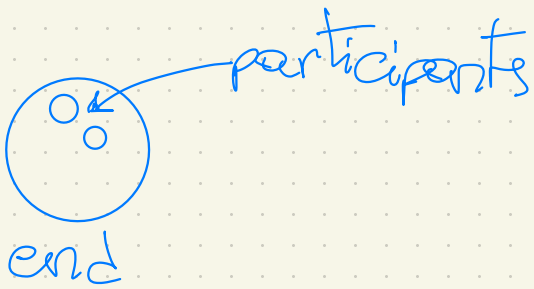
# Encoding Global Types as Petri-Nets

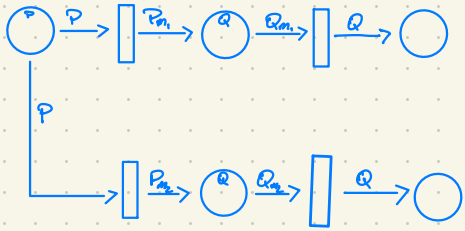
# Petri-Nets as Global Types

1- end

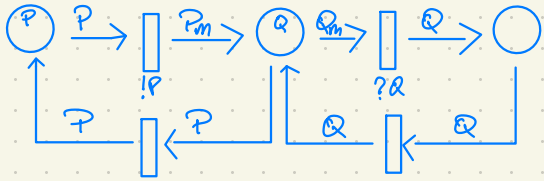



2- $p \to q : m.\ end$

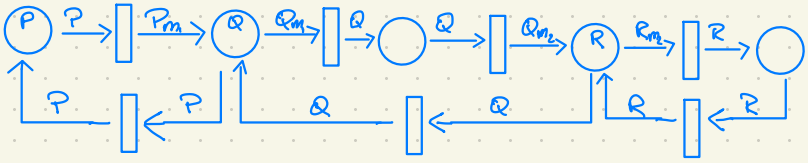

3- $p \to q : \{ m_1.\ end \mid m_2.\ end \}$



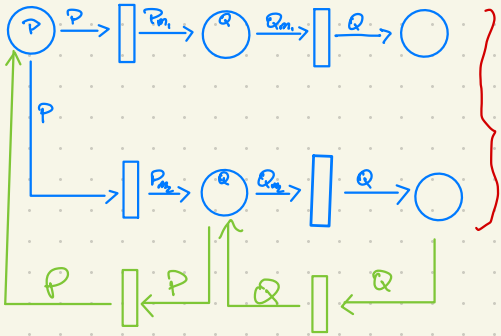

4- $\mu X.\ p \to q : m.\ X$



5_ $\mu X.\ P \to Q : m_1 .\ Q \to R : m_2 .\ X$



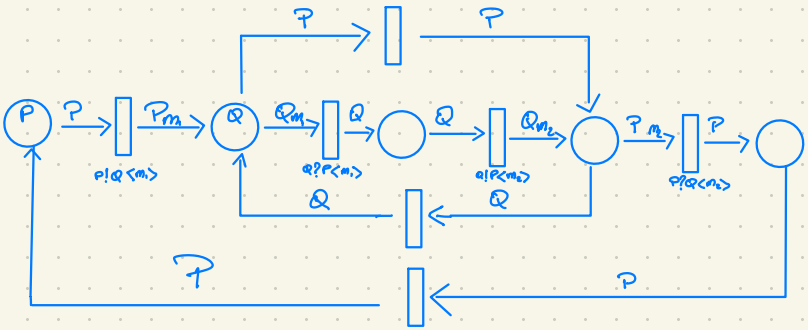

6_ $\mu X.\ P \to Q : \begin{cases} m_1 : \text{end} \\ m_2 . X \end{cases}$



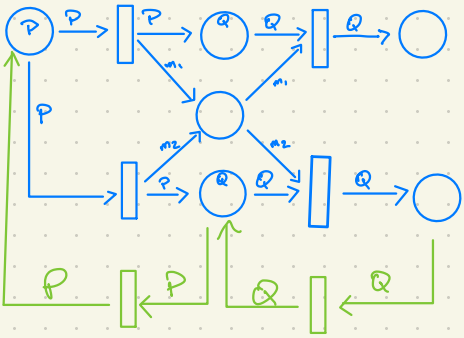

Q: can we use token numbers to overcome this?

Important note: the order of messages is not represented. Notably, if P sends 10 messages and then quits. Q may read 5, then $m_1$, and then 5 more. Q: How do we address this?

