Model Checking for Performance Analysis of Klaim Systems

Michele Loreti

DSIUF- Università di Firenze

L'Aquila, March 3, 2010

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 $(\mathscr{P}, \mathscr{Q}, \ldots)$ 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:

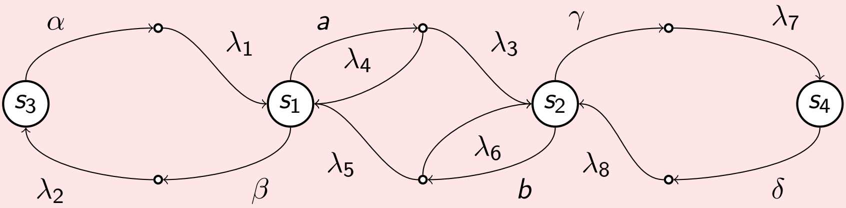
- S is a set of states;
- A is a set of transition labels;
- $\rightarrow \subseteq S \times A \times [S \rightarrow \mathbb{R}_{\geq 0}]$

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;
- $\rightarrow \subseteq S \times A \times [S \rightarrow \mathbb{R}_{\geq 0}]$



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

$$\Phi ::= \mathsf{tt} \,|\, \aleph \,|\, \neg\Phi \,|\, \Phi \vee \Phi$$

MoSL: State formulae

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

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

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

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

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

- 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 ∞).

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

Model Checking MoSL

- 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

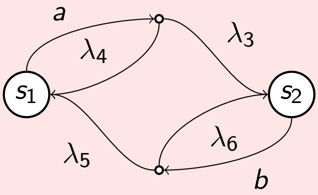
- Model-checking of RTSs is performed by using a CSL model checker.
- The proposed model-checking algorithm manipulates the input RTS obtained from a StoKlaim specification
  - the RTS to be model-checked is translated into an *equivalent* state-labelled CTMC
  - obtained CTMC is then analysed by making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

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.

An example...

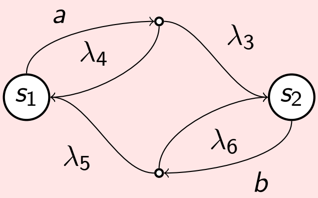
From RTS...



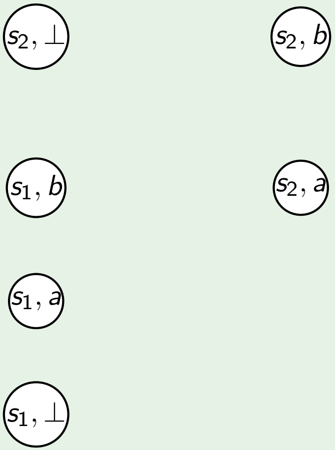
...to CTMC

An example...

From RTS...

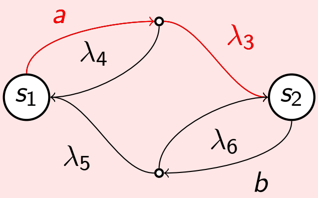


...to CTMC

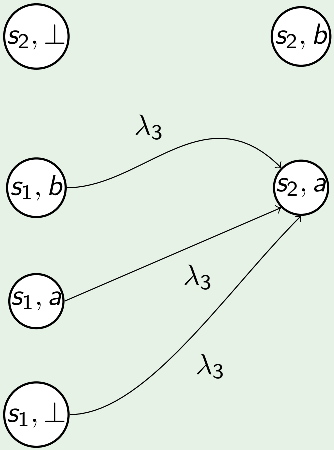


An example...

From RTS...

