Model Checking Multiagent Systems

Runtime Verification of Deontic and Trust Models in Multiagent Interactions

Nardine Z. Osman

Presented @ AIG, UPF

April 17, 2009
Motivating Example

- auction scenario
- auction scenario
- travel agency scenario
Motivating Example

- travel agency scenario
- auction scenario
- auction scenario
- auction scenario
- travel agency scenario

where should I go?
Motivating Example

We believe the problem of multiagent system verification should be viewed/addressed as the problem of having the agent select the suitable interaction group to join.

Problem Definition: a preliminary statement
The exact specification of the system (or processes) is provided. Formal verification is carried out over this precise specification.
How do MAS differ from traditional hardware/software systems?

- Unlike traditional distributed open systems, agents are:
  - autonomous
  - flexible
    - reactive
    - pro-active
    - social
Multiagent Systems (MAS)

How do MAS differ from traditional hardware/software systems?
- furthermore, the system specification is weak:
  - the system relies heavily on the agents involved
  - the agent specification is not part of the system specification
  - the system cannot enforce agents to abide to it
    - agents may be incapable of performing a given action
    - agents may be unwilling to perform a given action
Hence, we say the best verification results would be obtained if the system is viewed as a combination of both interaction and agent models.

But the agent specifications cannot and should not be made available. Hence, the agent model will resemble a set of constraints that has been shared/learned, and that is deemed necessary for the interaction.
The MAS verification problem definition

*Given an interaction model and a group of agents willing to participate in that interaction model, would this combination work?*
MAS Verification: problem (re-)definition

Multiagent systems are highly dynamic systems. Furthermore, in MAS, it is the agent that decides:

- how to interact
- who to interact with
- when to interact

Hence, it is the agent that needs to verify, at runtime, whether the interaction it wishes to join and the group of agents it is willing to interact with make a suitable combination.
MAS Verification: problem (re-)definition
MAS Verification: problem (re-)definition

- travel agency scenario
- auction scenario
- travel agency scenario

where should I go?

which interaction model should I engage in?

which agents should I interact with?
MAS Verification: problem (re-)definition

- auction scenario
- auction scenario
- travel agency scenario

where should I go?
which interaction model should I engage in?
which agents should I interact with?

does this combination of agents with this interaction work?

Model Checking Multiagent Systems
MAS Verification: problem (re-)definition

The MAS verification problem definition

Given an interaction model and a group of agents willing to participate in that interaction model, would this combination work?
The MAS verification problem definition - revisited

Can agents verify at interaction time, when the conditions for verification are met, that a given interaction group is suitable?
trust application
The issue of selecting the appropriate interaction model and the appropriate agents to interact with may be viewed as an issue of trust.
Trust Example: an auction system
Trust Example: an auction system

Model Checking Multiagent Systems
Trust Example: an auction system

Some trust issues:

1. Deadlock free interaction:
   The interaction is trusted if it is deadlock free.

2. Incentive compatible interaction:
   For a given interaction, the bidders are better off when they truthfully reveal their true valuation (i.e. truth telling bidders).

3. Experience based trust:
   Agent $\mathcal{A}$ is trusted to deliver good quality DVDs, only if I know it has done so in the past.

4. Reputation based trust:
   Agent $\mathcal{A}$ is trusted to take the role of the auctioneer only if it has numerous ratings and holds a good reputation, with more importance being given to the most recent ratings.

5. Subjective perception based trust:
   If agent $\mathcal{A}$ is trusted in delivering good quality CDs, then it is trusted to deliver DVDs as well.
Trust Example: an auction system

Some *trust* issues:

1. Deadlock free interaction
2. Incentive compatible interaction
3. Experience based trust
4. Reputation based trust
5. Subjective perception based trust
Trust Example: an auction system

Model Checking Multiagent Systems
Trust Example: an auction system

Model Checking Multiagent Systems
Trust in MAS: what we do & do not do

- Reputation Models
- Socio-Cognitive Models
- Evolutionary & Learning Models
- Trustworthy Interaction Mechanisms
- Reputation Mechanisms
- Distributed Security Mechanisms

Reasoning

Individual Level

System Level

Actions

Model Checking Multiagent Systems
Trust in MAS: what we do & do not do

We do not analyse, develop, & implement trust strategies.

We do specify & verify given strategies.

We do not analyse, develop, & implement trust strategies. We do specify & verify given strategies.

taken from [6]

Model Checking Multiagent Systems
Proposal

We choose **model checking** to provide agents with an automated formal verification mechanism.

Why model checking?
Simply, it provides full **automation** of the verification process.

---

**The model checking problem definition**

Given a finite transition system \( S \) and a temporal property \( \phi \), does \( S \models \phi \) hold?
Proposal: the model checker’s design

- System Model
  - Transition Graph

- Properties to Verify
  - Temporal Properties

Model Checking Algorithm

Result
Proposal: the model checker’s design

Interaction Model

Temporal Properties

Model Checking Algorithm

Result

System Model

Properties to Verify

Temporal Properties

Interaction Model
Proposal: the model checker’s design

- System Model
  - Interaction Model
- Properties to Verify
  - Trust Properties
  - Temporal Properties
- Model Checking Algorithm
- Result

- Introduction
- Motivation
- Challenges
  - Traditional Sys.
  - Multiagent Sys.
- Problem Definition
- Proposal
  - Trust Application
  - Trust Example
  - Trust in MAS
  - Proposed Verifier
- Specification
  - System Specs.
  - Property Specs.
  - Temporal Specs.
  - Trust Specs.
- Verification
  - MC Process
  - MC Components
  - Proof Rules
  - Transition Rules
  - Translator
  - Implementation
- Discussion
  - Results
  - Compare to others
  - Applications
- Conclusion

Model Checking Multiagent Systems
System & Property Specification

Model Checking Multiagent Systems

Introduction
Motivation
Challenges
Traditional Sys.
Multiagent Sys.
Problem Definition

Proposal
Trust Application
Trust Example
Trust in MAS
Proposed Verifier

Specification
System Specs.
Property Specs.
Temporal Specs.
Trust Specs.

