Open Nets, Contexts and Their Properties

Michele Loreti

joint work with Rocco De Nicola

Properties for Global applications...

- Modal logics can be used for specifying and verifying properties of global applications
- However, for this class of applications, one has a limited knowledge of the *involved* components
- We present a new approach for *partial* and *incremental* specification of global applications:
  - not all the components are completely specified
  - a stepwise approach is used to refine the specification

Our proposal...

- A global application can be thought as composed of two parts:
  - a fully known component;
  - its (partially known) operating context.
- We shall rely on:
  - A calculus for modelling distributed and mobile systems
  - A context-specification language for modelling contexts
  - A *location aware* modal logic
  - An *agreement* relation (preserving formulae satisfaction) for refining context components

Our proposal...

- A global application can be thought as composed of two parts:
  - a fully known component;
  - its (partially known) operating context.
- We shall rely
  - A calculus ed and mobile systems
  - A context- for modelling contexts
  - A *location aware* modal logic
  - An *agreement* relation (preserving formulae satisfaction) for refining context components

μKLAIM

μKLAIM Nodes...

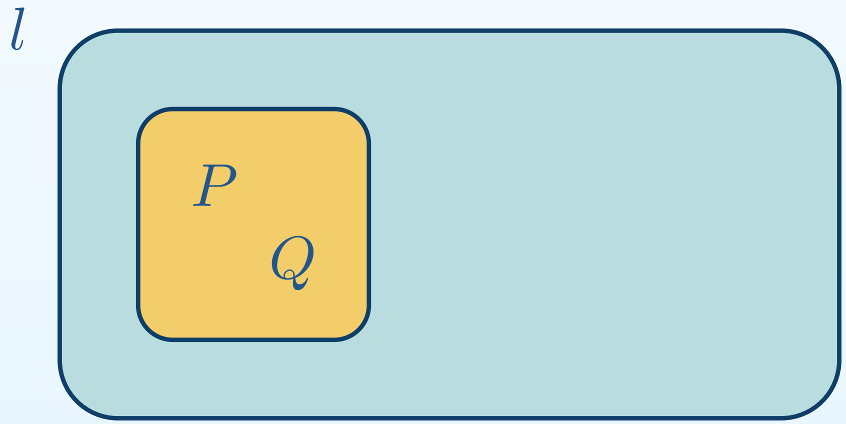
μKLAIM Nodes...

μKLAIM Nodes...

- Locality

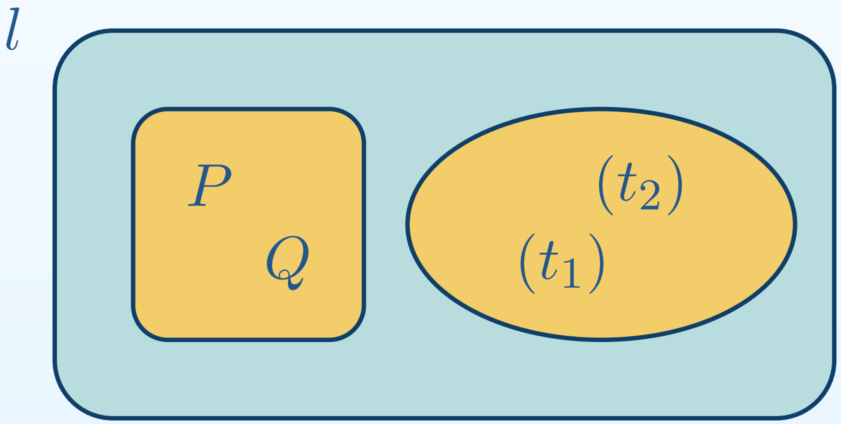
μKLAIM Nodes...

