Model Checking Mobile Stochastic Logics

Rocco De Nicola\textsuperscript{1}  Joost-Pieter Katoen\textsuperscript{2}
Diego Latella\textsuperscript{3}
Michele Loreti\textsuperscript{1}  Mieke Massink\textsuperscript{3}

\textsuperscript{1}DSIUF- Università di Firenze
\textsuperscript{2}RHWT - University of Aachen
\textsuperscript{3}ISTI-Istituto di Scienza e Tecnologie della Informazione “A. Faedo”

Lucca, June 25, 2009
Outline

1. A brief introduction to StoKlaim

2. MoSL: Mobile Stochastic Logic

3. Model Checking MoSL

4. Concluding Remarks
Kernel Language for Agent Interaction and Mobility

Process Calculus Flavored
- Small set of basic combinator;
- Clean operational semantics.

Linda based communication model
- Asynchronous communication;
- Shared tuple spaces;
- Pattern Matching

Explicit Distribution
- Multiple distributed tuple spaces;
- Code and Process mobility.
Explicit Localities to model distribution

- Physical Locality (sites)
- Logical Locality (names for sites)
- A distinct name `self` (or `here`) indicates the site a process is on.

Allocation environment to associate sites to logical localities

- This avoids the programmers to know the exact physical structure.

Process Algebras Operators to compose programs

- Sequentialization
- Parallel composition
- Creation of new names
**Klaim Nodes and Klaim Nets**

**Klaim Nodes**

 consist of:

- a site
- a tuple space
- a set of parallel processes
- an allocation environment

**Klaim Nets**

 are:

- a set of Klaim nodes linked via the allocation environment
StoKlaim: Stochastically Timed Actions

- Actions execution take time
StoKlaim: Stochastically Timed Actions

- Actions execution take time
- Execution times is described by means of Random Variables
**StoKlaim: Stochastically Timed Actions**

- Actions execution **take time**
- Execution times is described by means of **Random Variables**
- Random Variables are assumed to be **Exponentially Distributed**
StoKlaim: *Stochastically Timed Actions*

- Actions execution take time
- Execution times is described by means of Random Variables
- Random Variables are assumed to be Exponentially Distributed
- Random Variables are fully characterized by their Rates
StoKlaim: Stochastically Timed Actions

- Actions execution take time
- Execution times is described by means of Random Variables
- Random Variables are assumed to be Exponentially Distributed
- Random Variables are fully characterized by their Rates

From Klaim to StoKlaim
**StoKlaim: Stochastically Timed Actions**

- Actions execution take time
- Execution times is described by means of Random Variables
- Random Variables are assumed to be Exponentially Distributed
- Random Variables are fully characterized by their Rates

**From Klaim to StoKlaim**

- **Klaim Action Prefix:** $A.P$
**StoKlaim: Stochastically Timed Actions**

- Actions execution take time
- Execution times is described by means of Random Variables
- Random Variables are assumed to be Exponentially Distributed
- Random Variables are fully characterized by their Rates

---

**From Klaim to StoKlaim**

- **Klaim** Action Prefix: $A.P$
- **StoKlaim** Action Prefix: $(A, r).P$
**StoKlaim Actions**

- *(out(\(T\))@I2, r1)*
  - *uploads* tuple \(T\) to \(I2\),
  - *the time it takes* is e.d. with rate \(r1\)

- *(eval(P))@I1, r2)*
  - *spawns* process \(P\) to \(I1\),
  - *the time it takes* is e.d. with rate \(r2\)

- *(newloc(!u), r3)*
  - *creates* a new site (with locality) \(u\),
  - *the time it takes* is e.d. with rate \(r3\)

- *(in(F))@I1, r4)*
  - *downloads*, if available, a tuple matching \(F\) from \(I1\),
  - *it takes a time* which is e.d. with rate \(r4\),

- *(read(F))@I1, r4)*
  - *reads*, if available, a tuple matching \(F\) from \(I1\), *without consuming it*
  - *it takes a time* which is e.d. with rate \(r4\),
StoKlaim Syntax

Nets: \( N ::= 0 \mid i ::_\rho E \mid N \parallel N \)

Node Elements: \( E ::= P \mid \langle \vec{f} \rangle \)

Processes: \( P ::= \text{nil} \mid (A, r).P \mid P + P \mid P \mid P \mid X(\vec{P}, \vec{\ell}, \vec{e}) \)

Actions: \( A ::= \text{out}(\vec{f})@\ell \mid \text{in}(\vec{F})@\ell \mid \text{read}(\vec{F})@\ell \mid \text{eval}(P)@\ell \mid \text{newloc}(!u) \)

Tuple Fields: \( f ::= P \mid \ell \mid e \)