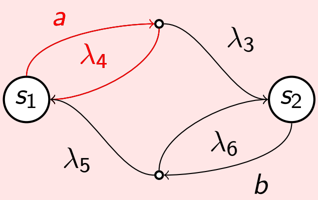


...to CTMC

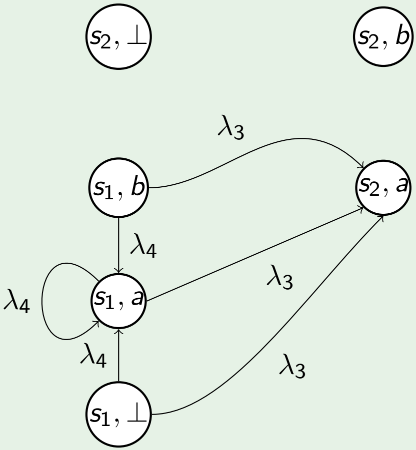


An example...

From RTS...

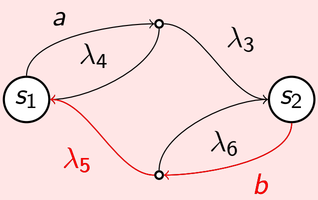


...to CTMC

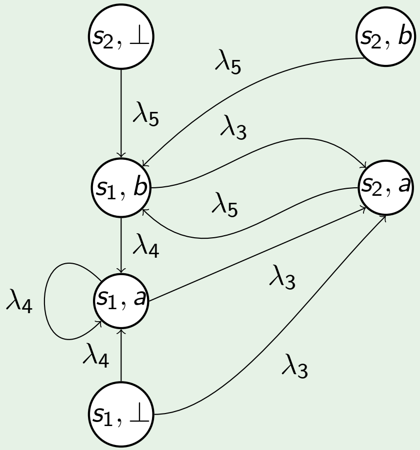


An example. . .

From RTS. . .

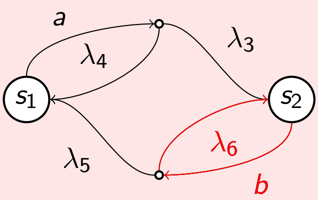


. . . to CTMC

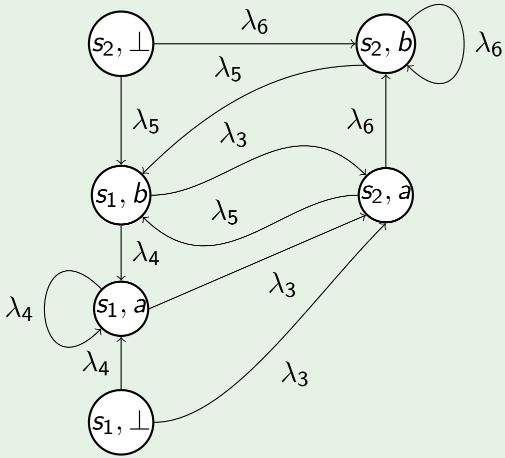


An example. . .

From RTS. . .



. . . to CTMC



An example: Leader Election

Distributed leader election:

We consider an algorithm for *distributed leader election*:

- it is assumed that the nodes are always arranged in a ring
- in StoKlaim the system consists of N nodes each of which hosts the execution of a process.

All The Way...

In this algorithm every participants is univocally identified by an *id*. The leader will be the node with the minimum *id*. We assume that nodes identifiers are selected randomly.

An example: Leader Election

An example: Leader Election

1. When a process has determined its *id*, an ELECTION message is sent to the next node in the ring.
   - This message contain node's *id* and a counter (set to zero at the beginning).
   - Election message travels *all the way* along the ring, forwarded by the other processes.

An example: Leader Election

1. When a process has determined its *id*, an ELECTION message is sent to the next node in the ring.
   - This message contain node's *id* and a counter (set to zero at the beginning).
   - Election message travels *all the way* along the ring, forwarded by the other processes.
2. Each time a process receives an ELECTION message, it store the smallest received *id* then forwards the message to the next node in the ring

An example: Leader Election

1. When a process has determined its *id*, an ELECTION message is sent to the next node in the ring.
   - This message contain node's *id* and a counter (set to zero at the beginning).
   - Election message travels *all the way* along the ring, forwarded by the other processes.
2. Each time a process receives an ELECTION message, it store the smallest received *id* then forwards the message to the next node in the ring
3. When a process receives back its ELECTION message, it knows the ring size.

