

Scope

Every system needs to have local variables /channels that only its members can interact with. At the same time, the channel can be passed to an external system, thus the scope of the channel changes, it encompasses this external system as well. In π -calculus, this is expressed by the structural rule:

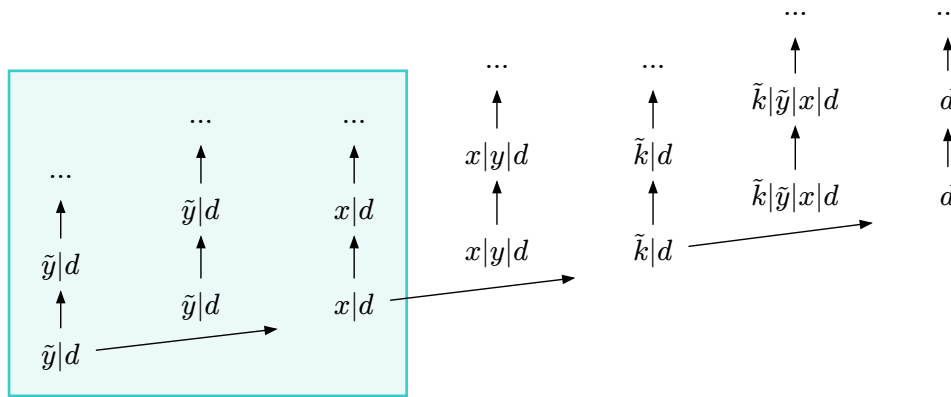
$$(\nu x)(P \mid Q) \equiv (\nu x)P \mid Q \text{ when } x \text{ is not a free variable in } Q.$$

Let us now have an example where we show how we handle scope in our system. Consider these systems:

$$A := (\nu x)(\tilde{y}\langle x \rangle.0 \mid x(k).\tilde{k}\langle q \rangle.0 \mid d(q).A)$$

$$B := y(e).\tilde{e}\langle z \rangle.\tilde{d}\langle k \rangle.0 \mid z(q).B$$

We have discussed that our systems are described by sets of potentialities. Here though, we have the simplest systems. Let us describe system A:



Each horizontal line represents the function of change after system A receives/sends a msg from a specific channel. For the first case, there are two possibilities, either it sent a message on channel \tilde{y} or it received a message on channel d . For this reason, there are two potentialities.

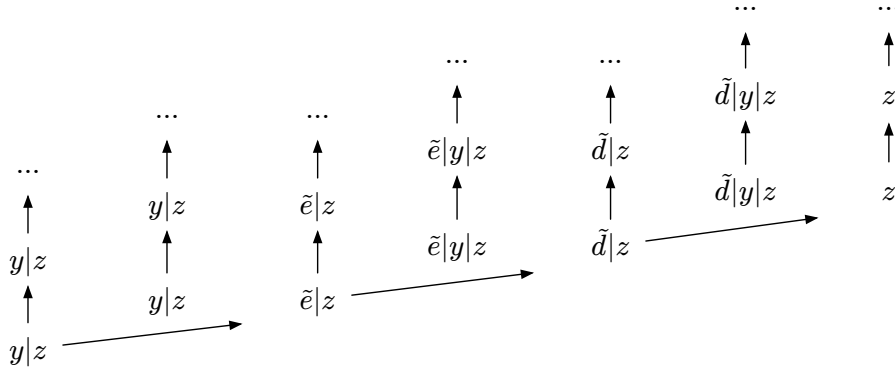
k is a variable, it is the secret to be received by channel x .

Keep in mind that the type does not track the number of **actors** that are present. It only cares whether there is at least one actor that accepts/sends a specific channel. It is idempotent. It is for this reason that the second column is not $\tilde{y}|\tilde{y}|d$

Also, the diagram is incomplete, since we do not describe the functions of change of the other potentialities.

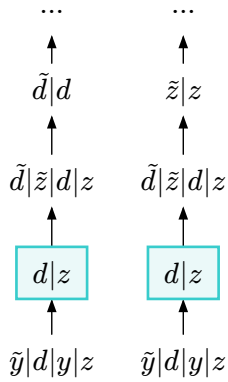
Now, if you look closer, at the first type, it is $\tilde{y}|d$ when channel x is ready to receive new messages. Of course, this is not possible since x is a local variable. In column 3 though, x is part of the type, the reason for that is that channel x has been transmitted to the outside world.

