Model Checking Mobile Stochastic Logics

Rocco De Nicola[1] Joost-Pieter Katoen[2]
Diego Latella[3]
Michele Loreti[1] Mieke Massink[3]

[1]DSIUF- Università di Firenze
[2]RHWT - University of Aachen
[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.

From Linda and Process Algebras to KLAIM

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

- $(\mathbf{out}(T)@l2, r1)$
  - *uploads* tuple T to $l2$,
  - *the time it takes* is e.d. with rate $r1$
- $(\mathbf{eval}(P)@l1, r2)$
  - *spawns* process P to $l1$,
  - *the time it takes* is e.d. with rate $r2$
- $(\mathbf{newloc}(!u), r3)$
  - *creates* a new site (with locality) u,
  - *the time it takes* is e.d. with rate $r3$
- $(\mathbf{in}(F)@l1, r4)$
  - *downloads*, if available, a tuple matching F from $l1$,
  - *it takes a time* which is e.d. with rate $r4$,
- $(\mathbf{read}(F)@l1, r4)$
  - *reads*, if available, a tuple matching F from $l1$, without consuming it
  - *it takes a time* which is e.d. with rate $r4$,

StoKlaim Syntax

Nets: $N ::= \mathbf{0} \mid i ::_\rho E \mid N \parallel N$

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

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

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

Tuple Fields: $f ::= P \mid \ell \mid e$

Template Fields: $F ::= f \mid !X \mid !u \mid !x$

Operational Semantics for StoKlaim

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

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

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

Operational Semantics for StoKlaim

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

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

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

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

- $\oplus\mathscr{P} = \sum_Q \mathscr{P}(Q)$ represents the total rate/weight of α in P.

Rate transition systems...

Definition (Rate Transition Systems)

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

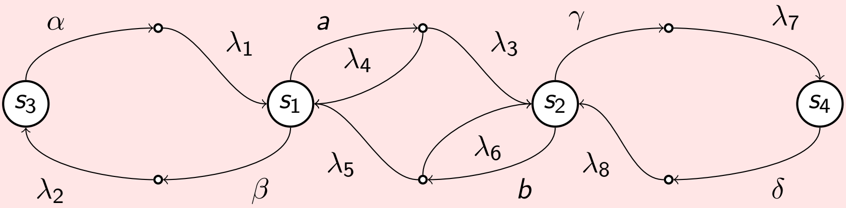
- S is a set of states;
- A is a set of transition labels;
- $\rightarrow \subseteq S \times A \times [S \rightarrow \mathbb{R}_{\geq 0}]$

Rate transition systems...

Definition (Rate Transition Systems)

A rate transition system is a triple $(S, A, \longrightarrow)$ 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. . .

Notations:

- RTS will be denoted by $\mathcal{R}$,$\mathcal{R}_1$,$\mathcal{R}'$,. . . ,
- Elements of $[S \rightarrow \mathbb{R}_{\geq 0}]$ are denoted by $\mathscr{P}, \mathscr{Q}, \mathscr{R}, \ldots$
- $\emptyset$ 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.
- χ_s stands for $[s \mapsto 1]$.
- $\mathscr{P} + \mathscr{Q}$ denotes the function $\mathscr{R}$ such that: $\mathscr{R}(s) = \mathscr{P}(s) + \mathscr{Q}(s)$.
- $\mathscr{P} \cdot \frac{x}{y}$ denotes function $\mathscr{R}$ such that: $\mathscr{R}(s) = \mathscr{P}(s) \cdot \frac{x}{y}$ 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

$$\aleph ::= Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \rightarrow \Phi \mid \langle \vec{F} \rangle @\imath \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \leftarrow \Phi \mid \langle \vec{f} \rangle @\imath \leftarrow \Phi$$

MoSL: Atomic propositions

$$\aleph ::= Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \rightarrow \Phi \mid \langle \vec{F} \rangle @\imath \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \leftarrow \Phi \mid \langle \vec{f} \rangle @\imath \leftarrow \Phi$$

Process Consumption:

Holds for a network whenever in the network there exists a process Q running at site $\imath$, and the "remaining" network satisfies Φ.

MoSL: Atomic propositions

$$\aleph ::= Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \rightarrow \Phi \mid \langle \vec{F} \rangle @\imath \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \leftarrow \Phi \mid \langle \vec{f} \rangle @\imath \leftarrow \Phi$$

Tuple Consumption:

Holds whenever a tuple $\vec{f}$ matching $\vec{F}$ is stored in a node of site $\imath$ and the "remaining" network satisfies Φ.