Template Fields: \( F ::= f \mid !X \mid !u \mid !x \)
Operational Semantics for \textit{StoKlaim}

Stochastic semantics of \textit{StoKlaim} is defined by means of a transition relation $\rightarrow$ that associates to a process $P$ and a transition label $\alpha$ a function $(\mathcal{P}, \mathcal{Q}, \ldots)$ that maps each process into a non-negative real number.
Operational Semantics for StoKlaim

Stochastic semantics of StoKlaim is defined by means of a transition relation \( \rightarrow \) that associates to a process \( P \) and a transition label \( \alpha \) a function \((P, Q, \ldots)\) that maps each process into a non-negative real number.

\[ P \xrightarrow{\alpha} \mathcal{P} \] means that:

- if \( \mathcal{P}(Q) = x \neq 0 \) then \( Q \) is reachable from \( P \) via the execution of \( \alpha \) with rate or weight \( x \)
- if \( \mathcal{P}(Q) = 0 \) then \( Q \) is not reachable from \( P \) via \( \alpha \)
Operational Semantics for StoKlaim

Stochastic semantics of StoKlaim is defined by means of a transition relation $\xrightarrow{\alpha}$ that associates to a process $P$ and a transition label $\alpha$ a function $(P, Q, \ldots)$ that maps each process into a non-negative real number.

$P \xrightarrow{\alpha} P'$ means that:

- if $P(Q) = x$ ($\neq 0$) then $Q$ is reachable from $P$ via the execution of $\alpha$ with rate or weight $x$
- if $P(Q) = 0$ then $Q$ is not reachable from $P$ via $\alpha$

We have that if $P \xrightarrow{\alpha} P'$ then

- $\oplus P = \sum Q P(Q)$ represents the total rate/weight of $\alpha$ in $P$. 

M. Loreti (DSIUF)  Model Checking MoSL  25/06/2009  9 / 30
Rate transition systems...

**Definition (Rate Transition Systems)**

A rate transition system is a triple \((S, A, \rightarrow)\) where:

- \(S\) is a set of states;
- \(A\) is a set of transition labels;
- \(\rightarrow \subseteq S \times A \times [S \rightarrow \mathbb{R}_{\geq 0}]\)
Rate transition systems...

**Definition (Rate Transition Systems)**

A rate transition system is a triple \((S, A, \rightarrow)\) where:

- \(S\) is a set of states;
- \(A\) is a set of transition labels;
- \(\rightarrow \subseteq S \times A \times [S \rightarrow \mathbb{R}_{\geq 0}]\)
Notations:

- RTS will be denoted by \( \mathcal{R}, \mathcal{R}_1, \mathcal{R}', \ldots \)
- Elements of \( [S \rightarrow \mathbb{R}_{\geq 0}] \) are denoted by \( P, Q, R, \ldots \)
- \( \emptyset \) denotes the constant function 0
- \( [s_1 \mapsto v_1, \ldots, s_n \mapsto v_n] \) identifies a function associating \( v_i \) to \( s_i \) and 0 to all the other states.
- \( \chi_s \) stands for \( [s \mapsto 1] \).
- \( P + Q \) denotes the function \( \mathcal{R} \) such that: \( \mathcal{R}(s) = P(s) + Q(s) \).
- \( P \cdot \frac{x}{y} \) denotes function \( \mathcal{R} \) such that: \( \mathcal{R}(s) = P(s) \cdot \frac{x}{y} \) if \( y \neq 0 \), and \( \emptyset \) if \( y = 0 \).
MoSL: General

1. a *temporal logic* (dynamic evolution);
2. both *action*- and *state*-based;
3. a *real-time* logic (real-time bounds);
4. a *probabilistic logic* (performance and dependability aspects);
5. a *spatial logic* (spatial structure of the network).
MoSL: Atomic propositions

\[ \mathcal{N} ::= Q(\vec{Q}', \vec{\ell}, \vec{e}) @ \rightarrow \Phi \mid \langle \vec{F} \rangle @ \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e}) @ \leftarrow \Phi \mid \langle \vec{f} \rangle @ \leftarrow \Phi \]
\( \Delta ::= Q(\vec{Q}', \vec{l}, \vec{e})@i \rightarrow \Phi \mid \langle \vec{F} \rangle@i \rightarrow \Phi \mid Q(\vec{Q}', \vec{l}, \vec{e})@i \leftarrow \Phi \mid \langle \vec{f} \rangle@i \leftarrow \Phi \)

**Process Consumption:**

Holds for a network whenever in the network there exists a process \( Q \) running at site \( i \), and the “remaining” network satisfies \( \Phi \).
MoSL: Atomic propositions

\[ \mathcal{N} ::= Q(\vec{Q}', \vec{\ell}, \vec{e})@ı \rightarrow \Phi \mid \langle \vec{F} \rangle @ı \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e})@ı \leftarrow \Phi \mid \langle \vec{f} \rangle @ı \leftarrow \Phi \]