Verification
MC Process
MC Components
Proof Rules
Transition Rules
Translator
Implementation

Discussion
Results
Compare to others
Applications

Conclusion
System & Property Specification

Model Checking Algorithm

- Interaction Model
  - process calculus: LCC

- Trust Properties
  - trust policy: TPL

- Temporal Properties
  - temporal logic: μ-calculus
Interaction Model Specification

A state transition system is needed to specify the system model fed to the model checker.

We choose a **process calculus** for specifying interactions: the Lightweight Coordination Calculus (LCC), a calculus focusing on agent interactions in MAS.

Process calculus:

- allows the formal modelling of concurrent systems
- focuses on the interactions, communications, and synchronizations between independent processes
- allows the formal reasoning about the system, such as analysing equivalences between processes
Interaction Model Specification

Examples of some process calculi notations/operators:

<table>
<thead>
<tr>
<th></th>
<th>LCC</th>
<th>CCS</th>
<th>CSP</th>
<th>$\pi$-Calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition / Recursion</strong></td>
<td>A::ADef</td>
<td>$P\overset{\text{def}}{=} \text{PDef}$</td>
<td>$P=\text{PDef}$</td>
<td>$P\overset{\text{def}}{=} \text{PDef}$</td>
</tr>
<tr>
<td><strong>Prefix</strong></td>
<td>A then B</td>
<td>a.$P$</td>
<td>$a\rightarrow P$</td>
<td>a.$P$</td>
</tr>
<tr>
<td><strong>Sequential</strong></td>
<td>A then B</td>
<td>$P;Q$</td>
<td>$P;Q$</td>
<td>$P;Q$</td>
</tr>
<tr>
<td><strong>Choice</strong></td>
<td>A or B</td>
<td>$P+Q$</td>
<td>$(a\rightarrow P \mid b\rightarrow Q)$</td>
<td>$P+Q$</td>
</tr>
<tr>
<td><strong>Concurrency / Parallelism</strong></td>
<td>A par B</td>
<td>$P</td>
<td>Q$</td>
<td>$P\parallel Q$</td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td>P\text{\textbackslash}J</td>
<td>P\text{\textbackslash}J</td>
<td>P\text{\textbackslash}J</td>
<td>(\forall x)P</td>
</tr>
<tr>
<td><strong>Concealment / Abstraction</strong></td>
<td>P\text{\textbackslash}C</td>
<td>P\text{\textbackslash}C</td>
<td>P\text{\textbackslash}C</td>
<td>P\text{\textbackslash}C</td>
</tr>
<tr>
<td><strong>Relabelling</strong></td>
<td>P[f]</td>
<td>L:P</td>
<td>L:P</td>
<td>L:P</td>
</tr>
<tr>
<td><strong>Replication</strong></td>
<td>!P</td>
<td>!P</td>
<td>!P</td>
<td>!P</td>
</tr>
<tr>
<td><strong>Input Action</strong></td>
<td>M$\leftarrow$A $[\leftarrow C]$</td>
<td>a($x$)</td>
<td>a?$x$</td>
<td>a($x$)</td>
</tr>
<tr>
<td><strong>Output Action</strong></td>
<td>M$\rightarrow$A $[\leftarrow C]$</td>
<td>$\bar{a}$($x$)</td>
<td>a!$x$</td>
<td>$\bar{a}$($x$)</td>
</tr>
<tr>
<td><strong>Empty Action / Empty Process</strong></td>
<td>null $[\leftarrow C]$</td>
<td>NIL</td>
<td>STOP</td>
<td>0</td>
</tr>
</tbody>
</table>
## Interaction Model Specification

Examples of some process calculi notations/operators:

<table>
<thead>
<tr>
<th></th>
<th>LCC</th>
<th>CCS</th>
<th>CSP</th>
<th>π-Calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition /</strong></td>
<td>A::ADef</td>
<td>P(^{\text{def}})PDef</td>
<td>P=PDef</td>
<td>P(^{\text{def}})PDef</td>
</tr>
<tr>
<td><strong>Recursion</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Prefix</strong></td>
<td>A then B</td>
<td>a.P</td>
<td>a(\rightarrow)P</td>
<td>a.P</td>
</tr>
<tr>
<td><strong>Sequential</strong></td>
<td>A then B</td>
<td>P;Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Choice</strong></td>
<td>A or B</td>
<td>P+Q</td>
<td>(a(\rightarrow)P (\mid) b(\rightarrow)Q)</td>
<td>P+Q</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>P(\square)Q</td>
<td>P(\square)Q</td>
</tr>
<tr>
<td><strong>Concurrency /</strong></td>
<td>A par B</td>
<td>P(\parallel)Q</td>
<td>P(\parallel)Q</td>
<td>P(\parallel)Q</td>
</tr>
<tr>
<td><strong>Parallelism</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td>P(\setminus)J</td>
<td></td>
<td>P(\setminus)C</td>
<td>((\forall x))P</td>
</tr>
<tr>
<td><strong>Concealment /</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Abstraction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Relabelling</strong></td>
<td>P([f])</td>
<td>L:P</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Replication</strong></td>
<td></td>
<td></td>
<td></td>
<td>!P</td>
</tr>
<tr>
<td><strong>Input Action</strong></td>
<td>M(\leftarrow)A ([\leftarrow\cdot\cdot])</td>
<td>a(x)</td>
<td>a?x</td>
<td>a(x)</td>
</tr>
<tr>
<td><strong>Output Action</strong></td>
<td>M(\rightarrow)A ([\leftarrow\cdot\cdot])</td>
<td>(\overline{a}(x))</td>
<td>a!x</td>
<td>(\overline{a}(x))</td>
</tr>
<tr>
<td><strong>Empty Action /</strong></td>
<td>null ([\leftarrow\cdot\cdot])</td>
<td>NIL</td>
<td>STOP</td>
<td>0</td>
</tr>
<tr>
<td><strong>Empty Process</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interaction Model Specification