- Locality
- Processes



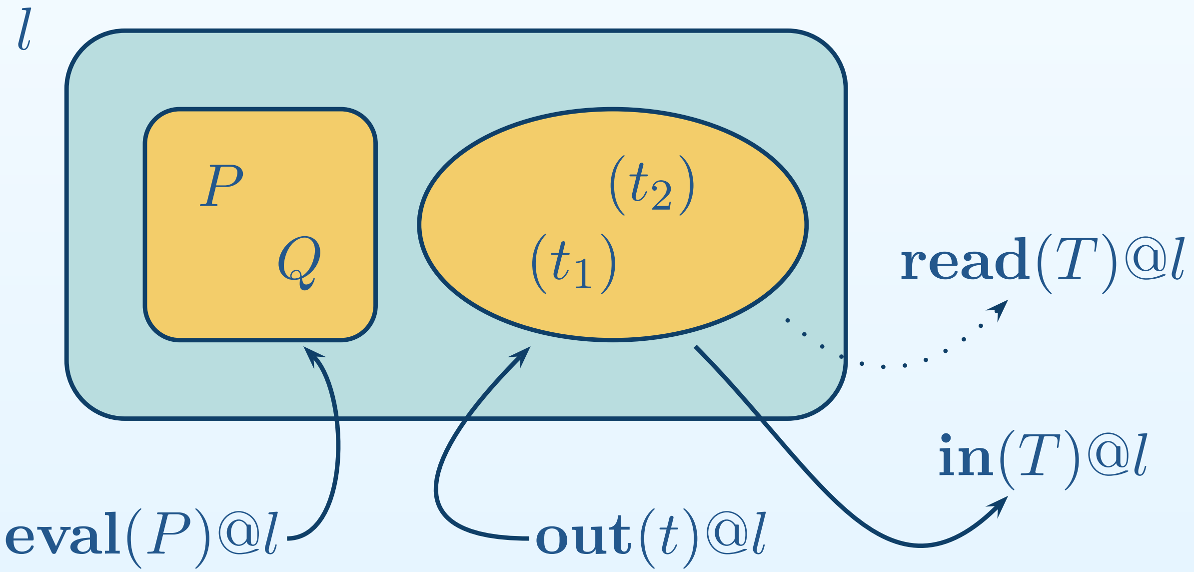
μKLAIM Nodes...

- Locality
- Processes
- Tuple Space



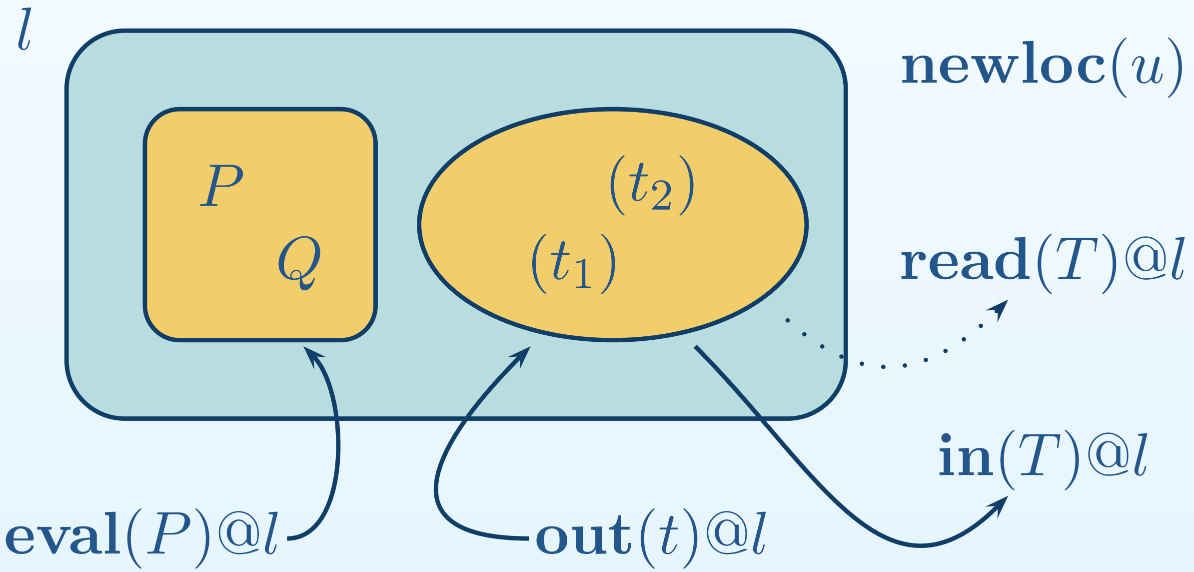
μKLAIM Nodes...

- Locality
- Processes
- Tuple Space



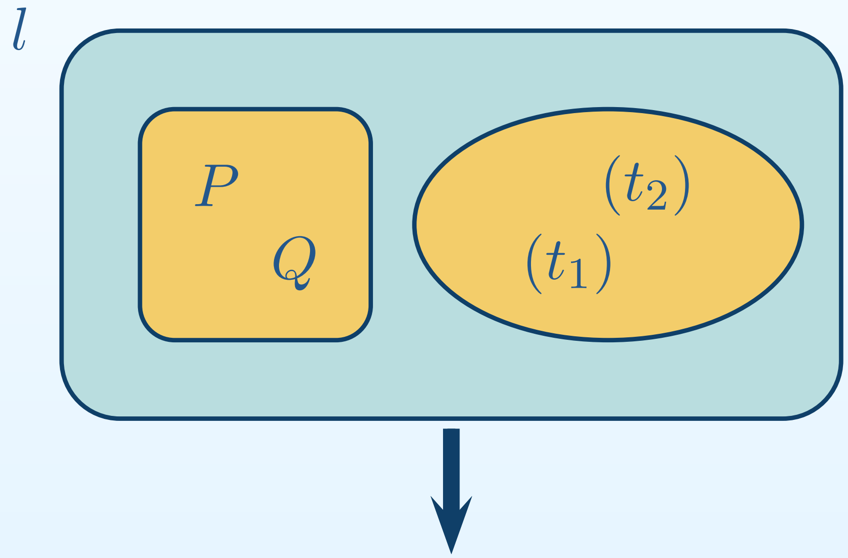
μKLAIM Nodes...