7_ $\mu X.\ P \to Q : m_1 .\ Q \to P : m_2 .\ X$ (Ping-pong)



6- $\mu X.\ P \to Q: \begin{cases} m_1 : end \\ m_2 . X \end{cases}$




Note: This net should still be choice free with regards to each sort of token.

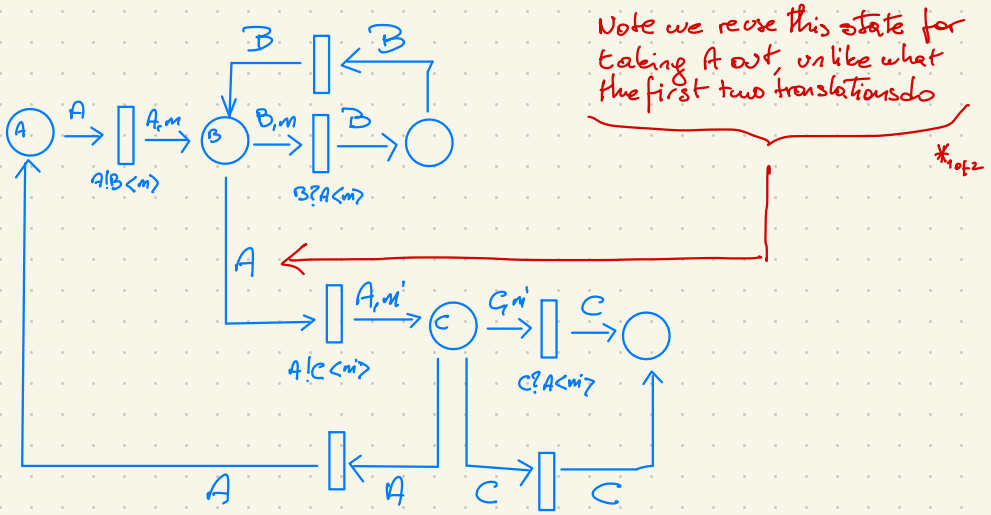
8- $\mu X.\ A \to B: lock_A .$
$B \to A: \begin{cases} acc . A \to B : rel . end \\ lock_B . A \to B : \begin{cases} acc . B \to A : rel . end \\ retry . X \end{cases} \end{cases}$

more complex example, still mergeable but soon to be extended to 3 participants.

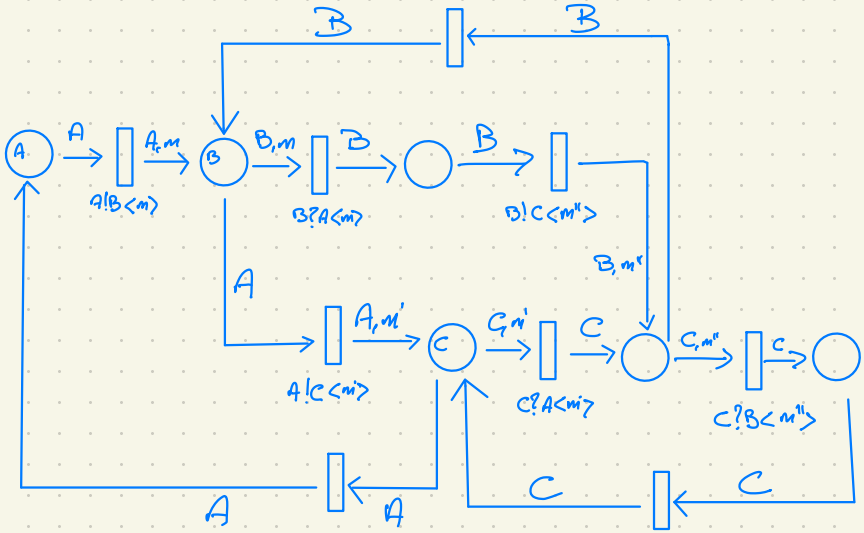
```
module test;

global protocol Lock (role A, role B)
{
  lockA() from A to B;
  choice at B
        {
              acc() from B to A;
              rel() from A to B;
        }
        or
        {
              lockB() from B to A;
              choice at A
              {
                    acc() from A to B;
                    rel() from B to A;
              } or {
                    retry() from A to B ;
                    do Lock (A,B);
              }
  }
}
```