**Tuple Consumption:**
Holds whenever a tuple $\vec{f}$ matching $\vec{F}$ is stored in a node of site ı and the “remaining” network satisfies $\Phi$. 
MoSL: Atomic propositions

\[ \forall i \::= \ Q(\vec{Q}', \vec{l}, \vec{e})@i \rightarrow \Phi \mid \langle \vec{F} \rangle@i \rightarrow \Phi \mid \ Q(\vec{Q}', \vec{l}, \vec{e})@i \leftarrow \Phi \mid \langle \vec{f} \rangle@i \leftarrow \Phi \]

**Process Production:**

Holds if the network satisfies \( \Phi \) whenever process \( Q(\vec{Q}', \vec{l}, \vec{e}) \) is executed at site \( i \).
MoSL: Atomic propositions

\[ \mathcal{N} ::= Q(\vec{Q}', \vec{\ell}, \vec{e})@i \rightarrow \Phi \mid \langle \vec{F} \rangle@i \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e})@i \leftarrow \Phi \mid \langle \vec{f} \rangle@i \leftarrow \Phi \]

**Tuple Production:**

Holds if the network satisfies $\Phi$ whenever tuple $\vec{f}$ is stored at site $i$. 
MoSL: State formulae

Φ ::= tt | ℵ | ¬ Φ | Φ ∨ Φ

CSL path-operator: P ⊘◁ p (ϕ) Satisfied by a state s iff the total probability mass for all paths starting in s that satisfy ϕ meets the bound ⊘◁ p;

CSL Steady-state operator: S ⊘◁ p (Φ) Satisfied by a state s iff the probability of reaching from s, in the long run, a state which satisfies Φ is ⊘◁ p.
MoSL: State formulae

\[ \Phi ::= \text{tt} | \aleph | \neg \Phi | \Phi \lor \Phi | P_{\nabla p}(\varphi) \]

with \( \nabla \in \{<, >, \leq, \geq\} \) and \( p \in [0, 1] \)

CSL path-operator: \( P_{\nabla p}(\varphi) \)
Satisfied by a state \( s \) iff the total probability mass for all paths starting in \( s \) that satisfy \( \varphi \) meets the bound \( \nabla p \);
\[
\Phi ::= \text{tt} | \exists | \neg \Phi | \Phi \lor \Phi | P_{\bowtie p}(\varphi) | S_{\bowtie p}(\Phi)
\]

with \(\bowtie \in \{<, >, \leq, \geq\}\) and \(p \in [0, 1]\)

**CSL path-operator:** \(P_{\bowtie p}(\varphi)\)

Satisfied by a state \(s\) iff the total probability mass for all paths starting in \(s\) that satisfy \(\varphi\) meets the bound \(\bowtie p\);

**CSL Steady-state operator:** \(S_{\bowtie p}(\Phi)\)

Satisfied by a state \(s\) iff the probability of reaching from \(s\), in the long run, a state which satisfies \(\Phi\) is \(\bowtie p\).
MoSL: Path formulae

\[
\Phi \models \Delta \mathcal{U}^{<t} \Omega \Psi
\]

- Satisfied by those paths where eventually a \( \Psi \)-state is reached, by time \( t \), via a \( \Phi \)-path, and, in addition, while evolving between \( \Phi \) states, actions are performed satisfying \( \Delta \) and the \( \Psi \)-state is entered via an action satisfying \( \Omega \).
MoSL: Path formulae

\[ \Phi \, \Delta \, U^{<t} \, \Omega \, \Psi \]

- Satisfied by those paths where eventually a \( \Psi \)-state is reached, by time \( t \), via a \( \Phi \)-path, and, in addition, while evolving between \( \Phi \) states, actions are performed satisfying \( \Delta \) and the \( \Psi \)-state is entered via an action satisfying \( \Omega \).

- Instantiations of variables in \( \Omega \) act as binders \( \Psi \).
MoSL: Path formulae

\[ \Phi \triangleleft U^t \Psi \]

- Satisfied by those paths where eventually a \( \Psi \)-state is reached, by time \( t \), via a \( \Phi \)-path, and, in addition, while evolving between \( \Phi \) states, actions are performed satisfying \( \triangle \) and the \( \Psi \)-state is entered via an action satisfying \( \Omega \).

- Instantiations of variables in \( \Omega \) act as binders \( \Psi \).