- Locality
- Processes
- Tuple Space



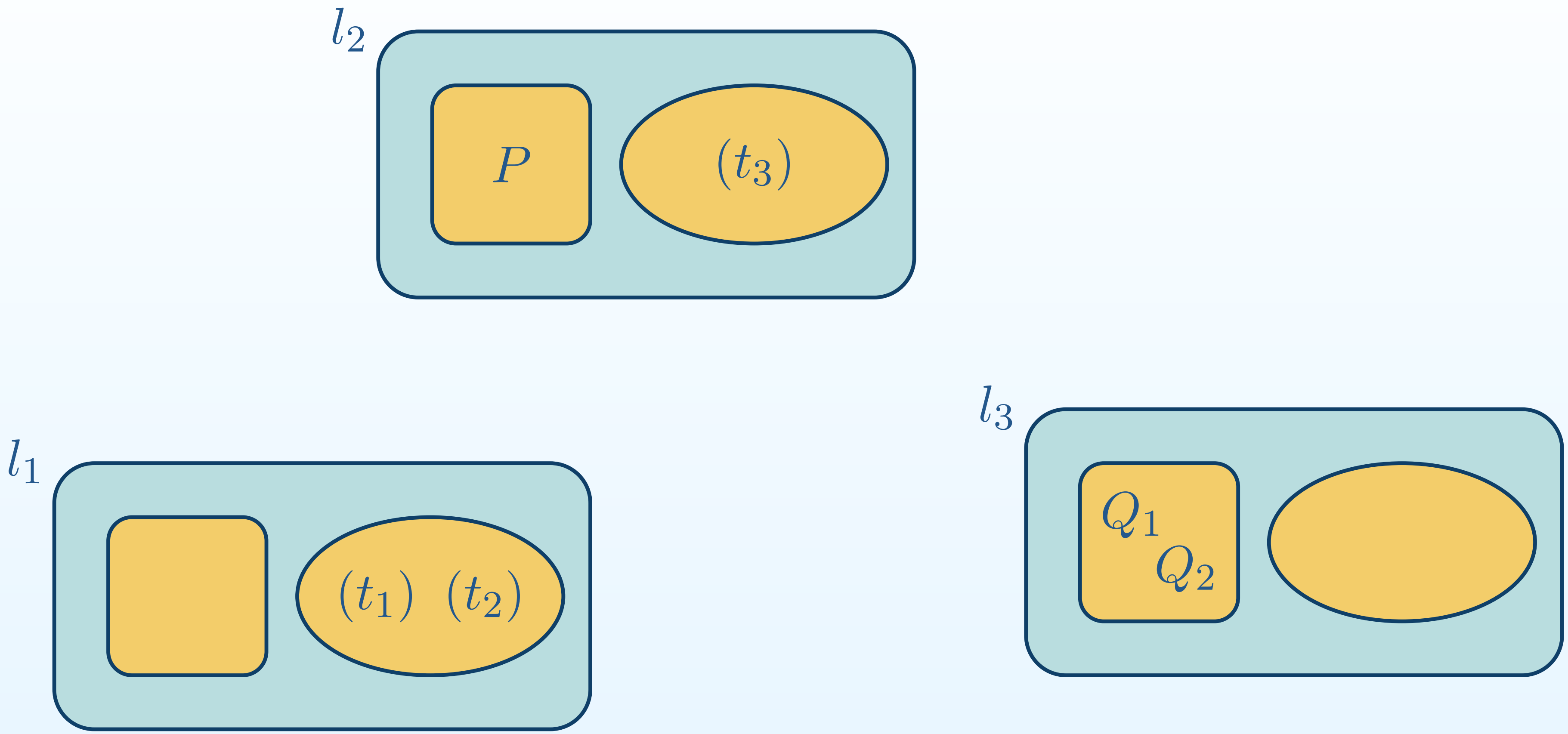
μKLAIM Nodes...

