PSC 2020/21 (375AA, 9CFU)

Principles for Software Composition

Roberto Bruni
http://www.di.unipi.it/~bruni/

http://didawiki.di.unipi.it/doku.php/magistraleinformatica/psc/

Exercises #7

Temporal and modal logics

[**Ex. 1**] Two processes p_1 and p_2 want to access a single shared resource r. Consider the atomic propositions:

req_i: holds when process p_i is requesting access to r;
use_i: holds when process p_i has had access to r;
rel_i: holds when process p_i has released r.

with $i \in [1, 2]$. Use LTL formulas to specify the following properties:

1. *mutual exclusion*: r is accessed by only one process at a time;
2. *release*: every time p_1 accesses r, it releases r after some time;
3. *priority*: whenever both p_1 and p_2 require r, p_1 is granted access first;
4. *no starvation*: whenever p_1 requires r, it is eventually granted access.

Ex. 1, LTL

req_i — p_i requests r

use_i — p_i has access to r

rel_i — p_i releases r

mutual exclusion $\mathsf{G}\ \neg(use_1 \wedge use_2)$

release $\mathsf{G}\ (use_1 \Rightarrow \mathsf{F}\ rel_1)$

priority $\mathsf{G}\ ((req_1 \wedge req_2) \Rightarrow ((\neg use_2)\ \mathsf{U}\ (use_1 \wedge \neg use_2)))$

no starvation $\mathsf{G}\ (\mathrm{req}_1 \Rightarrow \mathsf{F}\ use_1)$

1. *mutual exclusion*: r is accessed by only one process at a time;
2. *release*: every time p_1 accesses r, it releases r after some time;
3. *priority*: whenever both p_1 and p_2 require r, p_1 is granted access first;
4. *no starvation*: whenever p_1 requires r, it is eventually granted access.

[Ex. 2] Three dogs live in a house with two couches and a front garden. Let $couch_{i,j}$ represent the predicate “the dog i sits on couch j” and $garden_i$ represent the predicate “the dog i plays in the front garden”.

1. Write an LTL formula expressing the fact that whenever dog 1 plays in the garden then he keeps playing until he sits on some couch (but he may also play forever).

2. Write a CTL formula expressing the fact that dog 2 eventually plays in the garden whenever couch 1 is occupied by another dog.

3. Write a μ-calculus formula expressing the fact that no more than one couch is occupied at any time by dog 3.

Ex. 2, logics

$couch_{i,j}$ — i sits on j

$garden_i$ — i plays in the garden

LTL

$$\mathsf{G}\ (garden_1 \Rightarrow ((\mathsf{G}\ garden_1) \vee (garden_1\ \mathsf{U}\ (couch_{1,1} \vee couch_{1,2}))))$$

CTL

$$\mathsf{AG}\ ((couch_{1,1} \vee couch_{3,1}) \Rightarrow \mathsf{AF}\ garden_2)$$

μ-calculus

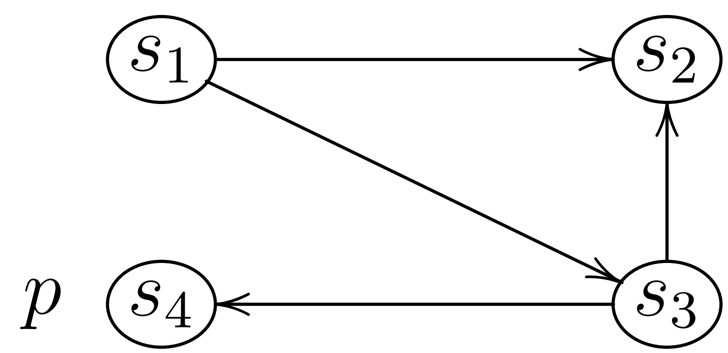
$$\nu x.\ ((\neg couch_{3,1} \vee \neg couch_{3,2}) \wedge \Box x)$$

1. Write an LTL formula expressing the fact that whenever dog 1 plays in the garden then he keeps playing until he sits on some couch (but he may also play forever).