Example: the auction system's interaction protocol in LCC

\[
\text{a(auctioneer}(I,R,Bs),A) ::
\]
\[
( \text{invite}(I,R) \Rightarrow \text{a(bidder},B) \leftarrow Bs=[B|T] \text{ then}
\]
\[
\text{a(auctioneer}(I,R,T),A) )
\]
\[
\text{or}
\]
\[
\text{a(auctioneer2}(Bs,[]),A) \leftarrow Bs=[].
\]

\[
\text{a(auctioneer2}(Bs,Vs),A) ::
\]
\[
\text{append}([B,V],Vs,Vn) \leftarrow \text{bid}(B,V) \leftarrow \text{a(bidder},B) \text{ then}
\]
\[
( \text{a(auctioneer2}(Bs,Vn),A) \leftarrow \text{not(all_bid}(Bs,Vn))
\]
\[
\text{or}
\]
\[
( \text{win}(B1,V2) \Rightarrow \text{a(bidder},B1) \leftarrow \text{all_bid}(Bs,Vn) \land
\]
\[
\text{highest}(Vn,B1,\_\_) \land \text{second_highest}(Vn,\_,V2) \text{ then}
\]
\[
\text{deliver}(I,B1) \leftarrow \text{payment}(P) \leftarrow \text{a(bidder},B1) ) ).
\]

\[
\text{a(bidder},B) ::
\]
\[
\text{invite}(I,R) \leftarrow \text{a(auctioneer}(\_,\_,\_),A) \text{ then}
\]
\[
\text{bid}(B,V) \Rightarrow \text{a(auctioneer2}(\_,\_),A) \leftarrow \text{valuation}(I,V) \text{ then}
\]
\[
\text{win}(Bi,Vi) \leftarrow \text{a(auctioneer2}(\_,\_),A) \text{ then}
\]
\[
( \text{payment}(P) \Rightarrow \text{a(auctioneer2}(\_,\_),A) \leftarrow Bi=B \land \text{payment}(P)
\]
\[
\text{or}
\]
\[
\text{null }.\]

Model Checking Multiagent Systems
Temporal Property Specification

Temporal logic:

- allows the formal representation of temporal information
- time, in a temporal logic, is viewed as a sequence of states, representing the flow of time
- allows the formal reasoning about temporal properties of propositions by studying which states these propositions are valid at
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \alpha$</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>$H \alpha$</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>$F \alpha$</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>$G \alpha$</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>$\alpha U \beta$</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>$\alpha W \beta$</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>$\alpha R \beta$</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>$N \alpha/X \alpha$</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>$A \alpha$</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>$E \alpha$</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>
## Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P (\alpha)</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H (\alpha)</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>F (\alpha)</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G (\alpha)</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>(\alpha U \beta)</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>(\alpha W \beta)</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>(\alpha R \beta)</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N (\alpha/X \alpha)</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A (\alpha)</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E (\alpha)</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>
### Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P α</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H α</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>F α</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G α</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>α U β</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>α W β</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>α R β</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N α/X α</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A α</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E α</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>

- **P**: Past
- **H**: -
- **F**: Future/Eventually
- **G**: Globally
- **U**: Until
- **W**: Weak Until
- **R**: Release
- **X**: Next
- **A**: All/Always
- **E**: Exists

- **Tense**: Tense, LTL, CTL
- **LTL**: LTL, CTL
- **CTL**: Constraint Temporal Logic
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P α</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H α</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>F α</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G α</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>α U β</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>α W β</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>α R β</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N α/X α</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A α</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E α</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P α</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H α</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>F α</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G α</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>α U β</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>α W β</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>α R β</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N α/X α</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A α</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E α</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>

Introduction
Motivation
Challenges
Traditional Sys.
Multiagent Sys.
Problem Definition
Proposal
Trust Application
Trust Example
Trust in MAS
Proposed Verifier
Specification
System Specs.
Property Specs.
Temporal Specs.
Verification
MC Process
MC Components
Proof Rules
Transition Rules
Translator
Implementation
Discussion
Results
Compare to others
Applications
Conclusion
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \alpha$</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>$H \alpha$</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>$F \alpha$</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>$G \alpha$</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>$\alpha U \beta$</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>$\alpha W \beta$</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>$\alpha R \beta$</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>$N \alpha/X \alpha$</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>$A \alpha$</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>$E \alpha$</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P α</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H α</td>
<td></td>
<td>Tense</td>
</tr>
<tr>
<td>F α</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G α</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>α U β</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>α W β</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>α R β</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N α/X α</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A α</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E α</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P α</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H α</td>
<td></td>
<td>Tense</td>
</tr>
<tr>
<td>F α</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G α</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>α U β</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>α W β</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>α R β</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N α/X α</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A α</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E α</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \alpha$</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>$H \alpha$</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>$F \alpha$</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>$G \alpha$</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>$\alpha U \beta$</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>$\alpha W \beta$</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>$\alpha R \beta$</td>
<td>Release</td>
<td>CTL</td>
</tr>
<tr>
<td>$N \alpha/X \alpha$</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>$A \alpha$</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>$E \alpha$</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P α</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H α</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>F α</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G α</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>α U β</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>α W β</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>α R β</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N α/X α</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A α</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E α</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>

Note that A and E are branching time operators.
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P α</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H α</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>F α</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G α</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>α U β</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>α W β</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>α R β</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N α/X α</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A α</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E α</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>

Note that A and E are branching time operators.
Temporal Property Specification

Examples of some temporal operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>P α</td>
<td>Past</td>
<td>Tense</td>
</tr>
<tr>
<td>H α</td>
<td>-</td>
<td>Tense</td>
</tr>
<tr>
<td>F α</td>
<td>Future/Eventually</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>G α</td>
<td>Globally</td>
<td>Tense, LTL, CTL</td>
</tr>
<tr>
<td>α U β</td>
<td>Until</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>α W β</td>
<td>Weak Until</td>
<td>CTL</td>
</tr>
<tr>
<td>α R β</td>
<td>Release</td>
<td>LTL</td>
</tr>
<tr>
<td>N α/X α</td>
<td>Next</td>
<td>LTL, CTL</td>
</tr>
<tr>
<td>A α</td>
<td>All/Always</td>
<td>CTL</td>
</tr>
<tr>
<td>E α</td>
<td>Exists</td>
<td>CTL</td>
</tr>
</tbody>
</table>

Note that A and E are branching time operators.

Example: A(α W β)
Temporal Property Specification

We choose the $\mu$-calculus, a different branching temporal logic. Why?