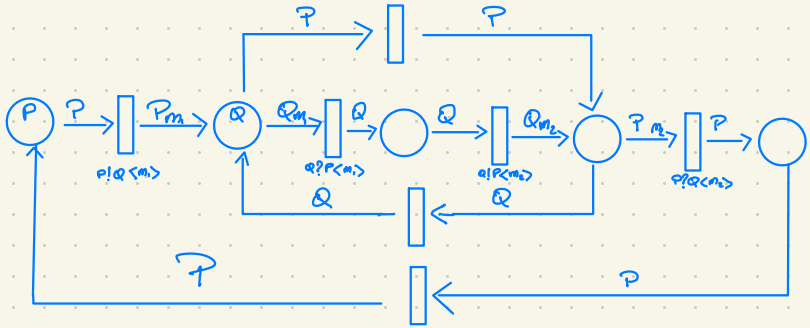
9_ $\mu X.\ A \to B : m.\ A \to C : m'.\ X$

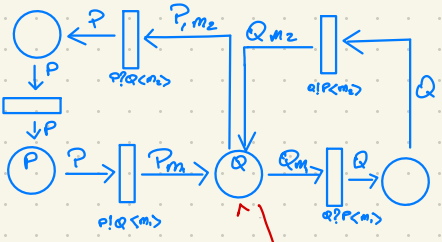



10_ $\mu X.\ A \to B : m.\ A \to C : m'.\ B \to C : m''.\ X$



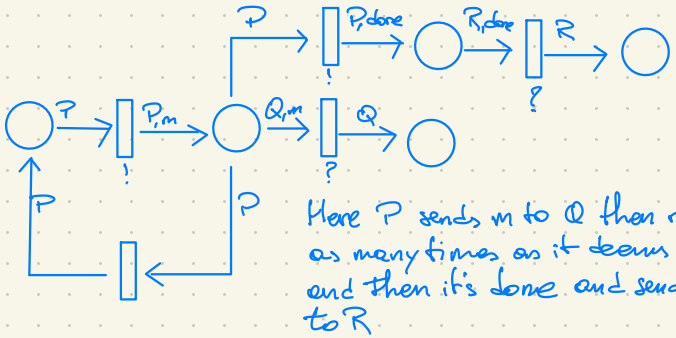

11. $\mu X.\ P \to Q : m_1 .\ Q \to P : m_2 .\ X$ (Two takes on Pingpong) (or whiff-whaff)

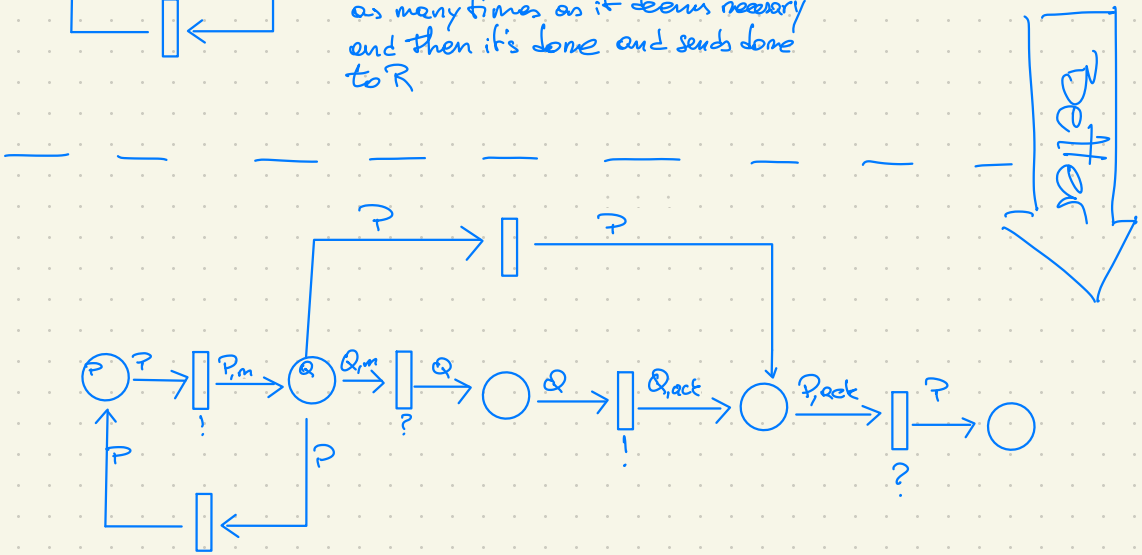



Note: reusing this state may be problematic, because it combines with recursion. In this case it works. However, if Q did something other than receive from P in the continuation, it would be an ill-formed net. We may not want this.