An example: Leader Election

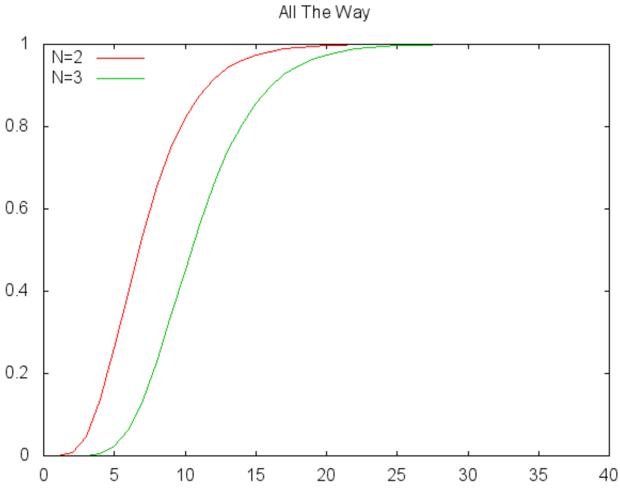
1. When a process has determined its *id*, an ELECTION message is sent to the next node in the ring.
    - This message contain node's *id* and a counter (set to zero at the beginning).
    - Election message travels *all the way* along the ring, forwarded by the other processes.
2. Each time a process receives an ELECTION message, it store the smallest received *id* then forwards the message to the next node in the ring
3. When a process receives back its ELECTION message, it knows the ring size.
4. The algorithm terminates when the number of messages received is equal to the ring size.

An example: Leader Election

$$\mathcal{P}_{=?}(\ {}_{true}\mathcal{U}^{\top}_{\leq t} \vee_i \langle \mathsf{ELECTED} \rangle @s_i \rightarrow \mathsf{tt})$$

An example: Leader Election

$$\mathcal{P}_{=?}(\ {}_{true}\mathcal{U}^{\top}_{\leq t}\ \vee_i \langle \mathsf{ELECTED} \rangle @ s_i \rightarrow \mathsf{tt})$$



An example: Leader Election

Numerical model-checking cannot be used when the considered specification leads to *large* CTMC

An example: Leader Election

Numerical model-checking cannot be used when the considered specification leads to *large* CTMC

All The Way:

Components	States	Transitions
2	116	180
3	6821	15129
4	952154	2770320

Statistical Model-Checking

Statistical Model-Checking

To overcome the state explosion problem, a *statistical model-checker* can be used.

- this approach has been successfully applied to existing model checkers (YMER, sCOWS,...)

Statistical Model-Checking

To overcome the state explosion problem, a *statistical model-checker* can be used.

- this approach has been successfully applied to existing model checkers (YMER, sCOWS,. . .)

In a numerical model-checker, the exact probability to satisfy a path-formula is computed up to a precision ϵ.

Statistical Model-Checking

To overcome the state explosion problem, a *statistical model-checker* can be used.

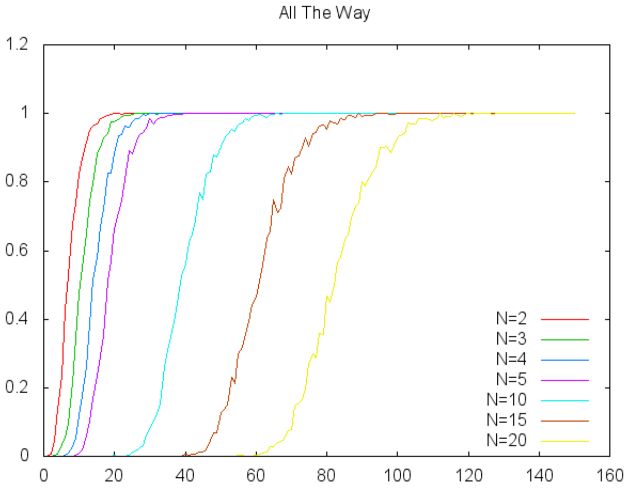
- this approach has been successfully applied to existing model checkers (YMER, sCOWS,...)