- Locality
- Processes
- Tuple Space

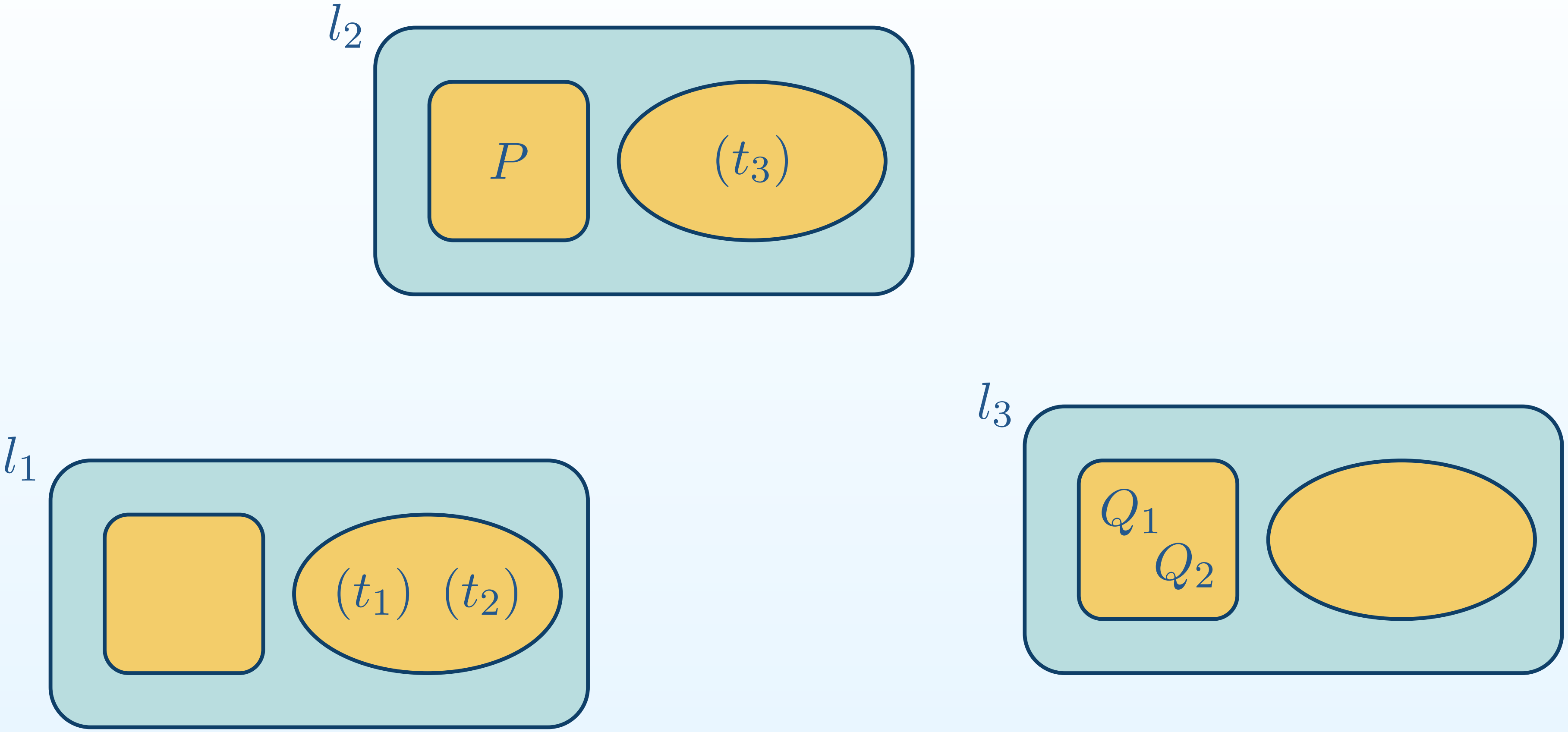


$$l :: P \parallel l :: Q \parallel l :: (t_1) \parallel l :: (t_2)$$

μKLAIM Nets...



μKLAIM Nets...



$$l_1 :: (t_1) \parallel l_1 :: (t_2) \parallel$$
$$l_2 :: P \parallel l_2 :: (t_3) \parallel l_3 :: Q_1 \parallel l_3 :: Q_2$$

μKlaim syntax...

$$
\begin{array}{lcl}
N & ::= & l :: R \;\Big|\; N_1 \parallel N_2 \\
R & ::= & P \;\Big|\; (et) \\
P & ::= & \mathbf{nil} \;\Big|\; act.P \;\Big|\; P_1 \mid P_2 \;\Big|\; X \;\Big|\; \mathbf{rec}X.P \\
act & ::= & \mathbf{out}(t)@l \;\Big|\; \mathbf{in}(T)@l \;\Big|\; \mathbf{read}(T)@l \;\Big| \\
 & & \mathbf{eval}(P)@l \;\Big|\; \mathbf{newloc}(u) \\
t & ::= & f \;\Big|\; f, t \\
f & ::= & e \;\Big|\; l \;\Big|\; u \\
T & ::= & F \;\Big|\; F, T \\
F & ::= & f \;\Big|\; !\,x \;\Big|\; !\,u
\end{array}
$$

Labelled Operational Semantics...