MoSL: Atomic propositions

$$\aleph ::= Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \rightarrow \Phi \mid \langle \vec{F} \rangle @\imath \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \leftarrow \Phi \mid \langle \vec{f} \rangle @\imath \leftarrow \Phi$$

Process Production:

Holds if the network satisfies Φ whenever process $Q(\vec{Q}', \vec{\ell}, \vec{e})$ is executed at site $\imath$.

MoSL: Atomic propositions

$$\aleph ::= Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \rightarrow \Phi \mid \langle \vec{F} \rangle @\imath \rightarrow \Phi \mid Q(\vec{Q}', \vec{\ell}, \vec{e})@\imath \leftarrow \Phi \mid \langle \vec{f} \rangle @\imath \leftarrow \Phi$$

Tuple Production:

Holds if the network satisfies Φ whenever tuple $\vec{f}$ is stored at site $\imath$.

MoSL: State formulae

$$\Phi ::= \mathrm{tt} \mid \aleph \mid \neg\,\Phi \mid \Phi \vee \Phi$$

MoSL: State formulae

$$\Phi ::= \mathrm{tt} \,|\, \aleph \,|\, \neg\, \Phi \,|\, \Phi \vee \Phi \,|\, \mathcal{P}_{\bowtie p}(\varphi)$$

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

CSL path-operator: $\mathcal{P}_{\bowtie p}(\varphi)$
Satisfied by a state s iff the total probability mass for all paths starting in s that satisfy φ meets the bound $\bowtie p$;

MoSL: State formulae

$$\Phi ::= tt \,|\, \aleph \,|\, \neg\, \Phi \,|\, \Phi \vee \Phi \,|\, \mathcal{P}_{\bowtie p}(\varphi) \,|\, \mathcal{S}_{\bowtie p}(\Phi)$$

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

CSL path-operator: $\mathcal{P}_{\bowtie p}(\varphi)$
Satisfied by a state s iff the total probability mass for all paths starting in s that satisfy φ meets the bound $\bowtie p$;

CSL Steady-state operator: $\mathcal{S}_{\bowtie p}(\Phi)$
Satisfied by a state s iff the probability of reaching from s, in the long run, a state which satisfies Φ is $\bowtie$ p.

MoSL: Path formulae

$$\Phi \;\, {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \;\Psi$$

- Satisfied by those paths where eventually a Ψ-state is reached, by time t, via a Φ-path, *and*, in addition, while evolving between Φ states, actions are performed satisfying Δ and the Ψ-state is entered via an action satisfying Ω.

MoSL: Path formulae

$$\Phi \; {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \; \Psi$$

- Satisfied by those paths where eventually a Ψ-state is reached, by time t, via a Φ-path, *and*, in addition, while evolving between Φ states, actions are performed satisfying Δ and the Ψ-state is entered via an action satisfying Ω.
- Instantiations of variables in Ω act as binders Ψ.

MoSL: Path formulae

$$\Phi \ {}_{\Delta}\mathcal{U}^{<t}_{\Omega}\ \Psi$$

- Satisfied by those paths where eventually a Ψ-state is reached, by time t, via a Φ-path, *and*, in addition, while evolving between Φ states, actions are performed satisfying Δ and the Ψ-state is entered via an action satisfying Ω.
- Instantiations of variables in Ω act as binders Ψ.
- Simpler operator: $\Phi \ {}_{\Delta}\mathcal{U}^{<t}\ \Psi$.

MoSL: Path formulae

$$\Phi \ {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \ \Psi$$

- Satisfied by those paths where eventually a Ψ-state is reached, by time t, via a Φ-path, *and*, in addition, while evolving between Φ states, actions are performed satisfying Δ and the Ψ-state is entered via an action satisfying Ω.
- Instantiations of variables in Ω act as binders Ψ.
- Simpler operator: $\Phi \ {}_{\Delta}\mathcal{U}^{<t} \ \Psi$.
- Time t can be omitted (assumed as ∞).