System B:



Both systems A and B are static, meaning that they cannot progress any further without external help. For this reason each column has a constant type. In general though, systems progress on their own.

In the next diagram, I will only describe the initial potentialities of the system A&B, when it is not perturbed by an external system.



The interesting thing happens at the second “state” of the potentialities. Here, it should have been $x|d|\tilde{x}|z$. The reason it is not is because channel x has been sent inside the system, thus we know that no one else has channel x , thus it is impossible that the external world communicate with A&B through channel x .

This is how I handle scope at the moment, by limiting the type of the system to remove channels that cannot be accessed by the exterior world.

In this ‘library’, we do not have channels, by predicates that require the knowledge of a list of secrets. Thus, any predicates that can only be fulfilled from inside the system are removed. This is the functionality of the **limit&** function.

```
restr : ∀{U V} → {A : U → Type} → (P : A → V → Type) → Σ P → A
restr P x = x .pr₁
```

```
_$2_ : ∀{U V} → {A : U → Type} → {B : V → Type} → (A → B) → A × A → B × B
f $2 (a , b) = f a , f b
```

```
+>2 : ∀{U V} → {X : U → Type} → {Y : V → Type} → X + Y → 2
+>2 (inl x) = 0
+>2 (inr x) = 1
```

```
scope-l1 : (x : Secret) → (ls : List Secret) → (A : 2 → W → Type)
           → is-decidable (x ∈ ls) → W → Type
scope-l1 x ls A r = A (+>2 r)
```

```

module BSet-scope ( _E?_ :  $\forall s\ ls \rightarrow \text{is-decidable } (s \in ls)$ ) where

Lim :  $\mathcal{V} \rightarrow 2 \rightarrow \text{Set } \mathcal{V}$ 
Lim P 0 = 0
Lim P 1 = P

limitPr : Secret  $\rightarrow \mathcal{V} \rightarrow \text{Pred } S \times \text{Msg } \mathcal{V}$ 
limitPr s P mp@(ls , msg) = scope-l1 s ls (Lim P) (s E? ls)

limit : Secret  $\rightarrow \text{BSet } \mathcal{V} \rightarrow \text{BSet } \mathcal{V}$ 
limit s bs .pr1 mp = limitPr s (< bs > mp) mp
limit s bs .pr2 =  $\lambda \text{ ascrs } \text{scrs } x \text{ (acs , a>s)} \rightarrow \text{l1 ascrs } \text{scrs } x \text{ acs a>s } (s \text{ E? ascrs})$ 
(s E? scrs) , l2 ascrs scrs x acs a>s (s E? scrs) (s E? ascrs) where
  l1 :  $\forall \text{ ascrs } \text{scrs } x \text{ a>s acs} \rightarrow (\text{deq} : \text{is-decidable } (s \in \text{ascrs})) \rightarrow (\text{deq2} : \text{is-decidable } (s \in \text{scrs})) \rightarrow \text{scope-l1 } s \text{ ascrs } (\text{Lim } (< \text{bs} > (\text{ascrs} , x))) \text{ deq} \rightarrow \text{scope-l1 } s \text{ scrs } (\text{Lim } (< \text{bs} > (\text{scrs} , x))) \text{ deq2}$ 
  l1 ascrs scrs x a>s acs (inr neq) (inl eq2) cond = 0-elim (neq ( $\epsilon \rightarrow \epsilon$  s scrs ascrs acs eq2))
  l1 ascrs scrs x a>s acs (inr neq) (inr x1) cond = bs .pr2 ascrs scrs x (a>s , acs) .pr1 cond

  l2 :  $\forall \text{ ascrs } \text{scrs } x \text{ a>s acs} \rightarrow (\text{deq} : \text{is-decidable } (s \in \text{scrs})) \rightarrow (\text{deq2} : \text{is-decidable } (s \in \text{ascrs})) \rightarrow \text{scope-l1 } s \text{ scrs } (\text{Lim } (< \text{bs} > (\text{scrs} , x))) \text{ deq} \rightarrow \text{scope-l1 } s \text{ ascrs } (\text{Lim } (< \text{bs} > (\text{ascrs} , x))) \text{ deq2}$ 
  l2 ascrs scrs x a>s acs (inr neq) (inl eq2) cond = 0-elim (neq ( $\epsilon \rightarrow \epsilon$  s ascrs scrs a>s eq2))
  l2 ascrs scrs x a>s acs (inr neq) (inr x1) cond = bs .pr2 ascrs scrs x (a>s , acs) .pr2 cond

limitMPr : Secret  $\rightarrow \text{List Secret} \rightarrow \mathcal{V} \rightarrow \text{Pred } S \times \text{Msg } \mathcal{V}$ 
limitMPr s [] bs mp = limitPr s bs mp
limitMPr s (l :: ls) w mp = let w2 = limitPr s w mp
                             w3 = limitMPr l ls w2 mp
                             in w3

limitPr-0 :  $\forall s\ mp \rightarrow \text{limitPr } \{\mathcal{V}\} s\ 0\ mp = 0$ 
limitPr-0 s mp@(scr , _) with (s E? scr)
... | inl x = refl
... | inr x = refl

limitMPr-0 :  $\forall s\ ls\ mp \rightarrow \text{limitMPr } \{\mathcal{V}\} s\ ls\ 0\ mp = 0$ 
limitMPr-0 s [] mp@(scr , _) = limitPr-0 s mp
limitMPr-0 s (l :: ls) mp = ap ( $\lambda z \rightarrow \text{limitMPr } l\ ls\ z\ mp$ ) (limitPr-0 s mp) •
limitMPr-0 l ls mp

limitM : Secret  $\rightarrow \text{List Secret} \rightarrow \text{BSet } \mathcal{V} \rightarrow \text{BSet } \mathcal{V}$ 
limitM s ls bs .pr1 mp = limitMPr s ls (< bs > mp) mp
limitM s [] bs .pr2 = limit s bs .pr2
limitM s (l :: ls) bs .pr2 = limitM l ls (limit s bs) .pr2

limitM' : List Secret  $\rightarrow \text{BSet } \mathcal{V} \rightarrow \text{BSet } \mathcal{V}$ 
limitM' [] bs = bs
limitM' (s :: ls) bs = limitM s ls bs