- Λ denotes the set of transizion label λ defined as follows:

$$\lambda ::= l : act \,\Big|\, \tau$$

- the operational semantics of μKLAIM nets is defined using relation $\stackrel{\cdot}{\rightarrowtail}$

Labelled Operational Semantics...

- Λ denotes the set of transizion label λ defined as follows:

$$\lambda ::= l : act \;\Big|\; \tau$$

- the operational semantics of μKLAIM nets is defined using relation $\succ\!\xrightarrow{\cdot}$
- some rules:

$$l_1 :: \mathbf{out}(t)@l_2.P \succ\!\xrightarrow{l_1:\mathbf{out}(\mathcal{T}[\![\, t \,]\!])@l_2} l_1 :: P$$

$$\frac{N_1 \succ\!\xrightarrow{l_1:\mathbf{out}(et)@l_2} N_2}{N_1 \parallel l_2 :: P \succ\!\xrightarrow{\tau} N_2 \parallel l_2 :: P \parallel l_2 :: (et)}$$

Labelled Operational Semantics...

- Λ denotes the set of transizion label λ defined as follows:

$$\lambda ::= l : act \;\Big|\; \tau$$

- the operational semantics of μKLAIM nets is defined using relation $\succ\!\xrightarrow{\cdot}$
- some rules:

$$l_1 :: \mathbf{in}(T)@l_2.P \succ\!\xrightarrow{l_1 : \mathbf{in}(T)@l_2} l :: P$$

$$\frac{N_1 \succ\!\xrightarrow{l_1 : \mathbf{in}(T)@l_2} N_2 \;\; \sigma = match(T, et)}{N_1 \parallel l_2 :: (et) \succ\!\xrightarrow{\tau} N_2\sigma \parallel l_2 :: \mathbf{nil}}$$

Dining philosophers in μKLAIM...

- There is a node for each fork (f_i)
  - the i–th fork is *free* if $(''fork'')$ is at f_i
- There is a node for each philosopher (p_i)
  - the process below is at p_i:

$$
\begin{aligned}
P_i = \mathbf{rec}\ X.\ & \\
& \text{\# think...} \\
& \mathbf{in}(''fork'')@f_i. \\
& \mathbf{in}(''fork'')@f_{(i+1)\mathbf{mod}\ n}. \\
& \text{\# eat...} \\
& \mathbf{out}(''fork'')@f_i. \\
& \mathbf{out}(''fork'')@f_{(i+1)\mathbf{mod}\ n}. \\
& X
\end{aligned}
$$

Dining philosophers in μKLAIM (1)...

$$N_{DP} = f_0 :: (''fork'') \parallel p_0 :: P_0 \parallel$$
$$f_1 :: (''fork'') \parallel p_1 :: P_1 \parallel$$
$$f_2 :: (''fork'') \parallel p_2 :: P_2 \parallel$$
$$f_3 :: (''fork'') \parallel p_3 :: P_3 \parallel$$
$$f_4 :: (''fork'') \parallel p_4 :: P_4 \parallel$$
$$f_5 :: (''fork'') \parallel p_5 :: P_5$$

Properties of dining philosophers...

- Deadlock freedom
- Philosopher at p_i accesses only f_i and $f_{(i+1) \mathbf{mod}\ n}$
- The $i-$th philosopher can hope to eat
- The $i-$th philosopher cannot starve

A Modal logic for μKLAIM...

- The proposed logic is a variant of HML where:
  - the modal operator $\langle\cdot\rangle$ is indexed with a *label predicate*
  - state formulae are introduced for specifying the distribution of resources (i.e. data stored in nodes) in the system

Some definitions...

- N_1 and N_2 are *data equivalent* ($N_1 \asymp N_2$) if and only if they have the *same* tuple spaces
- $\stackrel{\varepsilon}{\Longrightarrow}$ denotes the reflexive and transitive closure of $\stackrel{\tau}{\succ\!\!\longrightarrow}$
- $N_1 \stackrel{l:act}{\Longrightarrow} N_2$ if and only if there exist N_1' and N_2' such that:

$$N_1 \stackrel{\varepsilon}{\Longrightarrow} N_1' \stackrel{l:act}{\succ\!\!\longrightarrow} N_2' \stackrel{\varepsilon}{\Longrightarrow} N_2$$