MoSL: Path formulae

$$\Phi \;{}_{\Delta}\mathcal{U}^{<t}_{\Omega}\; \Psi$$

- Satisfied by those paths where eventually a Ψ-state is reached, by time t, via a Φ-path, *and*, in addition, while evolving between Φ states, actions are performed satisfying Δ and the Ψ-state is entered via an action satisfying Ω.
- Instantiations of variables in Ω act as binders Ψ.
- Simpler operator: $\Phi \;{}_{\Delta}\mathcal{U}^{<t}\; \Psi$.
- Time t can be omitted (assumed as ∞).

$$\mathrm{tt}\;{}_{\top}\mathcal{U}^{<t}_{\{init:\mathbf{O}(GO,A)\}}\;\mathrm{tt}$$

MoSL: Path formulae

$$\Phi \ {}_{\Delta}\mathcal{U}^{<t}_{\Omega} \ \Psi$$

- Satisfied by those paths where eventually a Ψ-state is reached, by time t, via a Φ-path, *and*, in addition, while evolving between Φ states, actions are performed satisfying Δ and the Ψ-state is entered via an action satisfying Ω.
- Instantiations of variables in Ω act as binders Ψ.
- Simpler operator: $\Phi \ {}_{\Delta}\mathcal{U}^{<t} \ \Psi$.
- Time t can be omitted (assumed as ∞).

$$\mathrm{tt} \ {}_{\top}\mathcal{U}^{<t}_{\{init:\mathbf{O}(GO,A)\}} \ \mathrm{tt} \qquad \mathrm{tt} \ {}_{\top}\mathcal{U}^{\leq t}_{\top} \ \langle GO \rangle @A$$

MoSL: Path formulae

$$\Phi \ {}_{\Delta}\mathcal{U}^{<t}_{\Omega} \ \Psi$$

- Satisfied by those paths where eventually a Ψ-state is reached, by time t, via a Φ-path, *and*, in addition, while evolving between Φ states, actions are performed satisfying Δ and the Ψ-state is entered via an action satisfying Ω.
- Instantiations of variables in Ω act as binders Ψ.
- Simpler operator: $\Phi \ {}_{\Delta}\mathcal{U}^{<t} \ \Psi$.
- Time t can be omitted (assumed as ∞).

$$\mathsf{tt} \ {}_{\top}\mathcal{U}^{<t}_{\{init:\mathbf{O}(GO,A)\}} \ \mathsf{tt} \qquad \mathsf{tt} \ {}_{\top}\mathcal{U}^{<t}_{\top} \ \langle GO \rangle @ A \qquad \mathsf{tt} \ {}_{\top}\mathcal{U}^{<\infty}_{\{i_1:\mathbf{N}(!z)\}} \ \mathbf{nil}@z$$

MoSL: Action specifiers and action sets

$$init : \mathbf{o}(GO, A)$$

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

MoSL: Action specifiers and action sets

$$init : \mathbf{o}(GO, A)$$

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

$$!z_1 : \mathbf{o}(GO, !z_2)$$

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

$$init : \mathbf{o}(GO, A)$$

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

$$!z_1 : \mathbf{o}(GO, !z_2)$$

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

$\Delta = \{\xi_1, \dots \xi_n\}$ satisfied by any action satisfying any of ξ_j;

MoSL: Action specifiers and action sets

$$init : \mathbf{o}(GO, A)$$

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

$$!z_1 : \mathbf{o}(GO, !z_2)$$

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

$\Delta = \{\xi_1, \ldots \xi_n\}$ satisfied by any action satisfying any of ξ_j;

$\top$: satisfied by *any* action.

Model Checking MoSL

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…

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 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)

Model Checking MoSL...

Model Checking MoSL. . .

Idea:

- A finite RTS $\mathcal{R}$ is translete 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

Model Checking MoSL...

Idea:

- A finite RTS $\mathcal{R}$ is translete 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 translete 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

Model Checking MoSL...

Idea:

- A finite RTS $\mathcal{R}$ is translete 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.

Model Checking MoSL...

Idea:

- A finite RTS $\mathcal{R}$ is translete 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.

Model Checking MoSL. . .

Idea:

- A finite RTS $\mathcal{R}$ is translete 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

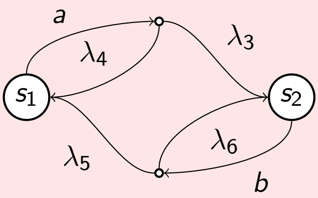
Translation:

- 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...

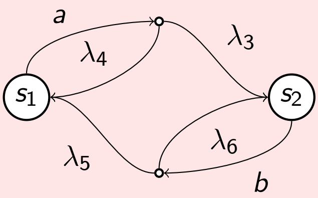
...to CTMC

From RTS...