- Simpler operator: \( \Phi \triangleleft U^t \Psi \).
MoSL: Path formulae

\[ \Phi \ 
\Delta \ U^t \ < \ 
\Omega \ 
\Psi \]

- Satisfied by those paths where eventually a \( \Psi \)-state is reached, by time \( t \), via a \( \Phi \)-path, and, in addition, while evolving between \( \Phi \) states, actions are performed satisfying \( \Delta \) and the \( \Psi \)-state is entered via an action satisfying \( \Omega \).

- Instantiations of variables in \( \Omega \) act as binders \( \Psi \).

- Simpler operator: \( \Phi \ 
\Delta \ U^t \ < \ 
\Psi \).

- Time \( t \) can be omitted (assumed as \( \infty \)).
MoSL: Path formulae

\[ \Phi \quad \Delta \quad \mathcal{U}_{<t} \quad \Psi \]

- Satisfied by those paths where eventually a \( \Psi \)-state is reached, by time \( t \), via a \( \Phi \)-path, and, in addition, while evolving between \( \Phi \) states, actions are performed satisfying \( \Delta \) and the \( \Psi \)-state is entered via an action satisfying \( \Omega \).

- Instantiations of variables in \( \Omega \) act as binders \( \Psi \).

- Simpler operator: \( \Phi \quad \Delta \quad \mathcal{U}_{<t} \quad \Psi \).

- Time \( t \) can be omitted (assumed as \( \infty \)).

\[ \text{tt} \quad \mathcal{T} \quad \mathcal{U}_{<t} \{ \text{init:O(GO,A)} \} \quad \text{tt} \]
MoSL: Path formulae

\[ \Phi \ \triangleleft U^<t \ \Psi \]

- Satisfied by those paths where eventually a \( \Psi \)-state is reached, by time \( t \), via a \( \Phi \)-path, *and*, in addition, while evolving between \( \Phi \) states, actions are performed satisfying \( \triangle \) and the \( \Psi \)-state is entered via an action satisfying \( \Omega \).

- Instantiations of variables in \( \Omega \) act as binders \( \Psi \).

- Simpler operator: \( \Phi \ \triangleleft U^<t \ \Psi \).

- Time \( t \) can be omitted (assumed as \( \infty \)).

\[ \text{tt} \ \top U^<t \{ \text{init:O(GO,A)} \} \text{tt} \quad \text{tt} \ \top U^<t \langle GO \rangle @A \]
MoSL: Path formulae

\[ \Phi \bigtriangleup \mathcal{U}^{<t} \Psi \]

- Satisfied by those paths where eventually a \( \Psi \)-state is reached, by time \( t \), via a \( \Phi \)-path, and, in addition, while evolving between \( \Phi \) states, actions are performed satisfying \( \bigtriangleup \) and the \( \Psi \)-state is entered via an action satisfying \( \Omega \).

- Instantiations of variables in \( \Omega \) act as binders \( \Psi \).

- Simpler operator: \( \Phi \bigtriangleup \mathcal{U}^{<t} \Psi \).

- Time \( t \) can be omitted (assumed as \( \infty \)).

\[ \text{tt} \quad \mathcal{T} \mathcal{U}^{<t}_{\{\text{init:O(GO,A)}\}} \quad \text{tt} \quad \mathcal{T} \mathcal{U}^{<t}_{\langle GO \rangle @A} \quad \text{tt} \quad \mathcal{T} \mathcal{U}^{<\infty}_{\{i_1:N(!z)\}} \quad \text{nil}@z \]
init : o(GO, A)

Satisfied by any action executed at site init, by means of which a process uploads value GO to site A;
init : o(GO, A)

Satisfied by any action executed at site *init*, by means of which a process *uploads* value *GO* to site *A*;

!z_1 : o(GO, !z_2)

Satisfied by any action, executed at *some* site (*z_1*), by means of which a process *uploads* value *GO* to *some* site (*z_2*);
MoSL: Action specifiers and action sets

\begin{equation}
\text{init} : o(GO, A)
\end{equation}

Satisfied by any action executed at site \textit{init}, by means of which a process \textit{uploads} value \textit{GO} to site \textit{A};

\begin{equation}
!z_1 : o(GO, !z_2)
\end{equation}

Satisfied by any action, executed at some site \textit{(z}_1\textit{)}, by means of which a process \textit{uploads} value \textit{GO} to some site \textit{(z}_2\textit{)};

\begin{equation}
\Delta = \{\xi_1, \ldots \xi_n\} \text{ satisfied by any action satisfying any of } \xi_j;
\end{equation}
**MoSL: Action specifiers and action sets**

\[ \text{init} : o(GO, A) \]

Satisfied by any action executed at site \textit{init}, by means of which a process uploads value \textit{GO} to site \textit{A};

\[ !z_1 : o(GO, !z_2) \]