2. Write a CTL formula expressing the fact that dog 2 eventually plays in the garden whenever couch 1 is occupied by another dog.

3. Write a μ-calculus formula expressing the fact that no more than one couch is occupied at any time by dog 3.

[Ex. 3] Given the μ-calculus formula $\Phi = \mu x.((p \wedge \Box x) \vee (\neg p \wedge \Diamond x))$ write its denotational semantics $\llbracket \Phi \rrbracket \rho$ and evaluate it on the LTS below (where $V = \{s_1, s_2, s_3, s_4\}$ and $P = \{p\}$).

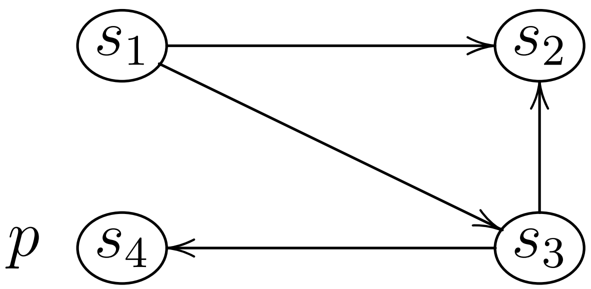


Ex. 3, mu-calculus

$\rho(p) = \{s_4\}$ $\qquad\qquad$ $\overline{\rho(p)} = \{s_1, s_2, s_3\}$

$$
\begin{aligned}
&[\![\mu x.((p \wedge \Box x) \vee (\neg p \wedge \Diamond x))]\!]\rho \\
&\triangleq \mathit{fix}\ \lambda S.\ [\![(p \wedge \Box x) \vee (\neg p \wedge \Diamond x)]\!]\rho[{}^{S}/_{x}] \\
&= \mathit{fix}\ \lambda S.\ [\![p \wedge \Box x]\!]\rho[{}^{S}/_{x}] \cup [\![\neg p \wedge \Diamond x]\!]\rho[{}^{S}/_{x}] \\
&= \mathit{fix}\ \lambda S.\quad ([\![p]\!]\rho[{}^{S}/_{x}] \cap [\![\Box x]\!]\rho[{}^{S}/_{x}]) \cup \\
&\qquad\qquad\quad ([\![\neg p]\!]\rho[{}^{S}/_{x}] \cap [\![\Diamond x]\!]\rho[{}^{S}/_{x}]) \\
&= \mathit{fix}\ \lambda S.\quad (\rho(p) \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in [\![x]\!]\rho[{}^{S}/_{x}]\}) \cup \\
&\qquad\qquad\quad (\overline{\rho(p)} \cap \{v \mid \exists w \in [\![x]\!]\rho[{}^{S}/_{x}].\ v \to w\}) \\
&= \mathit{fix}\ \lambda S.\quad (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S\}) \cup \\
&\qquad\qquad\quad (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S.\ v \to w\})
\end{aligned}
$$

Ex. 3, mu-calculus



$$fix\ \lambda S.\quad (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S\}) \cup \\ (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S.\ v \to w\})$$

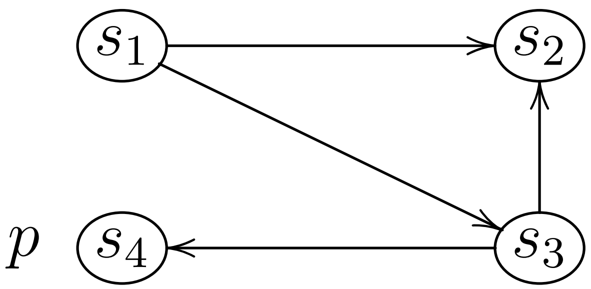
$$S_0 = \emptyset$$

$$\begin{aligned} S_1 = \quad & (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S_0\}) \cup \\ & (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S_0.\ v \to w\}) \\ = \quad & (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in \emptyset\}) \cup \\ & (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in \emptyset.\ v \to w\}) \\ = \quad & (\{s_4\} \cap \{s_2, s_4\}) \cup \\ & (\{s_1, s_2, s_3\} \cap \emptyset) \\ = \quad & \{s_4\} \end{aligned}$$

deadlock

emptyset

Ex. 3, mu-calculus



$$fix\ \lambda S.\quad (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S\}) \cup (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S.\ v \to w\})$$