In a numerical model-checker, the exact probability to satisfy a path-formula is computed up to a precision ϵ.

A statistical model-checker is parametrised with respect to a given *tollerance* ε and *error probability* δ. The algorithm guarantees that the difference between the computed values and the exact ones are greater than ε with a probability that is less than δ.

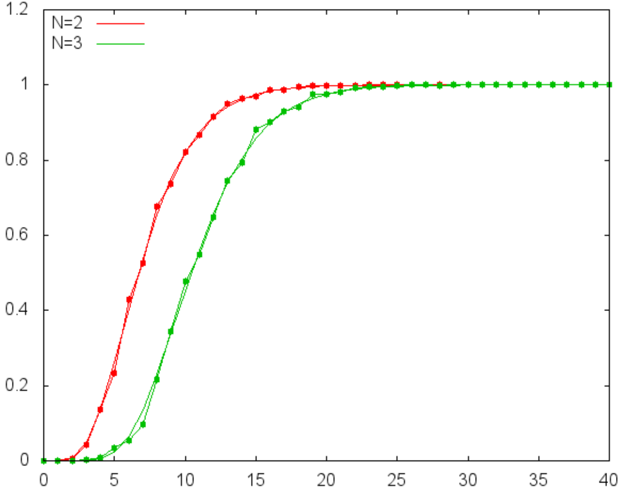
An example: Leader Election

Statistical Analysis



An example: Leader Election

Statistical vs Numerical

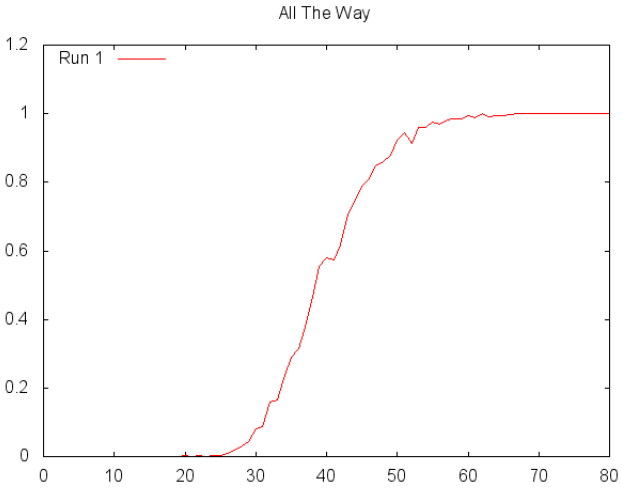


An example: Leader Election

Statistical Analysis

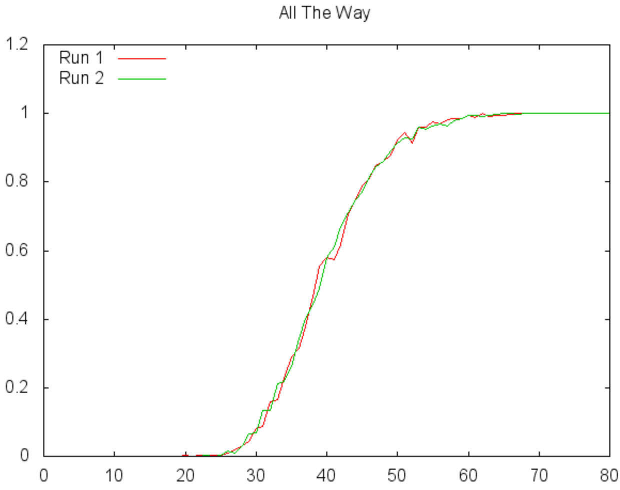
An example: Leader Election

Statistical Analysis ($N = 10$, $\varepsilon = 0.1$, $\delta = 0.1$)