Satisfied by any action, executed at some site \((z_1)\), by means of which a process uploads value \textit{GO} to some site \((z_2)\);

\[ \Delta = \{\xi_1, \ldots, \xi_n\} \text{ satisfied by any action satisfying any of } \xi_j; \]

\[ \top : \text{satisfied by } \text{any action.} \]
We present a strategy for model checking MoSL formulae against StoKlaim models. Model-checking of RTSs is performed by using a CSL model checker. The proposed model-checking algorithm manipulates the input RTS obtained from a StoKlaim specification through the following steps: the RTS to be model-checked is translated into an equivalent state-labelled CTMC, and then the obtained CTMC is analysed by making use of existing (state-based) CSL model checkers.
Model Checking MoSL

- We present a strategy for model checking MoSL formulae against StoKlaim models.
Model Checking MoSL

- We present a strategy for model checking MoSL formulae against StoKlaim models.
- Model-checking of RTSs is performed by using a CSL model checker.
Model Checking MoSL

- We present a strategy for model checking MoSL formulae against StoKlaim models.
- Model-checking of RTSs is performed by using a CSL model checker.
- The proposed model-checking algorithm manipulates the input RTS obtained from a StoKlaim specification.
Model Checking MoSL

- We present a strategy for model checking MoSL formulae against StoKlaim models.
- Model-checking of RTSs is performed by using a CSL model checker.
- The proposed model-checking algorithm manipulates the input RTS obtained from a StoKlaim specification:
  - the RTS to be model-checked is translated into an equivalent state-labelled CTMC
  - obtained CTMC is then analysed by making use of existing (state-based) CSL model checkers.
Model Checking MoSL...

N ⊕ (i, E) denotes the net obtained from N by adding element E at address i.

N ⊖ (i, E) denotes the net obtained from N by removing existing element E from i.

Let C be a set of StoKlaim nets:

- C ⊕ (i, E) denotes the set of N ⊕ (i, E) such that N ∈ C;
- C ⊖ (i, E) denotes the set of N ⊖ (i, E) such that N ∈ C;

R[C] denotes the RTS generated starting from the set of nets C.

R ⊕ (i, E) denotes the RTS obtained from R by adding (i, E).

R ⊖ (i, E) denotes the RTS obtained from R by removing (i, E).
Model Checking MoSL...

- $N \oplus (i, E)$ denotes the net obtained from $N$ by adding element $E$ at address $i$
- $N \ominus (i, E)$ denotes the net obtained from $N$ by removing existing element $E$ from $i$
Model Checking MoSL...

- $N \oplus (i, E)$ denotes the net obtained from $N$ by adding element $E$ at address $i$
- $N \ominus (i, E)$ denotes the net obtained from $N$ by removing existing element $E$ from $i$
- Let $C$ be a set of StoKlaim nets:
  - $C \oplus (i, E)$ denotes the set of $N \oplus (i, E)$ such that $N \in C$;
  - $C \ominus (i, E)$ denotes the set of $N \ominus (i, E)$ such that $N \in C$;
Model Checking MoSL...

- $N \oplus (i, E)$ denotes the net obtained from $N$ by adding element $E$ at address $i$
- $N \ominus (i, E)$ denotes the net obtained from $N$ by removing existing element $E$ from $i$

Let $C$ be a set of $\text{StoKlaim}$ nets:
- $C \oplus (i, E)$ denotes the set of $N \oplus (i, E)$ such that $N \in C$;
- $C \ominus (i, E)$ denotes the set of $N \ominus (i, E)$ such that $N \in C$;

- $\mathcal{R}[C]$ denotes the RTS generated starting from the set of nets $C$
Model Checking MoSL...

- $N \oplus (i, E)$ denotes the net obtained from $N$ by adding element $E$ at address $i$
- $N \ominus (i, E)$ denotes the net obtained from $N$ by removing existing element $E$ from $i$
- Let $C$ be a set of StoKlaim nets:
  - $C \oplus (i, E)$ denotes the set of $N \oplus (i, E)$ such that $N \in C$;
  - $C \ominus (i, E)$ denotes the set of $N \ominus (i, E)$ such that $N \in C$;
- $\mathcal{R}[C]$ denotes the RTS generated starting from the set of nets $C$
- $\mathcal{R} \oplus (i, E)$ denotes the RTS obtained from $\mathcal{R}$ by adding $(i, E)$
- $\mathcal{R} \ominus (i, E)$ denotes the RTS obtained from $\mathcal{R}$ by removing $(i, E)$
Idea:
A finite RTS $R$ is translated into a finite, state-labelled, CTMC ($K(R)$). The states of such CTMC will contain information which will be used by the model-checking algorithm; consequently, a distinct duplicate of each state is created in the target CTMC. In order to consider the first transition delay correctly, one additional $\perp$-labelled duplicate is added for each state. The outgoing transitions of these duplicate states have the same target and same rate as those of the original state.