Some nets without global type




In this case P sends to Q the m label, and retries as many times as needed, eventually Q receives m and sends P the label ack enabling P's reception, and transition to its final state.

Note on realizability:

Each participant has stages, at each stage they can do any of the actions enabled for them (actions that are reachable by silent transitions). Once they perform an action they move to the next stage where the available actions are as before.

In the example above for P has three stages: 1- send m, 2- send m again or, if available, receive the ack, 3- finished protocol.

The mixed choice in 2 is enabled by using async IO APIs that have a 'select'-like function to test if a channel has data without locking

# Notes on the algorithm for converting GT to PN

For well formedness we should consider the safety requirements before CFSMs namely: Section 3.3 of 2008's MPST Paper. That provides the criteria to do it. However it will be necessary to relax it for some extensions. The question remains: What are the properties we want/need in global types.

Second, to generate Petri nets check: Multiparty Session Type Meet Comm. Automata In particular section 3.3 and definition 3.4 show how the conversion is done, however I had in mind a more inductive definition using the structure of the global type. This paper is notation heavy and I need to spend more time to get it.

# Multi Party Session Nets

## Preliminaries

### Signature

Given a finite set K of kinds and a set of tokens To a signature $\Sigma$ is a function $To \to K$ that associates each token with a kind.
A well formed signature contains two or more singleton kinds (where a singleton kind is a kind with exactly one inhabitant. And a kind of message labels, inhabited by all the required message labels.

### Transition labels

Given a signature $\Sigma$, the set L of transaction labels is a set of 4-tuples with the following structure:

$$r \; t \; r' \langle m \rangle$$

where r and r' are distinct singleton tokens in $\Sigma$, t is either ! or ? for sending and receiving actions respectively and m is a message label also in $\Sigma$.

## Net Graphs

$N = (P, T, F, f, g)$ where:

P is a finite set of places. A place $p \in P$ contains a set of queues one for each kind of token.

T is a finite set of transitions disjoint from P ($P \cap T = \phi$)

F is a set of arcs $F \subseteq (P \times T) \cup (T \times P)$

f is a function that maps arcs to sets (multisets?) of tokens.

$g : T \to L$ is a partial function that maps transitions to communication labels.

## Net Marking

A net marking is a triple $M = (S, S_0, S_f)$ where:

- $S$, the current marking, is a partial map from places to sets (multiset) of tokens.
- $S_0$, the initial marking, is $S$ at the start.
- $S_f$, ~~the set of final states~~, is a finite set of mappings from places to sets (multisets) of tokens specifying final configurations.

## Multiparty Session Net

$G_\Sigma = (N, M)$ is a multiparty session net where $N$ is a session graph and $M$ is a net marking, and $\Sigma$ provides kinds and tokens.

Notes: still pending are the dynamics of net markings and well formed mess definitions + translation btw global types and networks.

we can avoid the question by saying that only if they are related to a global type.

As meta theory define: $\vdash G_\Sigma$ ok and show that a net related to a type $\supset$ the net is well-formed.

# Translation to Multiparty Session Nets

## Preliminaries

$T : r \to p$ last position seen for a role (if seen)

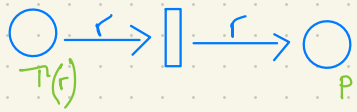
$\Delta : X \to [p]$ remembers all the positions for the roles at the recursion point

TODO: updates of contexts $T|_{r@p}$

of nets: $G_\Sigma$ with $S_j = ?$

## Building the primitives

$T \vdash \text{bring}(r, p) =$ if $T(r)$ exists




empty otherwise

$\text{comm}(p_1, p_2, p_3, m, r_1, r_2)$ yields:


$\Gamma; \Delta \vdash G \leftrightarrow \mathcal{G}_\Sigma$

$\Sigma$ can be trivially obtained from $G$ — first try

Global type $G$ and net $\mathcal{G}_\Sigma$ are related in a context where $\Gamma$ describes the "origin" of roles and $\Delta$ records all the nested recursion we are in.

