritická sekce*
(set! in1 nil)	(set! in2 nil)

Ne. Je porušeno vzájemné vyloučení: `(not (and in1 in2))`

Pomůže prohodit příkazy?

let ((in1 nil) (in2 nil))	
loop	loop
nekritická sekce	*nekritická sekce*
(await (not in2))	(await (not in1))
(set! in1 t)	(set! in2 t)
kritická sekce	*kritická sekce*
(set! in1 nil)	(set! in2 nil)

Pomůže prohodit příkazy?

let ((in1 nil) (in2 nil))	
loop	loop
nekritická sekce	*nekritická sekce*
(set! in1 t)	(set! in2 t)
(await (not in2))	(await (not in1))
kritická sekce	*kritická sekce*
(set! in1 nil)	(set! in2 nil)

Pomůže prohodit příkazy?

let ((in1 nil) (in2 nil))	
loop	loop
nekritická sekce	*nekritická sekce*
(set! in1 t)	(set! in2 t)
(await (not in2))	(await (not in1))
kritická sekce	*kritická sekce*
(set! in1 nil)	(set! in2 nil)

Ne. Může dojít k uváznutí: `(and in2 in1)`

Kdo je na řadě?

let ((turn 1))	

loop
  nekritická sekce
  (await (= turn 1))
  kritická sekce
  (set! turn 2)
	loop
	  nekritická sekce
	  (await (= turn 2))
	  kritická sekce
	  (set! turn 1)

Kdo je na řadě?

let ((turn 1))	

loop
  nekritická sekce
  (await (= turn 1))
  kritická sekce
  (set! turn 2)
	loop
	  nekritická sekce
	  (await (= turn 2))
	  kritická sekce
	  (set! turn 1)

- Vzájemné vyloučení:

Kdo je na řadě?

`let ((turn 1))`	

`loop`
`  `*`nekritická sekce`*
`  (await (= turn 1))`
`  `*`kritická sekce`*
`  (set! turn 2)`
	`loop`
	`  `*`nekritická sekce`*
	`  (await (= turn 2))`
	`  `*`kritická sekce`*
	`  (set! turn 1)`

- Vzájemné vyloučení: `(and (= turn 1) (= turn 2))`

Kdo je na řadě?

`let ((turn 1))`	

`loop`
  `nekritická sekce`
  `(await (= turn 1))`
  `kritická sekce`
  `(set! turn 2)`
	`loop`
	  `nekritická sekce`
	  `(await (= turn 2))`
	  `kritická sekce`
	  `(set! turn 1)`

- Vzájemné vyloučení: `(and (= turn 1) (= turn 2))`
- Absence uváznutí:

Kdo je na řadě?

`let ((turn 1))`	

`loop`
  `nekritická sekce`
  `(await (= turn 1))`
  `kritická sekce`
  `(set! turn 2)`
	`loop`
	  `nekritická sekce`
	  `(await (= turn 2))`
	  `kritická sekce`
	  `(set! turn 1)`

- Vzájemné vyloučení: `(and (= turn 1) (= turn 2))`
- Absence uváznutí: `(and (not (= turn 1)) (not (= turn 2)))`

Kdo je na řadě?

`let ((turn 1))`	

`loop`
  `nekritická sekce`
  `(await (= turn 1))`
  `kritická sekce`
  `(set! turn 2)`
	`loop`
	  `nekritická sekce`
	  `(await (= turn 2))`
	  `kritická sekce`
	  `(set! turn 1)`

- Vzájemné vyloučení: `(and (= turn 1) (= turn 2))`
- Absence uváznutí: `(and (not (= turn 1)) (not (= turn 2)))`
- Absence zbytečného čekání:

Kdo je na řadě?

`let ((turn 1))`	

`loop`
  `nekritická sekce`
  `(await (= turn 1))`
  `kritická sekce`
  `(set! turn 2)`
	`loop`
	  `nekritická sekce`
	  `(await (= turn 2))`
	  `kritická sekce`
	  `(set! turn 1)`

- Vzájemné vyloučení: `(and (= turn 1) (= turn 2))`
- Absence uváznutí: `(and (not (= turn 1)) (not (= turn 2)))`
- Absence zbytečného čekání: Ne.

Kdo je na řadě?