▶ All copies of state $s$ in the target CTMC are strong Markovian bisimilar and therefore enjoy the same transient and steady state properties.
Idea:

- A finite RTS $\mathcal{R}$ is translated into a finite, state-labelled, CTMC ($\mathcal{K}(\mathcal{R})$).
- The states of such CTMC will contain information which will be used by the model-checking algorithm; consequently...

All copies of state $s$ in the target CTMC are strong Markovian bisimilar and therefore enjoy the same transient and steady state properties.
Model Checking MoSL... 

Idea:
- A finite RTS $\mathcal{R}$ is translated into a finite, state-labelled, CTMC ($\mathcal{K}(\mathcal{R})$)
- The states of such CTMC will contain information which will be used by the model-checking algorithm; consequently

Translation:
Model Checking MoSL...

Idea:
- A finite RTS $\mathcal{R}$ is translated into a finite, state-labelled, CTMC ($\mathcal{K}(\mathcal{R})$).
- The states of such CTMC will contain information which will be used by the model-checking algorithm; consequently.

Translation:
- For each state $s$ in $\mathcal{R}$, and for each transition pointing to $s$ labelled by an action $a$, a distinct duplicate of $s$, labelled by $a$, is created in the target CTMC.
**Idea:**

- A finite RTS $\mathcal{R}$ is translated into a finite, state-labelled, CTMC ($\mathcal{K}(\mathcal{R})$).
- The states of such CTMC will contain information which will be used by the model-checking algorithm; consequently.

**Translation:**

- For each state $s$ in $\mathcal{R}$, and for each transition pointing to $s$ labelled by an action $a$, a distinct duplicate of $s$, labelled by $a$, is created in the target CTMC.
- In order to consider the first transition delay correctly, one additional $\perp$-labelled duplicate is added for $s$. 


Idea:

- A finite RTS $\mathcal{R}$ is translated into a finite, state-labelled, CTMC ($\mathcal{K}(\mathcal{R})$)
- The states of such CTMC will contain information which will be used by the model-checking algorithm; consequently

Translation:

- For each state $s$ in $\mathcal{R}$, and for each transition pointing to $s$ labelled by an action $a$, a distinct duplicate of $s$, labelled by $a$, is created in the target CTMC.
- In order to consider the first transition delay correctly, one additional $\bot$-labelled duplicate is added for $s$.
- The outgoing transitions of these duplicate states have the same target and same rate as those of the original state.
Idea:

- A finite RTS $\mathcal{R}$ is translated into a finite, state-labelled, CTMC ($\mathcal{K}(\mathcal{R})$).
- The states of such CTMC will contain information which will be used by the model-checking algorithm; consequently.

Translation:

- For each state $s$ in $\mathcal{R}$, and for each transition pointing to $s$ labelled by an action $a$, a distinct duplicate of $s$, labelled by $a$, is created in the target CTMC.
- In order to consider the first transition delay correctly, one additional $\perp$-labelled duplicate is added for $s$.
- The outgoing transitions of these duplicate states have the same target and same rate as those of the original state.
  - All copies of state $s$ in the target CTMC are strong Markovian bisimilar and therefore enjoy the same transient and steady state properties.
An example...
An example...

From RTS...

...to CTMC

\[ \begin{array}{cccc}
  s_1, \lambda_3 & s_1, \lambda_5 & s_1, \lambda_4 & s_2, b \\
  s_2, \lambda_3 & s_2, \lambda_5 & s_2, \lambda_4 & s_2, \lambda_6 \\
\end{array} \]
An example...

From RTS...

...to CTMC
An example... From RTS... to CTMC
An example…

From RTS…

…to CTMC
An example...

From RTS...

\[ \lambda_1 \]

\[ \lambda_2 \]

\[ \lambda_3 \]

\[ \lambda_4 \]

\[ \lambda_5 \]

\[ \lambda_6 \]

\[ a \]

\[ b \]

\[ s_1 \rightarrow s_2 \]

\[ s_2 \rightarrow s_1 \]

\[ s_2 \rightarrow s_2 \]

\[ \ldots \text{to CTMC} \]

\[ s_1 \]

\[ s_2 \]

\[ s_1 \rightarrow s_2 \]

\[ s_2 \rightarrow s_1 \]

\[ s_2 \rightarrow s_2 \]

\[ \lambda_1 \]

\[ \lambda_2 \]

\[ \lambda_3 \]

\[ \lambda_4 \]

\[ \lambda_5 \]

\[ \lambda_6 \]
Definition

For each RTS \( R \) and for each MoSL formula \( \Phi \), \( \text{Sat}(\Phi, R) \) returns the set of all states of \( R \) which satisfy \( \Phi \), and is defined recursively on the structure of \( \Phi \) as follows:

\[
\begin{align*}
\text{Sat}(\neg \Phi, R) &= \text{Sat}(\Phi, R) \\
\text{Sat}(\Phi \lor \Psi, R) &= \text{Sat}(\Phi, R) \cup \text{Sat}(\Psi, R)
\end{align*}
\]
Model Checking Algorithm