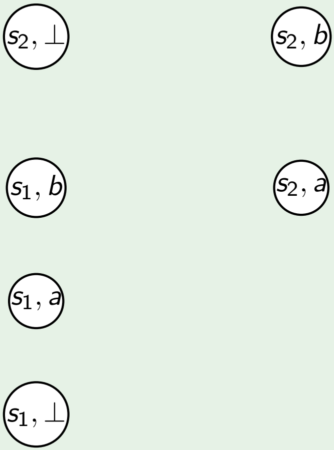


An example. . .

From RTS. . .

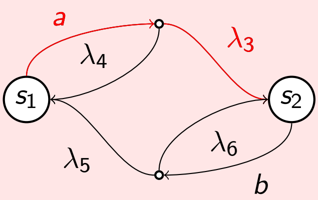


. . . to CTMC

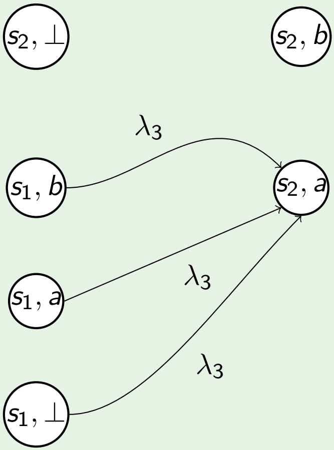


An example. . .

From RTS. . .

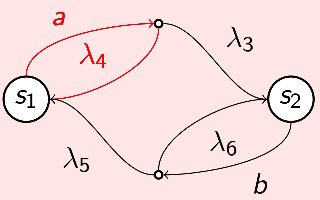


. . . to CTMC

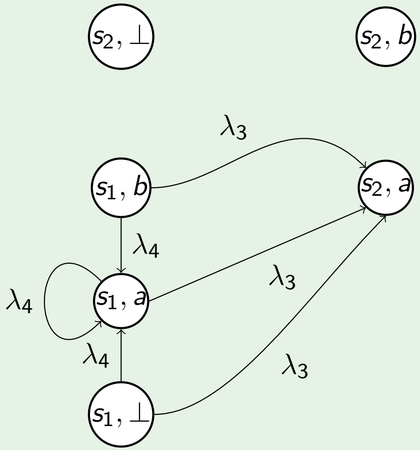


An example...

From RTS...

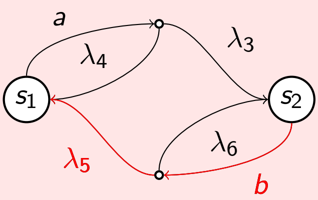


...to CTMC

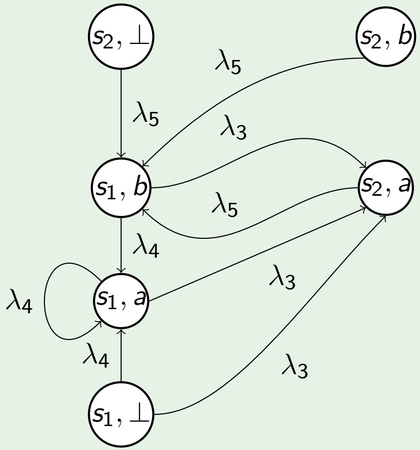


An example...

From RTS...

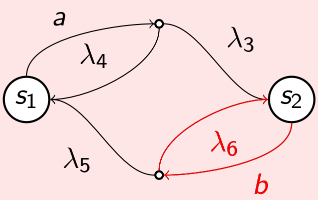


...to CTMC

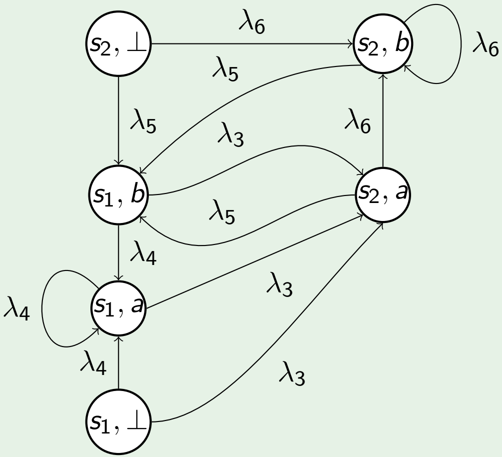


An example...

From RTS...