---

$\Gamma; \Delta \vdash \text{end} \leftrightarrow \bullet$ with $S_f = \{\Gamma_{\text{roles}}\}$

Note: this may be avoidable to prevent spurious silent transitions

$$\frac{p_1, p_2, p_3 \# P \qquad \Gamma \mid r_1 @ p_2 \mid r_2 @ p_3 \vdash \mathcal{G} \leftrightarrow \mathcal{G}}{\Gamma; \Delta \vdash r_1 \to r_2 : m.\ G \leftrightarrow \mathcal{G} \cup \Gamma \vdash \text{bring}(r_1, p_1) \cup \Gamma \vdash \text{bring}(r_2, p_2) \cup \text{comm}(p_1, p_2, p_3, m, r_1, r_2)}$$

with $S_0 = \{r_1 \mapsto p_1,\ r_2 \mapsto p_2\}$

$$\boxed{\begin{array}{c} \Upsilon; \Lambda; \cdot \vdash G \leftrightarrow G_\Sigma \\ \Upsilon; \Lambda; r@p \vdash G \leftrightarrow G_\Sigma \end{array}}$$

$\Sigma$ can be trivially obtained from $G$ — second try

Global type $G$ and net $G_\Sigma$ are related in a context where $\Upsilon$ describes the "origin" of roles and $\Lambda$ records all the nested recursion we are in, and $r$ was left at the output state $p$.

$$\Upsilon; \Lambda; \cdot \vdash \text{end} \leftrightarrow \bullet \text{ with } S_f = \{\Upsilon.\text{roles}\}$$

↑ it does not use the input state

Note: this may be avoidable to prevent spurious silent transitions

$$\frac{p_1, p_2, p_3 \# P \qquad \Upsilon | r_1@p_2 | r_2@p_3 ; r_2@p_3 \vdash G \leftrightarrow G}{\Upsilon; \Lambda; \cdot \vdash r_1 \to r_2 : m.\ G \leftrightarrow G \cup \Upsilon \vdash \text{bring}(r_1, p_1) \cup \Upsilon \vdash \text{bring}(r_2, p_2) \cup \text{comm}(p_1, p_2, p_3, m, r_1, r_2) \text{ with } S_0 = \{r_1 \mapsto p_1,\ r_2 \mapsto p_2\}}$$

$$\frac{r \neq r_1 \qquad \Upsilon; \Lambda; \cdot \vdash r_1 \to r_2 : m.\ G \leftrightarrow G_\Sigma}{\Upsilon; \Lambda; r@p \vdash r_1 \to r_2 : m.\ G \leftrightarrow G_\Sigma}$$

$$\frac{p_1, p_2 \# P \qquad \Upsilon | r_1@p_1 | r_2@p_2 ; \Lambda; r_2@p_2 \vdash G \leftrightarrow G_\Sigma}{\Upsilon; \Lambda; r_1@p \vdash r_1 \to r_2 : m.\ G \leftrightarrow G_\Sigma \cup \Upsilon \vdash \text{bring}(r_2, p_1) \cup \text{comm}(p, p_1, p_2, m, r_1, r_2) \text{ with } S_0 = \{r_1 \mapsto p,\ r_2 \mapsto p_1\}}$$

Note: this algorithm fails for example 9 where there would be one more silent transition.

Fix: instead of remembering the "input" for the next continuation, simply remember all of them and only avoid silent transitions when the transition is leaving the place.

$\Sigma$ can be trivially obtained from $G$

third try

$\boxed{\Gamma ; \Delta \vdash G \leftrightarrow G_\Sigma}$

Global type $G$ and net $G_\Sigma$ are related in a context where $\Gamma$ describes the "origin" of roles and $\Delta$ records all the nested recursion we are in.

---

$\Gamma ; \Delta \vdash end \leftrightarrow \cdot$ with $S_f = \{\Gamma_{roles}\}$

A bit of syntax

$\Gamma(r)! = \Gamma(r)$ if $r \in dom(\Gamma)$

$p \# P$ otherwise

$$\frac{P_1 = \Gamma(r_1)! \quad P_2, P_3 \# P \quad \Gamma \mid r_1 @ P_2 \mid r_2 @ P_3 \vdash G_1 \leftrightarrow G}{\Gamma ; \Delta \vdash r_1 \to r_2 : m . G \leftrightarrow G \cup \Gamma \vdash bring(r_1, P_1) \cup \Gamma \vdash bring(r_2, P_2) \cup comm(P_1, P_2, P_3, m, r_1, r_2)}$$

with $S_0 = \{r_1 \mapsto P_1, r_2 \mapsto P_2\}$