- $N_1 \stackrel{\tau}{\Longrightarrow} N_2$ if and only if there exist N_1' and N_2' such that $N_1' \not\asymp N_2'$ and:

$$N_1 \stackrel{\varepsilon}{\Longrightarrow} N_1' \stackrel{\tau}{\succ\!\!\longrightarrow} N_2' \stackrel{\varepsilon}{\Longrightarrow} N_2$$

Syntax of Formulae...

$$\phi ::= \mathbf{true} \;\Big|\; \phi \vee \phi \;\Big|\; \neg\phi \;\Big|\; (T)@\ell \Rightarrow \phi \;\Big|\; (et)@\ell \Leftarrow \phi \;\Big|\; n(u).\phi \;\Big|\; \langle\mathcal{A}\rangle\phi \;\Big|\; \kappa \;\Big|\; \nu\kappa.\phi$$

$$\mathcal{A} ::= \tau \;\Big|\; \ell_1 : \mathtt{O}(et)@\ell_2 \;\Big|\; \ell_1 : \mathtt{I}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{R}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{E}(\phi)@\ell_2 \;\Big|\; \ell_1 : \mathtt{N}(u)$$

Syntax of Formulae...

$$\phi ::= \mathbf{true} \;\Big|\; \phi \vee \phi \;\Big|\; \neg\phi \;\Big|\; (T)@\ell \Rightarrow \phi \;\Big|\; (et)@\ell \Leftarrow \phi \;\Big|\; n(u).\phi \;\Big|\; \langle\mathcal{A}\rangle\phi \;\Big|\; \kappa \;\Big|\; \nu\kappa.\phi$$

$$\mathcal{A} ::= \tau \;\Big|\; \ell_1 : \mathtt{O}(et)@\ell_2 \;\Big|\; \ell_1 : \mathtt{I}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{R}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{E}(\phi)@\ell_2 \;\Big|\; \ell_1 : \mathtt{N}(u)$$

- $N_1 \models (T)@l \Rightarrow \phi$ if and only if there exists N_2 such that:
  - $N_1 \equiv N_2 \parallel l :: (et)$
  - $\sigma = match(T, et)$
  - $N_2\sigma \models \phi\sigma$

Syntax of Formulae...

$$\phi ::= \mathbf{true} \;\Big|\; \phi \vee \phi \;\Big|\; \neg\phi \;\Big|\; (T)@\ell \Rightarrow \phi \;\Big|\; (et)@\ell \Leftarrow \phi \;\Big|\; n(u).\phi \;\Big|\; \langle \mathcal{A} \rangle \phi \;\Big|\; \kappa \;\Big|\; \nu\kappa.\phi$$

$$\mathcal{A} ::= \tau \;\Big|\; \ell_1 : \mathtt{O}(et)@\ell_2 \;\Big|\; \ell_1 : \mathtt{I}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{R}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{E}(\phi)@\ell_2 \;\Big|\; \ell_1 : \mathtt{N}(u)$$

- $N_1 \models (et)@l \Leftarrow \phi$ if and only if $N_1 \parallel l :: (et) \models \phi$

Syntax of Formulae...

$$\phi ::= \mathbf{true} \;\Big|\; \phi \vee \phi \;\Big|\; \neg\phi \;\Big|\; (T)@\ell \Rightarrow \phi \;\Big|\; (et)@\ell \Leftarrow \phi \;\Big|\; n(u).\phi \;\Big|\; \langle\mathcal{A}\rangle\phi \;\Big|\; \kappa \;\Big|\; \nu\kappa.\phi$$

$$\mathcal{A} ::= \tau \;\Big|\; \ell_1 : \mathtt{O}(et)@\ell_2 \;\Big|\; \ell_1 : \mathtt{I}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{R}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{E}(\phi)@\ell_2 \;\Big|\; \ell_1 : \mathtt{N}(u)$$

- $N_1 \models n(u).\phi$ if and only if there exists $l \notin N_1$ such that $N_1[l/u] \parallel l :: \mathbf{nil} \models \phi[l/u]$

Syntax of Formulae...

$$\phi ::= \mathbf{true} \;\Big|\; \phi \vee \phi \;\Big|\; \neg\phi \;\Big|\; (T)@\ell \Rightarrow \phi \;\Big|\; (et)@\ell \Leftarrow \phi \;\Big|\; n(u).\phi \;\Big|\; \langle\mathcal{A}\rangle\phi \;\Big|\; \kappa \;\Big|\; \nu\kappa.\phi$$

$$\mathcal{A} ::= \tau \;\Big|\; \ell_1 : \mathtt{O}(et)@\ell_2 \;\Big|\; \ell_1 : \mathtt{I}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{R}(T)@\ell_2 \;\Big|\; \ell_1 : \mathtt{E}(\phi)@\ell_2 \;\Big|\; \ell_1 : \mathtt{N}(u)$$