`let ((turn 1))`	

`loop`
  `nekritická sekce`
  `(await (= turn 1))`
  `kritická sekce`
  `(set! turn 2)`
	`loop`
	  `nekritická sekce`
	  `(await (= turn 2))`
	  `kritická sekce`
	  `(set! turn 1)`

- Vzájemné vyloučení: `(and (= turn 1) (= turn 2))`
- Absence uváznutí: `(and (not (= turn 1)) (not (= turn 2)))`
- Absence zbytečného čekání: Ne.
- Zaručení vstupu:

Kdo je na řadě?

`let ((turn 1))`	

`loop`
`  `*`nekritická sekce`*
`  (await (= turn 1))`
`  `*`kritická sekce`*
`  (set! turn 2)`
	`loop`
	`  `*`nekritická sekce`*
	`  (await (= turn 2))`
	`  `*`kritická sekce`*
	`  (set! turn 1)`

- Vzájemné vyloučení: `(and (= turn 1) (= turn 2))`
- Absence uváznutí: `(and (not (= turn 1)) (not (= turn 2)))`
- Absence zbytečného čekání: Ne.
- Zaručení vstupu: Ne.

Obsah

Petersonův algoritmus nahrubo

let ((in1 nil) (in2 nil))	
loop	loop
nekritická sekce	*nekritická sekce*
(set! in1 t)	(set! in2 t)
(await (not in2))	(await (not in1))
kritická sekce	*kritická sekce*
(set! in1 nil)	(set! in2 nil)

Petersonův algoritmus nahrubo

let ((in1 nil) (in2 nil) (last 1))	
loop	loop
nekritická sekce	*nekritická sekce*
(set! in1 t)	(set! in2 t)
(await (not in2))	(await (not in1))
kritická sekce	*kritická sekce*
(set! in1 nil)	(set! in2 nil)

Petersonův algoritmus nahrubo

let ((in1 nil) (in2 nil) (last 1))	
loop	loop
nekritická sekce	*nekritická sekce*
(set! in1 t)	(set! in2 t)
(set! last 1)	(set! last 2)
(await (not in2))	(await (not in1))
kritická sekce	*kritická sekce*
(set! in1 nil)	(set! in2 nil)

Petersonův algoritmus nahrubo

```
let ((in1 nil) (in2 nil) (last 1))
```

```
loop                                  loop
  nekritická sekce                      nekritická sekce
  (set! in1 t)                          (set! in2 t)
  (set! last 1)                         (set! last 2)
  [(await (or (not in2)                 [(await (or (not in1)
              (= last 2)))]                         (= last 1)))]
  kritická sekce                        kritická sekce
  (set! in1 nil)                        (set! in2 nil)
```

Petersonův algoritmus nahrubo

```
let ((in1 nil) (in2 nil) (last 1))
```

```
loop                                  loop
  nekritická sekce                      nekritická sekce
  (set! in1 t)                          (set! in2 t)
  (set! last 1)                         (set! last 2)
  [(await (or (not in2)                 [(await (or (not in1)
              (= last 2)))]                         (= last 1)))]
  kritická sekce                        kritická sekce
  (set! in1 nil)                        (set! in2 nil)
```

- Máme zarušení vstupu.

Petersonův algoritmus najemno

```
let ((in1 nil) (in2 nil) (last 1))
```

```
loop                                  loop
  nekritická sekce                      nekritická sekce
  (set! in1 t)                          (set! in2 t)
  (set! last 1)                         (set! last 2)
  [(await (or (not in2)                 [(await (or (not in1)
              (= last 2)))]                         (= last 1)))]
  kritická sekce                        kritická sekce
  (set! in1 nil)                        (set! in2 nil)
```

Petersonův algoritmus najemno

```
let ((in1 nil) (in2 nil) (last 1))
```

```
loop                                  loop
  nekritická sekce                      nekritická sekce
  (set! in1 t)                          (set! in2 t)
  (set! last 1)                         (set! last 2)
  [(await (or (not in2)                 [(await (or (not in1)
              (= last 2)))]                         (= last 1)))]
  kritická sekce                        kritická sekce
  (set! in1 nil)                        (set! in2 nil)
```

- Podmínky v `await` nejsou atomické.