```

```

-- limitM is a restriction, so it fits where bs fits.
lim-rec : ∀{W} → {A : W → Type} → ∀ s ls {bs mp} → < (limitM {W} s ls bs) > mp → (< bs >
mp → A) → A
lim-rec s [] {bs} {mp@(ws , msg)} c f = ll (s ∈? ws) c where
  ll : (w : (s ∈ ws) + (s ∈ ws → 0)) →
    Lim (< bs > (ws , msg)) (→2 w) → _
  ll (inr _) c = f c

lim-rec {W = V} s (l :: ls) {bs} {mp@(ws , msg)} c f = ll (s ∈? ws) c where
  ll : (w : (s ∈ ws) + (s ∈ ws → 0)) →
    limitMPr l ls (Lim (< bs > (ws , msg)) (→2 w)) (ws , msg) → _
  ll (inl x) c with limitMPr {V} l ls 0 mp | (limitMPr-0 {V} l ls mp)
  ll (inl x) () | r | refl
  ll (inr x) c = lim-rec l ls {bs} {mp} c f

lim-rec' : ∀{W} → {A : W → Type} → ∀ ls bs {mp} → < (limitM' {W} ls bs) > mp → (< bs > mp
→ A) → A
lim-rec' [] _ c f = f c
lim-rec' (x :: ls) bs {mp} = lim-rec x ls {bs}

module &PSet-scope {V} where

  limit&P : Secret → &PSet V W → &PSet V (V ⊔ V + ⊔ W)
  limit&P s ps .pr1 v = v ∈image λ x → (λ (a , bs) → a , limit s bs) (restr < ps > x)
  limit&P s ps .pr2 = cons-is-non-empty

  limit&PM : Secret → List Secret → &PSet V W → &PSet V (V ⊔ V + ⊔ W)
  limit&PM s ls ps .pr1 v = v ∈image λ x → (λ (a , bs) → a , limitM s ls bs) (restr <
ps > x)
  limit&PM s ls ps .pr2 = cons-is-non-empty

```