- Simple syntax: fewer operators, but denser formulae
  - Modal operators ($\Box$ and $\Diamond$) for branching properties
  - Fixed point operators ($\mu$ and $\nu$) for temporal properties
- Simple (logic-based local) model checking algorithm

### The $\mu$-calculus syntax: a modified version

$$\phi ::= \text{tt} | \text{ff} | Z | \phi_1 \land \phi_2 | \phi_1 \lor \phi_2 | \langle A \rangle \phi | [A] \phi | \nu Z. \phi | \mu Z. \phi$$

P.S. The difference between traditional $\mu$-calculus and this modified version is that the set of actions $A$ could be either communicative or **non-communicative**!
Trust Specification

Syntax of the trust policy language (TPL):

\[
\text{TrustRule} := \text{TrustSpecs} \leftarrow \text{Condition} \\
\text{TrustSpecs} := \text{trust}(\text{interaction}(\text{IP}), \text{Sign}) \mid \\
\quad \text{trust} (\text{Agent}, \text{Sign}) \mid \\
\quad \text{trust} (\text{Agent}, \text{Sign}, \text{Action}) \\
\text{Agent} := a(\text{Role}, \text{Id}) \\
\text{Sign} := + \mid - \\
\text{Action} := \text{MPA} \mid \text{N-MPA} \mid \text{TrustSpecs} \\
\text{MPA} := \text{Message} \Rightarrow \text{Agent} \mid \\
\quad \text{Message} \Leftarrow \text{Agent} \\
\text{Condition} := \text{Condition} \wedge \text{Condition} \mid \\
\quad \text{Condition} \vee \text{Condition} \mid \\
\quad \text{Temporal} \mid \\
\quad \text{Term} \\
\text{Role, N-MPA, Message} := \text{Term}
\]
Trust Specification

Examples:

1. \text{trust(interaction(IP),+)} \leftarrow \\
\nu Z. \langle -\rangle tt \land [-] Z.

2. \text{trust(a(auctioneer,A),+,deliver(cd(\_)))} \leftarrow \\
\text{know(a(auctioneer,A),deliver(cd(\_)),success)}.

4. \text{trust(a(auctioneer,A),+)} \leftarrow \\
\text{rating\_count(a(auctioneer,A),Total) \land Total>50 \land} \\
\text{rating\_average(a(auctioneer,A),Average) \land Average>0.7 \land} \\
\text{rating\_latest(a(auctioneer,A),0.2,Latest) \land Latest>0.95}.

5. \text{trust(a(auctioneer,A),+,deliver(dvd(\_)))} \leftarrow \\
\text{trust(a(auctioneer,A),+,deliver(cd(\_)))}.
Trust Specification

Examples:

\[
\text{trust(interaction(IP),+)} \leftarrow \\
V-- < B- \land B- < V- \land V- < V \land V < V+ \land V+ < B+ \land B+ < V++ \land \\
[bid(Bidder,V),bid(Competitor,B-)] \langle \text{win(Bidder,X)} \rangle tt \land \\
[bid(Bidder,V-),bid(Competitor,B-)] \langle \text{win(Bidder,Y)} \rangle tt \land \\
[bid(Bidder,V--),bid(Competitor,B-)] \langle \text{win(Competitor,Z)} \rangle tt \land \\
[bid(Bidder,V),bid(Competitor,B+)] \langle \text{win(Competitor,Z)} \rangle tt \land \\
[bid(Bidder,V+),bid(Competitor,B+)] \langle \text{win(Competitor,Z)} \rangle tt \land \\
[bid(Bidder,V++),bid(Competitor,B+)] \langle \text{win(Bidder,Z)} \rangle tt \land \\
X \geq Y \land X \geq Z .
\]

Case 1: with subcases (a), (b), and (c)
Case 2: with subcases (a), (b), and (c)
Trust Specification

Examples:

\[ \text{trust(interaction(IP),+)} \leftarrow \]
\[ V--<B- \land B-<V- \land V-<V \land V<V+ \land V+<B+ \land B+<V++ \land \]
\[ \text{[bid(Bidder,V),bid(Competitor,B-)]} \langle \text{win(Bidder,X)} \rangle \text{tt} \land \]
\[ \text{[bid(Bidder,V-),bid(Competitor,B-)]} \langle \text{win(Bidder,Y)} \rangle \text{tt} \land \]
\[ \text{[bid(Bidder,V--),bid(Competitor,B-)]} \langle \text{win(Competitor,_)tt} \land \]
\[ \text{[bid(Bidder,V),bid(Competitor,B+)]} \langle \text{win(Competitor,_)tt} \land \]
\[ \text{[bid(Bidder,V+),bid(Competitor,B+)]} \langle \text{win(Competitor,_)tt} \land \]
\[ \text{[bid(Bidder,V++),bid(Competitor,B+)]} \langle \text{win(Bidder,Z)} \rangle \text{tt} \land \]
\[ X\geq Y \land X\geq Z . \]

Case 1: with subcases (a), (b), and (c)

Case 2: with subcases (a), (b), and (c)
Trust Specification

Examples:

\[ \text{trust(interaction(IP),+)} \leftarrow \]
\[ V--<B- \land B--<V- \land V--<V \land V--<V+ \land V--<B+ \land B++<V++ \land \]
\[ \left\lbrack \text{bid(Bidder,V)}, \text{bid(Competitor,B-)} \right\rbrack \left\lbrack \text{win(Bidder,X)} \right\rbrack _{tt} \land \]
\[ \left\lbrack \text{bid(Bidder,V-)}, \text{bid(Competitor,B-)} \right\rbrack \left\lbrack \text{win(Bidder,Y)} \right\rbrack _{tt} \land \]
\[ \left\lbrack \text{bid(Bidder,V---)}, \text{bid(Competitor,B-)} \right\rbrack \left\lbrack \text{win(Competitor,0)} \right\rbrack _{tt} \land \]
\[ \left\lbrack \text{bid(Bidder,V)}, \text{bid(Competitor,B+)} \right\rbrack \left\lbrack \text{win(Competitor,0)} \right\rbrack _{tt} \land \]
\[ \left\lbrack \text{bid(Bidder,V+)}, \text{bid(Competitor,B+)} \right\rbrack \left\lbrack \text{win(Bidder,Z)} \right\rbrack _{tt} \land \]
\[ X+Y \land X+Z. \]

Case 1: with subcases (a), (b), and (c)