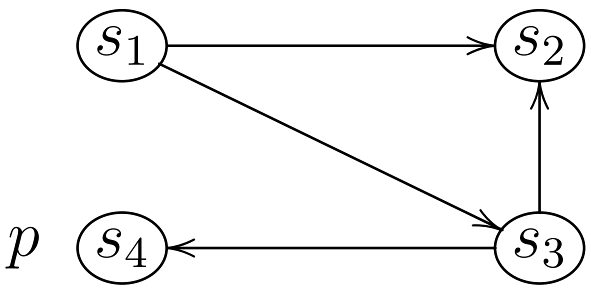
$$S_0 = \emptyset \qquad S_1 = \{s_4\}$$

deadlock or
can reach only s_4

$$\begin{aligned} S_2 = \quad & (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S_1\}) \cup \\ & (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S_1.\ v \to w\}) \\ = \quad & (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in \{s_4\}\}) \cup \\ & (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in \{s_4\}.\ v \to w\}) \\ = \quad & (\{s_4\} \cap \{s_2, s_4\}) \cup \\ & (\{s_1, s_2, s_3\} \cap \{s_3\}) \\ = \quad & \{s_3, s_4\} \end{aligned}$$

has a transition to s_4

Ex. 3, mu-calculus



$$\mathit{fix}\ \lambda S.\quad (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S\}) \cup (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S.\ v \to w\})$$

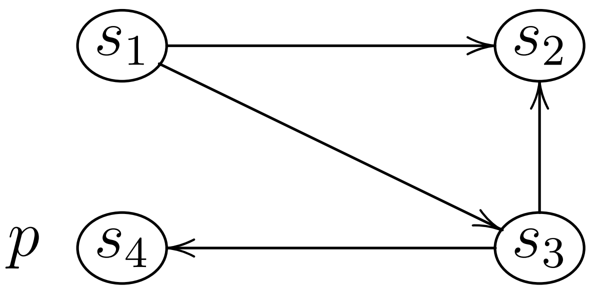
$$S_0 = \emptyset \qquad S_1 = \{s_4\} \qquad S_2 = \{s_3, s_4\}$$

deadlock or
can reach only s_3, s_4

$$\begin{aligned}
S_3 = \quad & (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S_2\}) \cup \\
& (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S_2.\ v \to w\}) \\
= \quad & (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in \{s_3, s_4\}\}) \cup \\
& (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in \{s_3, s_4\}.\ v \to w\}) \\
= \quad & (\{s_4\} \cap \{s_2, s_4\}) \cup \\
& (\{s_1, s_2, s_3\} \cap \{s_1, s_3\}) \\
= \quad & \{s_1, s_3, s_4\}
\end{aligned}$$

has a transition to s_3 or s_4

Ex. 3, mu-calculus



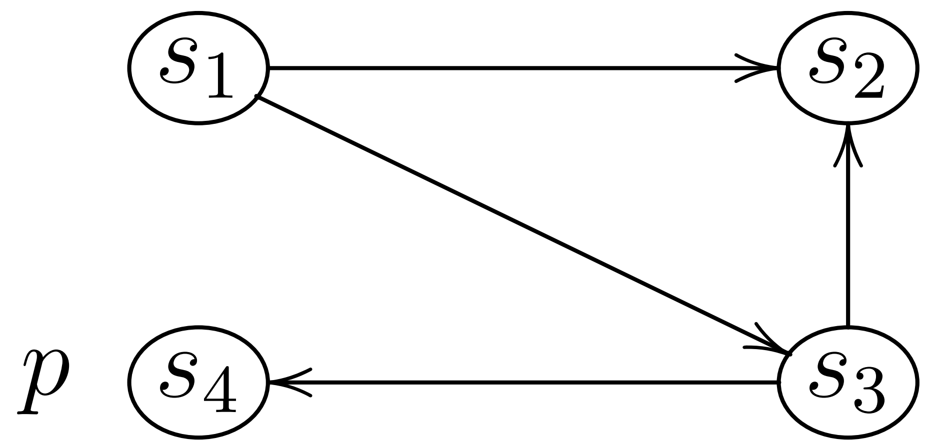
$$\mathit{fix}\ \lambda S.\quad (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S\}) \cup (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S.\ v \to w\})$$