...to CTMC



Model Checking Algorithm

Definition

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

Model Checking Algorithm

Definition

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

- $Sat(\mathrm{tt}, \mathcal{R}) \stackrel{\text{def}}{=} S$
- $Sat(\neg\, \Phi, \mathcal{R}) \stackrel{\text{def}}{=} S \setminus Sat(\Phi, \mathcal{R})$
- $Sat(\Phi \,\vee\, \Psi, \mathcal{R}) \stackrel{\text{def}}{=} Sat(\Phi, \mathcal{R}) \cup Sat(\Psi, \mathcal{R})$
- ...

Model Checking Algorithm

Definition

- ...
- $Sat(\mathcal{P}_{\bowtie p}(\Phi \ {}_{\Delta}\mathcal{U}_{\Omega}^{<t} \ \Psi), \mathcal{R}) \stackrel{\text{def}}{=}$

$$\begin{array}{l}\text{let } S_1 = Sat(\Phi, \mathcal{R}) \times (\Delta \cup \{\bot\}) \text{ in} \\ \text{let } S_2 = Sat(\Psi, \mathcal{R}) \times \Omega \text{ in}\end{array}$$

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

- ...

Model Checking Algorithm

Definition

- ...
- $Sat(\mathcal{P}_{\bowtie p}(\Phi\ {}_{\Delta}\mathcal{U}_{\Omega}^{<t}\ \Psi), \mathcal{R}) \stackrel{\text{def}}{=}$

$$\begin{array}{l} \text{let } S_1 = Sat(\Phi, \mathcal{R}) \times (\Delta \cup \{\bot\}) \text{ in} \\ \text{let } S_2 = Sat(\Psi, \mathcal{R}) \times \Omega \text{ in} \end{array}$$

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

- ...

Computation of function *until* relies on an existing Stochastic Model Checker like, for instance, MRMC.

Model Checking Algorithm

Definition

- ...
- $Sat(\langle\vec{f}\rangle@i \rightarrow \Psi, \mathcal{R}) = \{s \mid s \ominus (i, \vec{f}) \in Sat(\Psi, \mathcal{R} \ominus (i, \vec{f}))\}$
- $Sat(\langle\vec{f}\rangle@i \leftarrow \Psi, \mathcal{R}) = \{s \mid s \oplus (i, \vec{f}) \in Sat(\Psi, \mathcal{R} \oplus (i, \vec{f}))\}$

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

    $\Rightarrow$ *thus* the *agent* taking care of the request is *launched* in A and then *migrates* to B.

MoSL: DMS Example

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:

$$\mathcal{S}_{>0.2}(\langle AF \rangle \texttt{@} A)$$

MoSL: DMS Example

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

$$\mathcal{S}_{>0.2}(\langle AF \rangle @ A)$$

- 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:

MoSL: DMS Example

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

$$\mathcal{S}_{>0.2}(\langle AF \rangle @ A)$$

- 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:

$$\langle S2 \rangle @ A \Rightarrow \ \mathcal{P}_{>0.85}(\mathrm{tt}\ {}_{\top}\mathcal{U}^{<72.04}_{\{A:\mathbf{I}(S2,A)\}}\ \mathrm{tt})$$

MoSL: DMS Example

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

$$\mathcal{S}_{>0.2}(\langle AF \rangle @ A)$$

- 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:

$$\langle S2 \rangle @ A \Rightarrow \ \mathcal{P}_{>0.85}(\mathrm{tt}\ {}_{\top}\mathcal{U}^{<72.04}_{\{A:\mathbf{l}(S2,A)\}}\ \mathrm{tt})$$

- 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:

MoSL: DMS Example

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

$$\mathcal{S}_{>0.2}(\langle AF \rangle @ A)$$

- 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:

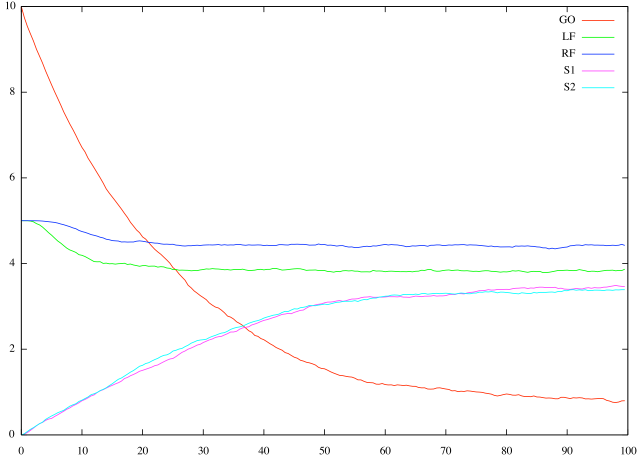
$$\langle S2 \rangle @ A \Rightarrow \; \mathcal{P}_{>0.85}(\mathrm{tt} \;_{\top}\mathcal{U}^{<72.04}_{\{A:\mathbf{I}(S2,A)\}} \; \mathrm{tt})$$