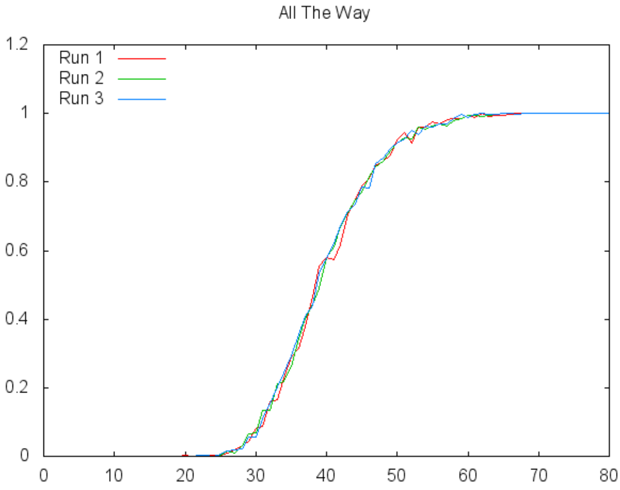
An example: Leader Election

Statistical Analysis ($N = 10$, $\varepsilon = 0.1$, $\delta = 0.1$)



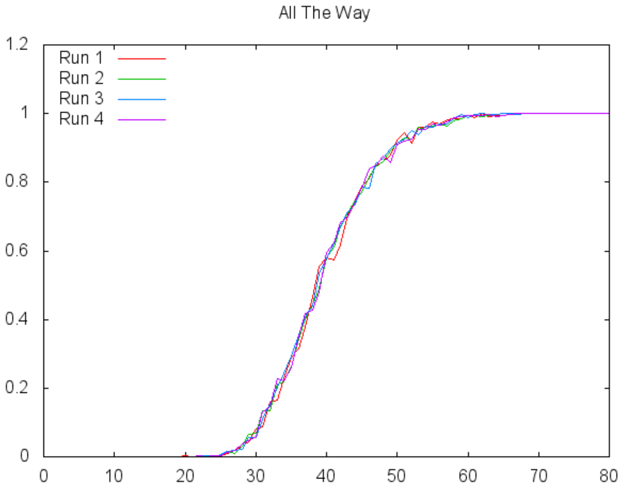
An example: Leader Election

Statistical Analysis ($N = 10$, $\varepsilon = 0.1$, $\delta = 0.1$)



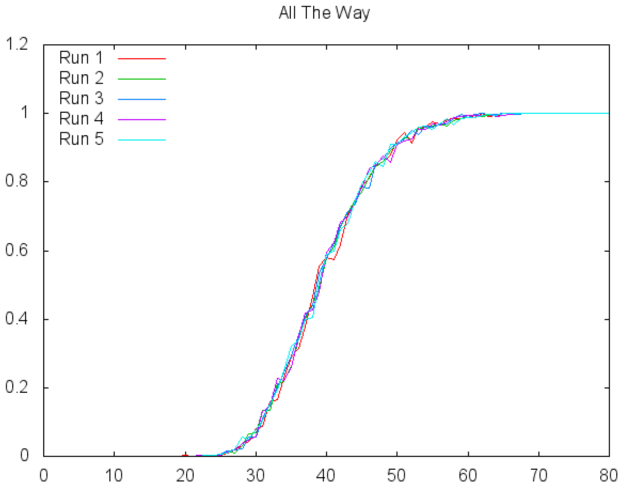
An example: Leader Election

Statistical Analysis ($N = 10$, $\varepsilon = 0.1$, $\delta = 0.1$)



An example: Leader Election

Statistical Analysis ($N = 10$, $\varepsilon = 0.1$, $\delta = 0.1$)



Concluding Remarks

Concluding Remarks

- StoKlaim and MoSL can ben used for specifying and verifying properties of mobile and distributed systems.
- A tool (SAM) has been developed for:
  - verifying whether a given system satisfies or not a given property
    - numerical model-checking (by relying on MRMC)
    - statistical model-checking
  - simulating system behaviour.

Concluding Remarks

- STOKLAIM and MOSL can ben used for specifying and verifying properties of mobile and distributed systems.
- A tool (SAM) has been developed for:
  - verifying whether a given system satisfies or not a given property
    - numerical model-checking (by relying on MRMC)
    - statistical model-checking
  - simulating 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

Only the braves...

SAM Home page:

http://rap.dsi.unifi.it/SAM/