- $N_1 \models \langle\mathcal{A}\rangle\phi$ if and only if $N_1 \xrightarrow{\lambda} N_2$, $\lambda \models \mathcal{A}$ and $N_2 \models \phi$
- where:
  - $\tau \models \tau$
  - $l_1 : \mathbf{out}(et)@l_2 \models l_1 : \mathtt{O}(et)@l_2$
  - $l_1 : \mathbf{in}(T)@l_2 \models l_1 : \mathtt{I}(T)@l_2$
  - $l_1 : \mathbf{read}(T)@l_2 \models l_1 : \mathtt{R}(T)@l_2$
  - $l_1 : \mathbf{eval}(P)@l_2 \models l_1 : \mathtt{E}(\phi)@l_2$ if and only $l_2 :: P \models \phi$
  - $l_1 : \mathbf{newloc}(u) \models l_1 : \mathtt{N}(u)$

Derivable operators...

Formulae:

- $[\mathcal{A}]\phi$
- $\phi_1 \wedge \phi_2$
- $\mu\kappa.\phi$

Label predicates:

- $\texttt{Src}(l_1)$, to denote actions performed at l_1 $\quad (l_1 : act \models \texttt{Src}(l))$
- $\texttt{Trg}(l_2)$, to denote actions that take effect at l_2 $\quad (l_2 : a@l_2 \models \texttt{Trg}(l_2))$
- $\mathcal{A}_1 \cup \mathcal{A}_2$, disjunction $\quad (\lambda \models \mathcal{A}_1 \cup \mathcal{A}_2 \Leftrightarrow \lambda \models \mathcal{A}_1$ or $\lambda \models \mathcal{A}_2)$
- $\mathcal{A}_1 \cap \mathcal{A}_2$, conjunction $\quad (\lambda \models \mathcal{A}_1 \cap \mathcal{A}_2 \Leftrightarrow \lambda \models \mathcal{A}_1$ and $\lambda \models \mathcal{A}_2)$
- $\mathcal{A}_1 - \mathcal{A}_2$, difference $\quad (\lambda \models \mathcal{A}_1 - \mathcal{A}_2 \Leftrightarrow \lambda \models \mathcal{A}_1$ and $\lambda \not\models \mathcal{A}_2)$

Properties of dining philosophers...

- Deadlock freedom

$$\nu\kappa.\langle\tau\rangle\mathbf{true} \vee [\tau]\kappa$$

- Philosopher at p_i accesses only f_i and $f_{(i+1)\mathbf{mod}\ n}$

$$\neg(\mu\kappa.\langle\mathtt{Src}(p_i) - (\mathtt{Trg}(f_i) \cup \mathtt{Trg}(f_{(i+1)\mathbf{mod\ n}}))\rangle\mathbf{true}\vee\langle\tau\rangle\kappa)$$

- The $i-$th philosopher can hope to eat

$$pick_i = \langle p_i : \mathbf{in}(''fork'')@f_{(i+1)\mathbf{mod}\ n}\rangle(''fork'') \Rightarrow \mathbf{true}$$

$$\mu\kappa.pick_i \vee \langle\tau\rangle\kappa$$

- The $i-$th philosopher cannot starve

$$\mu\kappa.take_i \vee [\tau]\kappa$$

Properties of dining philosophers...

- Deadlock freedom

$$N_{DF} \not\models \nu\kappa.\langle\tau\rangle\mathbf{true} \vee [\tau]\kappa$$

- Philosopher at p_i accesses only f_i and $f_{(i+1)\mathbf{mod}\ n}$

$$N_{DF} \models \neg(\mu\kappa.\langle\mathtt{Src}(p_i) - (\mathtt{Trg}(f_i) \cup \mathtt{Trg}(f_{(i+1)\mathbf{mod\ n}}))\rangle\mathbf{true} \vee \langle\tau\rangle\kappa)$$

- The $i-$th philosopher can hope to eat

$$pick_i = \langle p_i : \mathbf{in}(''fork'')@f_{(i+1)\mathbf{mod}\ n}\rangle(''fork'') \Rightarrow \mathbf{true}$$

$$N_{DF} \models \mu\kappa.pick_i \vee \langle\tau\rangle\kappa$$

- The $i-$th philosopher cannot starve

$$N_{DF} \not\models \mu\kappa.take_i \vee [\tau]\kappa$$

Different kinds of systems...

- Closed systems:
  - Complete knowledge of all system components
- Open systems:
  - Partial knowledge of systems, some components are fully known, others are unknown or partially known.
- Context dependent systems:
  - Abstract context specification plus concrete specification of some components

Dealing with context dependent nets...