- 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:

$$\mathcal{S}_{>0.87}(\mathcal{P}_{>0.75}(\mathrm{tt} \;_{\top}\mathcal{U}^{<500}_{\{!z:\mathbf{O}(S1,A)\}} \; \mathrm{tt}))$$

Simulating STOKLAIM Networks

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

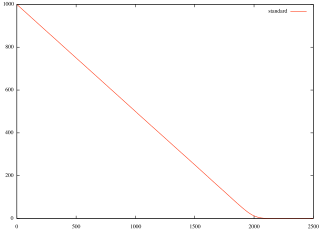
$$l :: A || l :: \langle X \rangle || \cdots || l :: \langle X \rangle$$

Simulating StoKlaim Networks...

Example

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

$$l :: A || l :: \langle X \rangle || \cdots || l :: \langle X \rangle$$

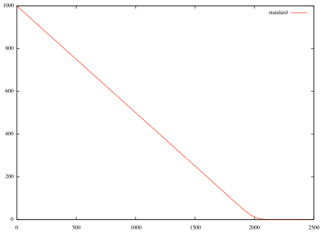


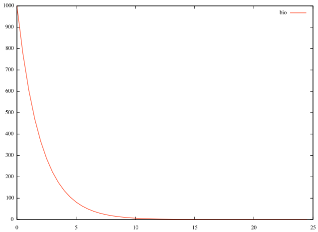
Simulating StoKlaim Networks...

Example

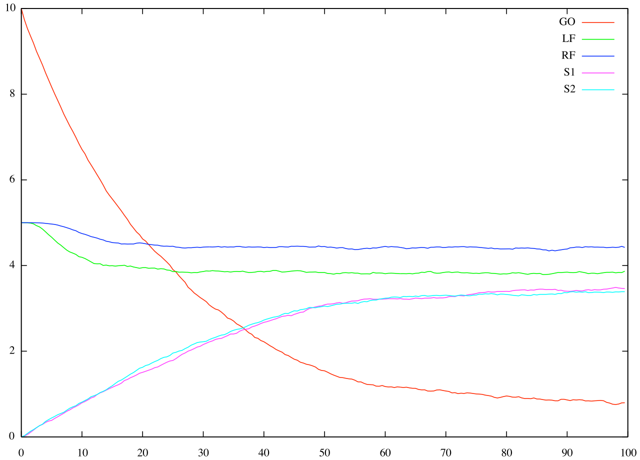
Let $A = (\mathbf{in}(X)@,\lambda).A$:

$$l :: A || l :: \langle X \rangle || \cdots || l :: \langle X \rangle$$

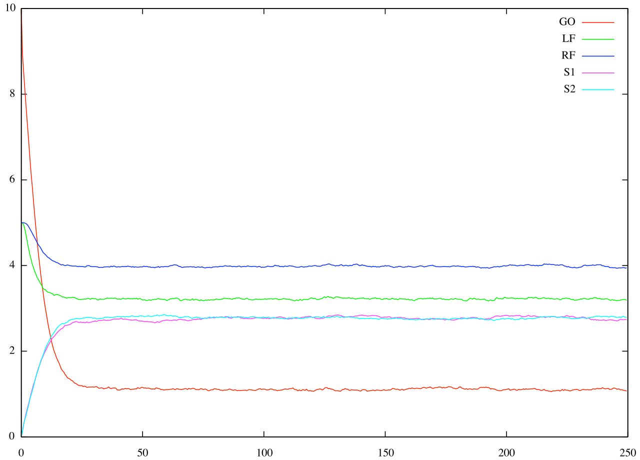




Simulating STOKLAIM Networks: DMS



Simulating StoKlaim Networks: DMS (bio)



Concluding Remarks

Concluding Remarks

- StoKlaim and MoSL can ben 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.

Concluding Remarks

- StoKlaim and MoSL can ben 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