Case 2: with subcases (a), (b), and (c)
Verifiable

Interaction Model

\[
a(\text{auctioneer}(I,R,Bs),A) ::
\)

\[
\begin{align*}
&\text{invited}(I,R) \Rightarrow a(\text{bidder},B) \leftarrow Bs=[B|T] \text{ then } \\
&a(\text{auctioneer}(I,R,T),A) \\
\text{or} \\
&a(\text{auctioneer2}(Bs,[]),A) \leftarrow Bs=[].
\end{align*}
\]

\[
\begin{align*}
\text{trust}(a(\text{auctioneer},A),+,\text{deliver}(\text{cd}(\text{.}))) &\leftarrow \\
\text{know}(a(\text{auctioneer},A),\text{deliver}(\text{cd}(\text{.})),\text{success})
\end{align*}
\]

Trust Properties

\[
\begin{align*}
\text{trust}(\text{interaction}(\text{IP}),+) &\leftarrow \\
V--<B- & B--<V- & V--<V & V<+ & V<+ & B+<V++ & \\
\text{bid}(\text{Bidder},V),\text{bid}(\text{Competitor},B-)\{\text{win}(\text{Bidder},X)\} tt & \\
\text{bid}(\text{Bidder},V-),\text{bid}(\text{Competitor},B-)\{\text{win}(\text{Bidder},Y)\} tt & \\
\text{bid}(\text{Bidder},V--),\text{bid}(\text{Competitor},B-)\{\text{win}(\text{Competitor,}_)\} tt & \\
\text{bid}(\text{Bidder},V),\text{bid}(\text{Competitor},B+)\{\text{win}(\text{Competitor,}_)\} tt & \\
\text{bid}(\text{Bidder},V+),\text{bid}(\text{Competitor},B+)\{\text{win}(\text{Competitor,}_)\} tt & \\
\text{bid}(\text{Bidder},V++),\text{bid}(\text{Competitor},B+)\{\text{win}(\text{Bidder},Z)\} tt & \\
X\leq Y & X\leq Z.
\end{align*}
\]

Temporal Properties

\[
\begin{align*}
\text{trust}(\text{interaction}(\text{IP}),+) &\leftarrow \\
V++<B- & B++<V- & V++<V & V++<+ & B++<V++ & \\
\text{bid}(\text{Bidder},V),\text{bid}(\text{Competitor},B-)\{\text{win}(\text{Bidder},X)\} tt & \\
\text{bid}(\text{Bidder},V-),\text{bid}(\text{Competitor},B-)\{\text{win}(\text{Bidder},Y)\} tt & \\
\text{bid}(\text{Bidder},V--),\text{bid}(\text{Competitor},B-)\{\text{win}(\text{Competitor,}_)\} tt & \\
\text{bid}(\text{Bidder},V+),\text{bid}(\text{Competitor},B+)\{\text{win}(\text{Competitor,}_)\} tt & \\
\text{bid}(\text{Bidder},V++),\text{bid}(\text{Competitor},B+)\{\text{win}(\text{Bidder},Z)\} tt & \\
X\leq Y & X\leq Z.
\end{align*}
\]

Result

true / false
Verification

Interaction Model

\[
a(\text{auctioneer}(I,R,Bs),A) ::
\begin{align*}
& ( \text{invite}(I,R) \Rightarrow a(\text{bidder},B) \leftarrow Bs=[B|T] \text{ then} \\
& a(\text{auctioneer}(I,R,T),A) ) \text{ or} \\
& a(\text{auctioneer2}(Bs,[]),A) \leftarrow Bs=[].
\end{align*}
\]

Trust Properties

\[
\text{trust}(a(\text{auctioneer},A),+,\text{deliver}(cd(_))) \leftarrow \\
\text{know}(a(\text{auctioneer},A),\text{deliver}(cd(_)),\text{success}).
\]

Temporal Properties

\[
\text{trust}(\text{interaction}(IP),+) \leftarrow \\
V<-B- \land B<-V- \land V<-V \land V< V+ \land V+<B+ \land B< V++ \land \\
[\text{bid}(\text{Bidder},V),\text{bid}(\text{Competitor},B-)]\langle\text{win}(\text{Bidder},X)\rangle tt \land \\
[\text{bid}(\text{Bidder},V-),\text{bid}(\text{Competitor},B-)]\langle\text{win}(\text{Bidder},Y)\rangle tt \land \\
[\text{bid}(\text{Bidder},V--),\text{bid}(\text{Competitor},B-)]\langle\text{win}(\text{Competitor},\_))tt \land \\
[\text{bid}(\text{Bidder},V),\text{bid}(\text{Competitor},B+)]\langle\text{win}(\text{Competitor},\_))tt \land \\
[\text{bid}(\text{Bidder},V+),\text{bid}(\text{Competitor},B+)]\langle\text{win}(\text{Competitor},\_))tt \land \\
[\text{bid}(\text{Bidder},V++),\text{bid}(\text{Competitor},B+)]\langle\text{win}(\text{Bidder},Z)\rangle tt \land \\
X\leq Y \land X\leq Z.
\]

Result

true / false
Verification Process

- State-space:
  - generated and traversed at runtime, one step at a time (this is local model checking)
Verification Process

- State-space:
  - generated and traversed at runtime, one step at a time (this is local model checking)
- At each step, satisfaction is verified:
  - if a result is achieved, then terminate
  - else, make a transition to next state(s) and repeat
Verification Process

- State-space:
  - generated and traversed at runtime, one step at a time (this is local model checking)
- At each step, satisfaction is verified:
  - if a result is achieved, then terminate
  - else, make a transition to next state(s) and repeat
MC Component 1: proof rules

The $\mu$-calculus proof rules

\[ \text{satisfies}(E, \text{tt}) \leftarrow \text{true} \]
\[ \text{satisfies}(E, \phi_1 \lor \phi_2) \leftarrow \text{satisfies}(E, \phi_1) \lor \text{satisfies}(E, \phi_2) \]
\[ \text{satisfies}(E, \phi_1 \land \phi_2) \leftarrow \text{satisfies}(E, \phi_1) \land \text{satisfies}(E, \phi_2) \]
\[ \text{satisfies}(E, \langle A \rangle \phi) \leftarrow \exists F.((E \xrightarrow{A} F) \land \text{satisfies}(F, \phi)) \]
\[ \text{satisfies}(E, [A] \phi) \leftarrow \forall F.((E \xrightarrow{A} F) \rightarrow \text{satisfies}(F, \phi)) \]
\[ \text{satisfies}(E, \mu Z. \phi) \leftarrow \text{satisfies}(E, \phi) \]
\[ \text{satisfies}(E, \nu Z. \phi) \leftarrow \text{dual}(\phi, \phi') \land \neg \text{satisfies}(E, \phi') \]
MC Component 2: transition rules