$$S_0 = \emptyset \qquad S_1 = \{s_4\} \qquad S_2 = \{s_3, s_4\} \qquad S_3 = \{s_1, s_3, s_4\}$$

$$\begin{aligned}
S_4 = \quad & (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in S_3\}) \cup \\
& (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in S_3.\ v \to w\}) \\
= \quad & (\{s_4\} \cap \{v \mid \forall w.\ v \to w \Rightarrow w \in \{s_1, s_3, s_4\}\}) \cup \\
& (\{s_1, s_2, s_3\} \cap \{v \mid \exists w \in \{s_1, s_3, s_4\}.\ v \to w\}) \\
= \quad & (\{s_4\} \cap \{s_2, s_4\}) \cup \\
& (\{s_1, s_2, s_3\} \cap \{s_1, s_3\}) \\
= \quad & \{s_1, s_3, s_4\} = S_3
\end{aligned}$$

fixpoint reached!

Ex. 3, mu-calculus



$$[\![\mu x.((p \wedge \Box x) \vee (\neg p \wedge \Diamond x))]\!]\rho \;=\; \{s_1, s_3, s_4\}$$

Google Go

[**Ex. 4**] Write a GoogleGo function that takes one channel `ini` for receiving integers and one channel `ins` for receving strings and returns a channel `outp` where all the messages received on `ini` and `ins` will be paired.
Hint: define a ***`struct`*** *to form pairs*

Ex. 4, pairing in Go

```
package main

import "fmt"

type Pair struct {
    N int
    S string
}
```

Ex. 4, pairing in Go

```
package main

import "fmt"

type Pair struct {
    N int
    S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {







    return
}
```

Ex. 4, pairing in Go

```
package main

import "fmt"

type Pair struct {
   N int
   S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
   outp = make(chan Pair)









   return
}
```

Ex. 4, pairing in Go

```
package main

import "fmt"

type Pair struct {
   N int
   S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
   outp = make(chan Pair)
   go func() {
      for {




      }
   }()
   return
}
```

Ex. 4, pairing in Go

```
package main

import "fmt"

type Pair struct {
   N int
   S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
   outp = make(chan Pair)
   go func() {
      for {
         i := <-ini
         s := <-ins

      }
   }()
   return
}
```

Ex. 4, pairing in Go

```
package main

import "fmt"

type Pair struct {
   N int
   S string
}

func pairing(ini chan int, ins chan string) (outp chan Pair) {
   outp = make(chan Pair)
   go func() {
      for {
         i := <-ini
         s := <-ins
         outp <- Pair{i, s}
      }
   }()
   return
}
```

Ex. 4, pairing in Go

```
func main() {
    chi := make(chan int)
    chs := make(chan string)









}
```

Ex. 4, pairing in Go

```
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chp := pairing(chi,chs)









}
```

Ex. 4, pairing in Go

```
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chp := pairing(chi,chs)
    chi <- 1
    chs <- "Alice"







}
```

Ex. 4, pairing in Go

```
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chp := pairing(chi,chs)
    chi <- 1
    chs <- "Alice"
    v := <- chp
    fmt.Println("got",v)




}
```

Ex. 4, pairing in Go

```
func main() {
    chi := make(chan int)
    chs := make(chan string)
    chp := pairing(chi,chs)
    chi <- 1
    chs <- "Alice"
    v := <- chp
    fmt.Println("got",v)
    chi <- 2
    chs <- "Bob"
    v = <- chp
    fmt.Println("got",v)
}
```

[**Ex. 5**] Write a GoogleGo function that takes two channels `f` and `q` and tries to send the stream of Fibonacci numbers on `f` but quits when it receives `true` on channel `q`. Write a `main` program to test the function by printing the first 10 Fibonacci numbers.