Definition

For each RTS $\mathcal{R}$ and for each MoSL formula $\Phi$, $Sat(\Phi, \mathcal{R})$ returns the set of all states of $\mathcal{R}$ which satisfy $\Phi$, and is defined recursively on the structure of $\Phi$ as follows:

- $Sat(tt, \mathcal{R}) \overset{\text{def}}{=} S$
- $Sat(\neg \Phi, \mathcal{R}) \overset{\text{def}}{=} S \setminus Sat(\Phi, \mathcal{R})$
- $Sat(\Phi \lor \Psi, \mathcal{R}) \overset{\text{def}}{=} Sat(\Phi, \mathcal{R}) \cup Sat(\Psi, \mathcal{R})$
- \ldots
Model Checking Algorithm

Definition

- ... 

\[ Sat(P_{\bowtie p}(\Phi \Delta U_{\Omega}^{\leq t} \Psi), R) \overset{\text{def}}{=} \]

let \( S_1 = Sat(\Phi, R) \times (\Delta \cup \{\bot\}) \) in

let \( S_2 = Sat(\Psi, R) \times \Omega \) in

\[ \{s \in S \mid (s, \bot) \in \text{until}(\bowtie, p, t, S_1, S_2, \mathcal{K}(R))\} \]

- ...
Model Checking Algorithm

Definition

- \( \text{Sat}(P \bowtie p(\Phi \triangleleft U^t \psi), R) \) \( \overset{\text{def}}{=} \)
- let \( S_1 = \text{Sat}(\Phi, R) \times (\Delta \cup \{\bot\}) \) in
- let \( S_2 = \text{Sat}(\Psi, R) \times \Omega \) in

\[ \{ s \in S \mid (s, \bot) \in \text{until}(\bowtie, p, t, S_1, S_2, K(R)) \} \]

Computation of function \text{until} relies on an existing Stochastic Model Checker like, for instance, MRMC.
Model Checking Algorithm

Definition

- \( \text{Sat}(\langle \vec{f} \rangle \circ i \rightarrow \Psi, \mathcal{R}) = \{ s | s \oplus (i, \vec{f}) \in \text{Sat}(\Psi, \mathcal{R} \oplus (i, \vec{f})) \} \)
- \( \text{Sat}(\langle \vec{f} \rangle \circ i \leftarrow \Psi, \mathcal{R}) = \{ s | s \oplus (i, \vec{f}) \in \text{Sat}(\Psi, \mathcal{R} \oplus (i, \vec{f})) \} \)
Distributed Mobile Service Example

- A service is built on two sites, $A$ and $B$;
- Client software and service dispatcher run on $A$;
- two types of services are available, S1 and S2:
  - each S1-service request is satisfied using local resources only (i.e. in $A$)
  - each S2-service request requires
    - first, some computation at $A$
    - followed by, a computation at $B$
  \[ \Rightarrow \text{thus the agent taking care of the request is launched in } A \text{ and then migrates to } B. \]
In the long run, the probability that the computational resource located at $A$ is available is greater than 0.2:

$$S > 0.$$ If, in the current state, at site $A$ a request of an $S_2$ service is issued, the probability that it gets served within 72.04 time-units is greater than 0.85:

$$\langle S_2 \rangle @ A \Rightarrow P > 0.85 \left( tt \top U < 72.04 \left\{ A : I( S_2, A) \right\} \right).$$

In the long run, there is a probability greater than 0.87, that a request for $S_1$-service is issued at site $A$ within 500 time units with a probability greater than 0.75:

$$S > 0.87 \left( P > 0.75 \left( tt \top U < 500 \left\{ !z : O( S_1, A) \right\} \right) \right).$$
MoSL: DMS Example

- In the long run, the probability that the computational resource located at $A$ is available is greater than 0.2:
MoSL: DMS Example

- In the long run, the probability that the computational resource located at $A$ is available is greater than 0.2:

$$S_{>0.2}(\langle AF \rangle @ A)$$
MoSL: DMS Example

- In the long run, the probability that the computational resource located at A is available is greater than 0.2:

\[ S_{>0.2}(⟨AF⟩@A) \]