The LCC transition rules

\[
\begin{align*}
M \leftarrow A & \xrightarrow{\text{in}(M)} \text{nil} \\
M \Rightarrow A & \xrightarrow{\text{out}(M)} \text{nil} \\
null & \xrightarrow{\#} \text{nil} \\
\text{sat}(C) \land X \text{ in } C & \xrightarrow{(A \leftarrow C) \#(X)} A \\
A & \xrightarrow{a} E \\
\text{sat}(C) \land (a \neq \#/) & \xrightarrow{(A \leftarrow C) a} E \\
B & \xrightarrow{a} E \\
A \xrightarrow{a} E & \xrightarrow{:: B} A \\
A \xrightarrow{a} E & \xrightarrow{A \text{ or } B} E \\
B & \xrightarrow{a} E \\
A \xrightarrow{a} E & \xrightarrow{A \text{ or } B} E \\
A \xrightarrow{a} E & \xrightarrow{A \text{ par } B} E \text{ par } B \\
B & \xrightarrow{a} E & \xrightarrow{A \text{ par } B} A \text{ par } E \\
A \xrightarrow{a} E & \xrightarrow{A \text{ par } B} E \text{ par } F \\
B & \xrightarrow{a} E & \xrightarrow{A \text{ par } B} A \text{ par } E \\
A \xrightarrow{a} E & \xrightarrow{A \text{ par } B} E \text{ par } F \\
A \xrightarrow{a} \text{ nil} & \xrightarrow{A \text{ then } B} B \\
A \xrightarrow{a} E & \xrightarrow{A \text{ then } B} E \text{ then } B \\
A \xrightarrow{a} E & \xrightarrow{E \neq \text{ nil}} E\neq\text{ nil}
\end{align*}
\]
MC Component 1: proof rules

The $\mu$-calculus proof rules

\[
\text{satisfies}(E, tt) \quad \leftarrow \quad \text{true}
\]
\[
\text{satisfies}(E, \phi_1 \lor \phi_2) \quad \leftarrow \quad \text{satisfies}(E, \phi_1) \lor \text{satisfies}(E, \phi_2)
\]
\[
\text{satisfies}(E, \phi_1 \land \phi_2) \quad \leftarrow \quad \text{satisfies}(E, \phi_1) \land \text{satisfies}(E, \phi_2)
\]
\[
\text{satisfies}(E, \langle A \rangle \phi) \quad \leftarrow \quad \exists F.((E \xrightarrow{A} F) \land \text{satisfies}(F, \phi))
\]
\[
\text{satisfies}(E, [A] \phi) \quad \leftarrow \quad \forall F.((E \xrightarrow{A} F) \rightarrow \text{satisfies}(F, \phi))
\]
\[
\text{satisfies}(E, \mu Z. \phi) \quad \leftarrow \quad \text{satisfies}(E, \phi)
\]
\[
\text{satisfies}(E, \nu Z. \phi) \quad \leftarrow \quad \text{dual}(\phi, \phi') \land \neg \text{satisfies}(E, \phi')
\]
MC Component 1: proof rules

The $\mu$-calculus proof rules

- \( \text{satisfies}(E, tt) \leftarrow \text{true} \)
- \( \text{satisfies}(E, \phi_1 \lor \phi_2) \leftarrow \text{satisfies}(E, \phi_1) \lor \text{satisfies}(E, \phi_2) \)
- \( \text{satisfies}(E, \phi_1 \land \phi_2) \leftarrow \text{satisfies}(E, \phi_1) \land \text{satisfies}(E, \phi_2) \)
- \( \text{satisfies}(E, \langle A \rangle \phi) \leftarrow \exists F.((E A \rightarrow F) \land \text{satisfies}(F, \phi)) \)
- \( \text{satisfies}(E, [A] \phi) \leftarrow \forall F.((E A \rightarrow F) \rightarrow \text{satisfies}(F, \phi)) \)
- \( \text{satisfies}(E, \mu Z. \phi) \leftarrow \text{satisfies}(E, \phi) \)
- \( \text{satisfies}(E, \nu Z. \phi) \leftarrow \text{dual}(\phi, \phi') \land \neg \text{satisfies}(E, \phi') \)

P.S. The properties to be verified need to be specified in the $\mu$-calculus.
\[ \Rightarrow \] A translator should rewrite the TPL properties into the $\mu$-calculus.
MC Component 3: properties translator

The TPL to $\mu$-calculus translation rules

- $\text{trust(interaction(IM),+)} \leftarrow C \quad \rightarrow \quad \text{satisfied}(C)$
- $\text{trust(interaction(IM),-)} \leftarrow C \quad \rightarrow \quad \neg \text{satisfied}(C)$
- $\text{trust(a(Role,Id),+)} \leftarrow C \quad \rightarrow \quad \text{satisfied}(C) \lor \neg \text{satisfied}(C) \land \nu Z. \left[ \text{in}(a\text{(Role,Id),} \_), \text{out}(a\text{(Role,Id),} \_), \#(a\text{(Role,Id),} \_)] \text{ff} \land [\neg]Z$}
- $\text{trust(a(Role,Id),-)} \leftarrow C \quad \rightarrow \quad \neg \text{satisfied}(C) \lor (\text{satisfied}(C) \land \nu Z. \left[ \text{in}(a\text{(Role,Id),} \_), \text{out}(a\text{(Role,Id),} \_), \#(a\text{(Role,Id),} \_)] \text{ff} \land [\neg]Z)$
- $\text{trust(a(Role,Id),+,A)} \leftarrow C \quad \rightarrow \quad \text{satisfied}(C) \lor (\neg \text{satisfied}(C) \land \nu Z. [A] \text{ff} \land [\neg]Z)$
- $\text{trust(a(Role,Id),-,A)} \leftarrow C \quad \rightarrow \quad \neg \text{satisfied}(C) \lor (\text{satisfied}(\text{Condition}) \land \nu Z. [A] \text{ff} \land [\neg]Z)$
Implementation Choices

- State-space:
  - generated and traversed at runtime, one step at a time (this is local model checking)
- At each step, satisfaction is verified:
  - if a result is achieved, then terminate
  - else, make a transition to next state(s) and repeat

![Diagram](image-url)
Implementation Choices

- State-space:
  - generated and traversed at runtime, one step at a time (this is local model checking)
- At each step, **satisfaction** is verified:
  - if a result is achieved, then terminate
  - else, make a **transition** to next state(s) and repeat
- Implemented via XSB tabled Prolog:
  - inspired from XMC
  - caches results in tables
  - result:
    - ensures a solution is reached
    - terminates
    - avoids redundant computations
Results

The interaction protocol verified is the Vickrey auction.