Ex. 5, Go Fibonacci

```
package main

import "fmt"

func fibonacci(f chan int, q chan bool) {





}
```

Ex. 5, Go Fibonacci

```
package main

import "fmt"

func fibonacci(f chan int, q chan bool) {
    x, y := 0, 1
    for {







    }
}
```

Ex. 5, Go Fibonacci

```
package main

import "fmt"

func fibonacci(f chan int, q chan bool) {
    x, y := 0, 1
    for {
        select {




        }
    }
}
```

Ex. 5, Go Fibonacci

```
package main

import "fmt"

func fibonacci(f chan int, q chan bool) {
    x, y := 0, 1
    for {
        select {
        case f <- x:
            x, y = y, x+y


        }
    }
}
```

Ex. 5, Go Fibonacci

```
package main

import "fmt"

func fibonacci(f chan int, q chan bool) {
    x, y := 0, 1
    for {
        select {
        case f <- x:
            x, y = y, x+y
        case <-q:
            return
        }
    }
}
```

Ex. 5, Go Fibonacci

```
func main() {
   fib := make(chan int)
   quit := make(chan bool)




   fibonacci(fib, quit)
}
```

Ex. 5, Go Fibonacci

```
func main() {
   fib := make(chan int)
   quit := make(chan bool)
   go func(){




   }()
   fibonacci(fib, quit)
}
```

Ex. 5, Go Fibonacci

```
func main() {
    fib := make(chan int)
    quit := make(chan bool)
    go func(){
        for i := 0; i < 10; i++ {
            fmt.Println(<-fib)
        }

    }()
    fibonacci(fib, quit)
}
```

Ex. 5, Go Fibonacci

```
func main() {
   fib := make(chan int)
   quit := make(chan bool)
   go func(){
      for i := 0; i < 10; i++ {
         fmt.Println(<-fib)
      }
      quit <- true
   }()
   fibonacci(fib, quit)
}
```

π-calculus

[**Ex. 6**] The *asynchronous* π-calculus requires that outputs have no continuation:

$$p \ ::= \ \mathbf{nil} \ | \ \overline{x}\langle y\rangle \ | \ x(y).p \ | \ \tau.p \ | \ [x=y]p \ | \ p+p \ | \ p|p \ | \ (x)p \ | \ !p$$

Show that any process in the original π-calculus can be represented in the asynchronous π-calculus using an extra (fresh) channel to simulate explicit acknowledgement of name transmission.

Ex. 6, async pi

we want to encode ordinary processes in asynchronous ones

$$\mathcal{A}(\mathbf{nil}) \triangleq \mathbf{nil}$$

$$\boxed{\mathcal{A}(\overline{x}y.p) \triangleq \overline{x}\langle y\rangle|\mathcal{A}(p)}$$

$$\mathcal{A}(x(y).p) \triangleq x(y).\mathcal{A}(p)$$

$$\mathcal{A}(\tau.p) \triangleq \tau.\mathcal{A}(p)$$

$$\mathcal{A}([x = y]p) \triangleq [x = y]\mathcal{A}(p)$$

$$\mathcal{A}(p + q) \triangleq \mathcal{A}(p) + \mathcal{A}(q)$$

$$\mathcal{A}(p|q) \triangleq \mathcal{A}(p)|\mathcal{A}(q)$$

$$\mathcal{A}((x)p) \triangleq (x)\mathcal{A}(p)$$

$$\mathcal{A}(!p) \triangleq !\mathcal{A}(p)$$

Ex. 6, async pi

we want to encode ordinary processes in asynchronous ones

$$\boxed{\mathcal{A}(\overline{x}y.p) \triangleq \overline{x}\langle y\rangle|\mathcal{A}(p)}$$