- Describe known components of a system with μKLAIM.
- Partially specify contexts (the rest of the systems) with an ad-hoc formalism.
- Specify system properties with μKLAIM logics and related tools.
- Guarantee properties preservation while instantiating (part of) the context.

Abstract nets...

- New syntax for nets: $AN ::= N \Bigm| c \Bigm| AN_1 \parallel AN_2$
- Context specification:
  - Deadlocked context: $\mathbf{0}$
  - Resources:
    - localities ($@l$)
    - data ($(et)@l_2$)
  - Computations: $(l_1 : act).c$
  - Composition: $c_1 \otimes c_2$
  - Choice: $c_1 \oplus c_2$
  - Recursion: $\mathbf{rec}\pi.c$

Context operational semantics...

- Operational semantics of μKLAIM is extended in order to consider contexts
- Some rules:

$$(l_1 : \mathbf{out}(t)@l_2).c \xrightarrow{l_1 : \mathbf{out}(\mathcal{T}[\![\, t \,]\!])@l_2} c$$

$$\frac{AN_1 \xrightarrow{l_1 : \mathbf{out}(et)@l_2} AN_2}{AN_1 \parallel @l_2 \xrightarrow{\tau} AN_2 \parallel @l_2 \parallel l_2 :: (et)}$$

Dining philosophers in μKLAIM: open specification...

$$
\begin{aligned}
N_{DP} = f_0 :: (''fork'') \parallel p_0 :: P_0 \parallel \\
f_1 :: (''fork'') \parallel p_1 :: P_1 \parallel \\
f_2 :: (''fork'') \parallel p_2 :: P_2 \parallel \\
f_3 :: (''fork'') \parallel p_3 :: P_3 \parallel \\
f_4 :: (''fork'') \parallel p_4 :: P_4 \parallel \\
f_5 :: (''fork'') \parallel p_5 :: P_5
\end{aligned}
$$

Dining philosophers in μKLAIM: open specification...

$$AN_{DP} = f_1 :: (''fork'') \parallel p_1 :: P_1 \parallel f_2 :: (''fork'') \parallel c_{dp}$$

where:

$$\begin{aligned}
c_{dp} \stackrel{\text{def}}{=} \mathbf{rec}\ \pi_1. \quad & (p_0 : \mathbf{in}(''fork'')@f_1).(p_0 : \mathbf{out}(''fork'')@f_1).\pi_1 \\
& \oplus \\
& (p_2 : \mathbf{in}(''fork'')@f_2).(p_2 : \mathbf{out}(''fork'')@f_2).\pi_1) \\
& \oplus \\
& (p_0 : \mathbf{in}(''fork'')@f_1).(p_2 : \mathbf{in}(''fork'')@f_2). \\
& \quad (p_0 : \mathbf{out}(''fork'')@f_1).(p_2 : \mathbf{out}(''fork'')@f_2).\pi_1) \\
& \oplus \\
& (p_2 : \mathbf{in}(''fork'')@f_2).(p_0 : \mathbf{in}(''fork'')@f_1). \\
& \quad (p_2 : \mathbf{out}(''fork'')@f_2).(p_0 : \mathbf{out}(''fork'')@f_1).\pi_1
\end{aligned}$$

Context concretion...

- Let L be a set of localities, we have defined two relations:
  - $\sqsubseteq_L$, inspired by *branching simulation*
  - $\simeq_L$, inspired by *branching bisimulation*
- A net N *approximates* the context c over the abstract net $c_1 \parallel N_1$ if and only if:

$$c \sqsubseteq_{loc(c_1 \| N_1)} N$$

- A net N *agrees* with the context c over the abstract net $c_1 \parallel N_1$ if and only if:

$$N \simeq_{loc(c_1 \| N_1)} c$$

Context concretion...

- If N approximates c over $c_1 \parallel N_1$, for each ϕ *positive* localized at L:

$$\text{if } c_1 \wedge c \parallel N_1 \models \neg\phi \text{ then } c_1 \parallel N \parallel N_1 \models \neg\phi$$

- If N agrees with c over $c_1 \parallel N_1$, for each ϕ localized at L:

$$c_1 \wedge c \parallel N_1 \models \phi \text{ if and only if } c_1 \parallel N \parallel N_1 \models \phi$$

Conclusions...

- Klaim can be used to model concurrent mobile spatially distributed systems
- Klaim logics can be used to verify properties of Klaim nets
- Contexts and the logics are tools for modelling open nets and establishing their properties guaranteeing properties preservation during progressive implementation of the context.

Future works:

- Evaluating the expressive (descriptive) power of our logic;
- Contrast it with other logics (pi-logics, spatial logics,...)
- Apply the open framework to other formalisms (Dpi, variants of Ambient,...)

The End