The properties verified are 1 2 3 4 5.

The scene is set to incorporate 3 agents:

✦ 1 auctioneer
✦ 2 bidders
Results

The interaction protocol verified is the Vickrey auction.

The properties verified are 1 2 3 4 5.

The scene is set to incorporate 3 agents:

- 1 auctioneer
- 2 bidders

Results:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU time (sec)</td>
<td>0.017</td>
<td>0.204</td>
<td>0.020</td>
<td>0.010</td>
<td>0.021</td>
</tr>
<tr>
<td>Memory usage (MB)</td>
<td>1.327</td>
<td>14.052</td>
<td>1.356</td>
<td>0.917</td>
<td>1.376</td>
</tr>
</tbody>
</table>
Comparing with other Existing Mechanisms

MAS verification (& specification) mechanisms fall into three categories:

1. Verification of agent models
2. Verification of interaction models
3. Verification of interaction & agent models
Comparing with other Existing Mechanisms

MAS verification (& specification) mechanisms fall into three categories:

1. **Verification of agent models**
   - Can verify rich type of properties (beliefs,...)
   - Agent model might not be too specific, hence the behaviour of the system has massive/incalculable # of possibilities
   - Assumes agent model is accessible, but this cannot & should not be the case

2. **Verification of interaction models**
   - Model checking interaction models

3. **Verification of interaction & agent models**
   - Interaction Model

Model Checking Multiagent Systems
Comparing with other Existing Mechanisms

MAS verification (& specification) mechanisms fall into three categories:

1. **Verification of agent models**
   - Can verify rich type of properties (beliefs,...)
   - Agent model might not be too specific, hence the behaviour of the system has massive/incalculable # of possibilities
   - Assumes agent model is accessible, but this cannot & should not be the case

2. **Verification of interaction models**
   - Results in an efficient verification mechanism, since the behaviour of the system is more restricted
   - Can verify less interesting properties, i.e. properties of the interaction only, independent of the agents
   - Some work verifies interesting agent-dependent properties by considering the interaction model only; however, these mechanisms are not fully automated & they require new specification languages for every new concept to be verified

3. **Verification of interaction & agent models**
   - Interaction Model
   - Can verify rich type of properties (beliefs,...)
   - Agent model might not be too specific, hence the behaviour of the system has massive/incalculable # of possibilities
   - Assumes agent model is accessible, but this cannot & should not be the case
Comparing with other Existing Mechanisms

MAS verification (& specification) mechanisms fall into three categories:

1. **Verification of agent models**
   - Can verify rich type of properties (beliefs, ...)
   - Agent model might not be too specific, hence the behaviour of the system has massive/incalculable # of possibilities
   - Assumes agent model is accessible, but this cannot & should not be the case

2. **Verification of interaction models**
   - Results in an efficient verification mechanism, since the behaviour of the system is more restricted
   - Can verify less interesting properties, i.e. properties of the interaction only, independent of the agents
   - Some work verifies interesting agent-dependent properties by considering the interaction model only; however, these mechanisms are not fully automated & they require new specification languages for every new concept to be verified

3. **Verification of interaction & agent models**
   - Our proposed work lies in this category
   - It offers highly efficient automated verification of agent-dependent properties
   - It does not require to change the specification languages everytime a new concept needs to be verified

---

Introduction
Motivation
Challenges
Traditional Sys.
Multiagent Sys.
Problem Definition
Proposal
Trust Application
Trust Example
Trust in MAS
Problem Definition
Specification
System Specs.
Property Specs.
Temporal Specs.
Trust Specs.
Verification
MC Process
MC Components
Proof Rules
Transition Rules
Translator
Implementation
Discussion
Results
Compare to others
Applications
Conclusion
Applications

System Model

Interaction Model

Trust Properties

Temporal Properties

Model Checking Algorithm

Result

Properties to Verify

Properties to Verify

Interaction Model

Trust Properties

Temporal Properties

Model Checking Algorithm

Result

Introduction
Motivation
Challenges
Traditional Sys.
Multiagent Sys.
Problem Definition
Proposal
Trust Application
Trust Example
Trust in MAS
Proposed Verifier
Specification
System Specs.
Property Specs.
Temporal Specs.
Trust Specs.
Verification
MC Process
MC Components
Proof Rules
Transition Rules
Translator
Implementation
Discussion
Results
Compare to others
Applications
Conclusion
Applications

Model Checking Multiagent Systems

System Model

Interaction Model

Properties to Verify

Deontic Properties

Trust Properties

Temporal Properties

Model Checking Algorithm

Result

Introduction
Motivation
Challenges
Traditional Sys.
Multiagent Sys.
Problem Definition

Proposal
Trust Application
Trust Example
Trust in MAS
Proposed Verifier

Specification
System Specs.
Property Specs.
Temporal Specs.
Trust Specs.