$$\begin{array}{ccc} \mathcal{A}((x)(\overline{x}.x.p)) & \triangleq & (x)(\overline{x}|x.\mathcal{A}(p)) \\ \not\downarrow & & \downarrow \tau \\ \text{deadlock} & & (x)(\mathbf{nil}|\boxed{\mathcal{A}(p)}) \end{array}$$

problem: $\mathcal{A}(p)$ can be executed before $\overline{x}y$ is received

Ex. 6, async pi

2nd attempt: wait for an ack

$$\mathcal{A}(\overline{x}y.p) \triangleq \overline{x}\langle y\rangle \mid a\,.\,\mathcal{A}(p)$$
$$\mathcal{A}(x(z).q) \triangleq x(z)\,.\,(\overline{a} \mid \mathcal{A}(q))$$

$$\mathcal{A}((x,u)(\overline{x}|x|\overline{u}.p)) \triangleq (x,u)(\boxed{\overline{x}}|a.\mathbf{nil} \mid \boxed{x.}(\overline{a}|\mathbf{nil}) \mid \overline{u}|a.\mathcal{A}(p))$$

$$\downarrow \tau \qquad\qquad \downarrow \tau$$

$$(x,u)(\mathbf{nil}|\mathbf{nil}|\overline{u}.p) \qquad (x,u)(\mathbf{nil}|a.\mathbf{nil} \mid \boxed{\overline{a}}\,\mathbf{nil} \mid \overline{u}|\boxed{a.}\mathcal{A}(p))$$

deadlock

$$\downarrow \tau$$

$$(x,u)(\mathbf{nil}|a.\mathbf{nil} \mid \mathbf{nil}|\mathbf{nil} \mid \overline{u}|\boxed{\mathcal{A}(p)})$$

problem: possible interferences on channel a

Ex. 6, async pi

3rd attempt: wait for a private ack

$$\mathcal{A}(\overline{x}y.p) \triangleq (a)(\overline{x}\langle a\rangle \mid \boxed{\overline{a}}\langle y\rangle \mid \boxed{a.}\mathcal{A}(p)) \quad a, x_a \text{ fresh}$$

$$\mathcal{A}(x(z).q) \triangleq x(x_a)\,.\,x_a(z)\,.\,(\overline{x_a} \mid \mathcal{A}(q))$$

problem: possible self-interferences on channels a

Ex. 6, async pi

4th attempt: use 2 private channels

$$\mathcal{A}(\overline{x}y.p) \triangleq (a)(\overline{x}\langle a\rangle \mid a(x_k)\,.\,(\overline{x_k}\langle y\rangle \mid \mathcal{A}(p)))$$

$$\mathcal{A}(x(z).q) \triangleq x(x_a)\,.\,(k)(\overline{x_a}\langle k\rangle \mid k(z).\mathcal{A}(q))$$

a, k, x_a, x_k fresh

Ex. 6, async pi

4th attempt: use 2 private channels

$$\mathcal{A}(\overline{x}y.p) \triangleq (a)(\overline{x}\langle a\rangle \mid a(x_k).(\overline{x_k}\langle y\rangle \mid \mathcal{A}(p)))$$

$$\mathcal{A}(x(z).q) \triangleq x(x_a).(k)(\overline{x_a}\langle k\rangle \mid k(z).\mathcal{A}(q))$$

a, k, x_a, x_k fresh

problem: 3 communications involved, can we do better?

Ex. 6, async pi

5th attempt: let the receiver start

$$\mathcal{A}(\overline{x}y.p) \triangleq x(x_a)\,.\,(\overline{x_a}\langle y\rangle \mid \mathcal{A}(p))$$

$$\mathcal{A}(x(z).q) \triangleq (a)(\overline{x}\langle a\rangle \mid a(z).\mathcal{A}(q))$$

a, x_a fresh

Ex. 6, async pi

5th attempt: let the receiver start

$$\mathcal{A}(\overline{x}y.p) \triangleq x(x_a).(\overline{x_a}\langle y\rangle \mid \mathcal{A}(p))$$

$$\mathcal{A}(x(z).q) \triangleq (a)(\overline{x}\langle a\rangle \mid a(z).\mathcal{A}(q))$$

a, x_a fresh

[**Ex. 7**] The *polyadic* π-calculus allows communicating more than one name in a single action, i.e., its action prefixes are of the form:

$$\pi \ ::= \ \tau \ \mid \ \overline{x}\langle z_1, ...z_n\rangle \ \mid \ x(z_1, ...z_n)$$