- If, in the current state, at site A a request of an S2 service is issued, the probability that it gets served within 72.04 time-units is greater than 0.85:

\[ S_{>0.2}(⟨AF⟩@A) \]
MoSL: DMS Example

- In the long run, the probability that the computational resource located at $A$ is available is greater than 0.2:

$$S_{>0.2}(\langle AF \rangle @ A)$$

- If, in the current state, at site $A$ a request of an $S2$ service is issued, the probability that it gets served within $72.04$ time-units is greater than 0.85:

$$\langle S2 \rangle @ A \Rightarrow P_{>0.85}(tt \uparrow U^{<72.04}_{\{A: \text{i}(S2, A)\}} tt)$$
MoSL: DMS Example

- In the long run, the probability that the computational resource located at A is available is greater than 0.2:

\[ S > 0.2(\langle AF \rangle @ A) \]

- If, in the current state, at site A a request of an S2 service is issued, the probability that it gets served within 72.04 time-units is greater than 0.85:

\[ \langle S2 \rangle @ A \Rightarrow P > 0.85(\langle tt U (\{ A : I(S2,A) \} tt) \rangle t) \]

- In the long run, there is a probability greater than 0.87, that a request for S1-service is issued at site A within 500 time units with a probability greater than 0.75:
In the long run, the probability that the computational resource located at $A$ is available is greater than 0.2:

$$S_{>0.2}(\langle AF \rangle @ A)$$

If, in the current state, at site $A$ a request of an $S_2$ service is issued, the probability that it gets served within 72.04 time-units is greater than 0.85:

$$\langle S_2 \rangle @ A \Rightarrow P_{>0.85}(tt \uparrow \mathcal{U}^{<72.04}_{\{A:l(S_2,A)\}} tt)$$

In the long run, there is a probability greater than 0.87, that a request for $S_1$-service is issued at site $A$ within 500 time units with a probability greater than 0.75:

$$S_{>0.87}(P_{>0.75}(tt \uparrow \mathcal{U}^{<500}_{\{!z:o(S_1,A)\}} tt))$$
Simulating StoKlaim Networks: DMS
Simulating **StoKlaim** Networks...

- In **StoKlaim** the number of tuples matching a given template does not alter the rate of executing action.
- Sometimes one is interested in increasing the rate of an input/read action when more instances of a same tuple are available (biological applications).
Simulating StoKlaim Networks... 

Example

Let $A = (\text{in}(X)\oplus, \lambda).A$:

$$I :: A \| I :: \langle X \rangle \| \cdots \| I :: \langle X \rangle$$
Example

Let \( A = (\text{in}(X) @, \lambda).A : \)

\[ I :: A || I :: \langle X \rangle || \cdots || I :: \langle X \rangle \]
Example

Let $A = (\text{in}(X)@,\lambda).A$:

$I :: A || I :: \langle X \rangle || \cdots || I :: \langle X \rangle$
Simulating **StoKlaim** Networks: DMS
Simulating StoKlaim Networks: DMS (bio)
Concluding Remarks

StoKlaim and MoSL can be used for specifying and verifying properties of mobile and distributed systems. The proposed tool (SAM) permits:

▶ verifying whether a given system satisfies or not a given property (by relying on MRMC)
▶ simulate system behaviour.

On-going work:

Investigating direct (on-the-fly) model-checking algorithms for the logic and StoKlaim
▶ An on-the-fly model-checker for PCTL is under construction
▶ Define an ODE semantics of StoKlaim to predict behaviour of StoKlaim systems
▶ Simulation and model checking will be used to validate the obtained results

M. Loreti (DSIUF)
StoKlaim and MoSL can be used for specifying and verifying properties of mobile and distributed systems.

The proposed tool (SAM) permits:
- verifying whether a given system satisfies or not a given property (by relying on MRMC)
- simulate system behaviour.

On going work:
- Investigating direct (on-the-fly) model-checking algorithms for the logic and StoKlaim
- An on-the-fly model-checker for PCTL is under construction
- Define an ODE semantics of StoKlaim to predict behaviour of StoKlaim systems
- Simulation and model checking will be used to validate the obtained results
Concluding Remarks

- **StoKlaim** and **MoSL** can be used for specifying and verifying properties of mobile and distributed systems.

- The proposed tool (SAM) permits:
  - verifying whether a given system satisfies or not a given property (by relying on MRMC)
  - simulate system behaviour.

**On going work:**

- Investigating direct (on-the-fly) model-checking algorithms for the logic and **StoKlaim**
  - An on-the-fly model-checker for PCTL is under construction

- Define an ODE semantics of **StoKlaim** to predict behaviour of **StoKlaim** systems
  - Simulation and model checking will be used to validate the obtained results
THANK YOU FOR YOUR ATTENTION