Petersonův algoritmus najemno

```
let ((in1 nil) (in2 nil) (last 1))
```

```
loop                                  loop
  nekritická sekce                      nekritická sekce
  (set! in1 t)                          (set! in2 t)
  (set! last 1)                         (set! last 2)
  [(await (or (not in2)                 [(await (or (not in1)
              (= last 2)))]                         (= last 1)))]
  kritická sekce                        kritická sekce
  (set! in1 nil)                        (set! in2 nil)
```

- Podmínky v `await` nejsou atomické.
- V tomto případě můžeme atomičnost zrušit.

Petersonův algoritmus najemno

```
let ((in1 nil) (in2 nil) (last 1))
```

```
loop                              loop
  nekritická sekce                  nekritická sekce
  (set! in1 t)                      (set! in2 t)
  (set! last 1)                     (set! last 2)
  (await (or (not in2)              (await (or (not in1)
             (= last 2)))                      (= last 1)))
  kritická sekce                    kritická sekce
  (set! in1 nil)                    (set! in2 nil)
```

- Podmínky v `await` nejsou atomické.
- V tomto případě můžeme atomičnost zrušit. Proč?

Důkaz vzájemného vyloučení

```
let ((in1 nil) (in2 nil) (last 1) (mid1 nil) (mid2 nil))
```

```
loop                                loop
  nekritická sekce                    nekritická sekce
  [(set! in1 t)                       [(set! in2 t)
   (set! mid1 t)]                      (set! mid2 t)]
  [(set! last 1)                      [(set! last 2)
   (set! mid1 nil)]                    (set! mid2 nil)]
  (await (or (not in2)                (await (or (not in1)
             (= last 2)))                        (= last 1)))
  kritická sekce                      kritická sekce
  (set! in1 nil)                      (set! in2 nil)
```

Důkaz vzájemného vyloučení

```
let ((in1 nil) (in2 nil) (last 1) (mid1 nil) (mid2 nil))
```

```
loop                                  loop
  nekritická sekce                      nekritická sekce
  [(set! in1 t)                         [(set! in2 t)
   (set! mid1 t)]                        (set! mid2 t)]
  [(set! last 1)                        [(set! last 2)
   (set! mid1 nil)]                      (set! mid2 nil)]
  (await (or (not in2)                  (await (or (not in1)
             (= last 2)))                          (= last 1)))
  kritická sekce                        kritická sekce
  (set! in1 nil)                        (set! in2 nil)
```

- první je v kritické sekci: `(and in1 (not mid1) (or (not in2) (= last 2) mid2))`

Důkaz vzájemného vyloučení

```
let ((in1 nil) (in2 nil) (last 1) (mid1 nil) (mid2 nil))
```

loop
  nekritická sekce
  [(set! in1 t)
  (set! mid1 t)]
  [(set! last 1)
  (set! mid1 nil)]
  (await (or (not in2)
  (= last 2)))
  kritická sekce
  (set! in1 nil)
	loop
	  nekritická sekce
	  [(set! in2 t)
	  (set! mid2 t)]
	  [(set! last 2)
	  (set! mid2 nil)]
	  (await (or (not in1)
	  (= last 1)))
	  kritická sekce
	  (set! in2 nil)

- první je v kritické sekci: `(and in1 (not mid1) (or (not in2) (= last 2) mid2))`
- druhý je v kritické sekci: `(and in2 (not mid2) (or (not in1) (= last 1) mid1))`

Důkaz vzájemného vyloučení

```
let ((in1 nil) (in2 nil) (last 1) (mid1 nil) (mid2 nil))
```

loop
  nekritická sekce
  [(set! in1 t)
  (set! mid1 t)]
  [(set! last 1)
  (set! mid1 nil)]
  (await (or (not in2)
  (= last 2)))
  kritická sekce
  (set! in1 nil)
	loop
	  nekritická sekce
	  [(set! in2 t)
	  (set! mid2 t)]
	  [(set! last 2)
	  (set! mid2 nil)]
	  (await (or (not in1)
	  (= last 1)))
	  kritická sekce
	  (set! in2 nil)

- první je v kritické sekci: `(and in1 (not mid1) (or (not in2) (= last 2) mid2))`
- druhý je v kritické sekci: `(and in2 (not mid2) (or (not in1) (= last 1) mid1))`
- podmínky se vylučují (jejich konjunkce je kontradikce)