The polyadic extension is useful especially when studying types for name passing processes. Show that the polyadic π-calculus can be encoded in the ordinary (monadic) π-calculus by passing the name of a private channel through which the multiple arguments are then passed in a sequence.

Ex. 7, polyadic pi

we want to encode polyadic processes in ordinary ones

$$\mathcal{M}(\mathbf{nil}) \triangleq \mathbf{nil}$$

$$\boxed{\begin{aligned} \mathcal{M}(\overline{x}\langle y_1, ..., y_n\rangle.p) &\triangleq \overline{x}y_1.\cdots.\overline{x}y_n.\mathcal{M}(p) \\ \mathcal{M}(x(z_1, ..., z_n).q) &\triangleq x(z_1).\cdots.x(z_n).\mathcal{M}(q) \end{aligned}}$$

$$\mathcal{M}(\tau.p) \triangleq \tau.\mathcal{M}(p)$$

$$\mathcal{M}([x = y]p) \triangleq [x = y]\mathcal{M}(p)$$

$$\mathcal{M}(p + q) \triangleq \mathcal{M}(p) + \mathcal{M}(q)$$

$$\mathcal{M}(p|q) \triangleq \mathcal{M}(p)|\mathcal{M}(q)$$

$$\mathcal{M}((x)p) \triangleq (x)\mathcal{M}(p)$$

$$\mathcal{M}(!p) \triangleq !\mathcal{M}(p)$$

Ex. 7, polyadic pi

we want to encode polyadic processes in ordinary ones

$$\mathcal{M}(\overline{x}\langle y_1, ..., y_n\rangle.p) \triangleq \overline{x}y_1.\cdots.\overline{x}y_n.\mathcal{M}(p)$$

$$\mathcal{M}(x(z_1, ..., z_n).q) \triangleq x(z_1).\cdots.x(z_n).\mathcal{M}(q)$$

$$\mathcal{M}((x)(\overline{x}\langle a,b\rangle \mid x(y_1,y_2).p \mid x(z_1,z_2).q)) \triangleq (x)(\boxed{\overline{x}a}\,\boxed{\overline{x}b} \mid \boxed{x(y_1)}.x(y_2).\mathcal{M}(p) \mid \boxed{x(z_1)}.x(z_2).\mathcal{M}(q))$$

$$\xrightarrow{\tau} (x)(\mathbf{nil}|p'|x(z_1,z_2).q) \qquad \xrightarrow{\tau} (x)(\mathbf{nil}|x(y_1,y_2).p|q')$$

$$p' = p[^a/_{y_1}, {}^b/_{y_2}] \qquad q' = q[^a/_{z_1}, {}^b/_{z_2}]$$

$$\xrightarrow{\tau} \bullet \xrightarrow{\tau} (x)(\mathbf{nil}|x(y_2).p''|x(z_2).q'')$$

deadlock

$$p'' = \mathcal{M}(p)[^a/_{y_1}] \qquad q'' = \mathcal{M}(q)[^b/_{z_1}]$$

problem: interference between multiple senders / receivers

Ex. 7, polyadic pi

fix a private channel and send all data on it

$$\boxed{\begin{aligned}\mathcal{M}(\overline{x}\langle y_1, ..., y_n\rangle.p) &\triangleq (a)\overline{x}a.\overline{a}y_1.\cdots.\overline{a}y_n.\mathcal{M}(p)\\ \mathcal{M}(x(z_1, ..., z_n).q) &\triangleq x(x_a).x_a(z_1).\cdots.x_a(z_n).\mathcal{M}(q)\end{aligned}}$$

a, x_a fresh