Verification
MC Process
MC Components
Proof Rules
Transition Rules
Translator
Implementation

Discussion
Results
Compare to others
Applications

Conclusion
Applications: specifying deontic properties

Syntax of the deontic policy language (DPL):

DeonticRule := must(Agent, Action, Sign) [← Condition] |
              can(Agent, Action, Sign) [← Condition]

Agent := a(Role, Id)
Sign := + | −
Action := MPA | N-MPA
MPA := Message ⇒ Agent |
       Message ⇐ Agent

Condition := Condition ∧ Condition |
             Condition ∨ Condition |
Temporal | Term
Role, N-MPA, Message := Term
Applications: verifying deontic properties

<table>
<thead>
<tr>
<th>The DPL to $\mu$-calculus translation rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>can(Agent,Action,+)$\leftarrow C$  $\rightarrow$ ( satisfied(C) $\land$ (\mu X.\langle \text{Action}\rangle \text{tt} \neg \langle -\rangle X ) ) $\lor$ ( $\neg$ satisfied(C) $\land$ (\nu X.\langle -\text{Action}\rangle X \lor \langle -\rangle \text{ff} ) ) $\lor$ ( $\neg$ satisfied(C) $\land$ (\mu X.\langle -\text{Action}\rangle X \lor \langle -\rangle \text{tt} ) )</td>
</tr>
<tr>
<td>can(Agent,Action,-)$\leftarrow C$  $\rightarrow$ ( satisfied(C) $\land$ (\nu X.\langle -\text{Action}\rangle X \lor \langle -\rangle \text{ff} ) ) $\lor$ ( $\neg$ satisfied(C) $\land$ (\mu X.\langle \text{Action}\rangle \text{tt} \lor \langle -\rangle X ) )</td>
</tr>
<tr>
<td>must(Agent,Action,+)$\leftarrow C$  $\rightarrow$ ( satisfied(C) $\land$ (\mu X.\langle -\text{Action}\rangle X \lor \langle -\rangle \text{tt} ) ) $\lor$ ( $\neg$ satisfied(C) $\land$ (\nu X.\langle -\text{Action}\rangle X \lor \langle -\rangle \text{ff} ) )</td>
</tr>
<tr>
<td>must(Agent,Action,-)$\leftarrow C$  $\rightarrow$ ( satisfied(C) $\land$ (\nu X.\langle \text{Action}\rangle \text{ff} \lor \langle -\rangle X ) ) $\lor$ ( $\neg$ satisfied(C) $\land$ (\mu X.\langle \text{Action}\rangle \text{tt} \lor \langle -\rangle X ) )</td>
</tr>
</tbody>
</table>
Applications (future work)
Applications (future work)
Conclusion

We’ve presented a mechanism that:

❖ verifies a combination of agent & interaction models
  ◆ yet, it does not require the complete agent specification
  ◆ this provides efficient verification of a variety of properties

❖ introduces interaction time verification
  ◆ allows agents to verify at runtime the interaction groups they
    wish to join, which is crucial for highly dynamic distributed
    open systems
  ◆ using a lightweight, efficient, & fully automated verifier

❖ uses traditional model checking & specification languages
  ◆ verification of new concepts require extensions that lies on top
    of the model checker (as opposed to modifying existing
    specification languages and verification mechanisms)
Conclusion

We’ve presented a mechanism that:

✦ verifies a combination of agent & interaction models
  ✦ yet, it does not require the complete agent specification
  ✦ this provides efficient verification of a variety of properties

✦ introduces interaction time verification
  ✦ allows agents to verify at runtime the interaction groups they wish to join, which is crucial for highly dynamic distributed open systems
  ✦ using a lightweight, efficient, & fully automated verifier

✦ uses traditional model checking & specification languages
  ✦ verification of new concepts require extensions that lies on top of the model checker (as opposed to modifying existing specification languages and verification mechanisms)
Conclusion

We’ve presented a mechanism that:

✦ verifies a combination of agent & interaction models
  ✦ yet, it does not require the complete agent specification
  ✦ this provides efficient verification of a variety of properties

✦ introduces interaction time verification
  ✦ allows agents to verify at runtime the interaction groups they
    wish to join, which is crucial for highly dynamic distributed
    open systems
  ✦ using a **lightweight, efficient, & fully automated** verifier

✦ uses traditional model checking & specification languages
  ✦ verification of new concepts require extensions that lies on top
    of the model checker (as opposed to modifying existing
    specification languages and verification mechanisms)
Conclusion

We’ve presented a mechanism that:

♦ verifies a combination of agent & interaction models
  ♦ yet, it does not require the complete agent specification
  ♦ this provides efficient verification of a variety of properties

♦ introduces interaction time verification
  ♦ allows agents to verify at runtime the interaction groups they wish to join, which is crucial for highly dynamic distributed open systems
  ♦ using a lightweight, efficient, & fully automated verifier

♦ uses traditional model checking & specification languages
  ♦ verification of new concepts require extensions that lies on top of the model checker (as opposed to modifying existing specification languages and verification mechanisms)
Conclusion

Future work:

✦ apply the proposed model checker to different fields
  ✦ with some extensions, the model checker can prove properties about belief, knowledge, strategies, etc.
  ✦ the model checker can contribute to the field of collaboration and coalition formation is clear
  ✦ the model checker can aid the agent’s decision process, which could be very helpful for dialogue games, negotiation protocols, argumentation, etc.

✦ address the issue of **automated property specification**
  ✦ can an agent learn which properties are important in a given situation?
  ✦ can an agent specify such properties in an automated manner?
  ✦ can an agent automatically verify the minimum number of agents required to verify the automatically specified property?
Conclusion

Future work:

- apply the proposed model checker to different fields
  - with some extensions, the model checker can prove properties about belief, knowledge, strategies, etc.
  - the model checker can contribute to the field of collaboration and coalition formation is clear
  - the model checker can aid the agent’s decision process, which could be very helpful for dialogue games, negotiation protocols, argumentation, etc.

- address the issue of **automated property specification**
  - can an agent learn which properties are important in a given situation?
  - can an agent specify such properties in an automated manner?
  - can an agent automatically verify the minimum number of agents required to verify the automatically specified property?
Conclusion

Future work:

♦ apply the proposed model checker to different fields
   ✷ with some extensions, the model checker can prove properties about belief, knowledge, strategies, etc.
   ✷ the model checker can contribute to the field of collaboration and coalition formation is clear
   ✷ the model checker can aid the agent’s decision process, which could be very helpful for dialogue games, negotiation protocols, argumentation, etc.

♦ address the issue of **automated property specification**
   ✷ can an agent learn which properties are important in a given situation?
   ✷ can an agent specify such properties in an automated manner?
   ✷ can an agent automatically verify the minimum number of agents required to verify the automatically specified property?
References

Runtime verification of deontic and trust models in multiagent interactions, PhD thesis, School of Informatics, University of Edinburgh, March 2008.


