Docent Lecture:
Compositional Verification of Interaction Behaviour

Dilian Gurov

Theoretical Computer Science Department
KTH Royal Institute of Technology

16 March 2007
Interaction Behaviour
Interaction Behaviour

- **Computation**: Data Transformation + *Interaction*
Interaction Behaviour

- **Computation**: Data Transformation + *Interaction*
- **Focus on**: *on-going* interaction behaviour
Interaction Behaviour

- **Computation**: Data Transformation + *Interaction*
- **Focus on**: *on-going* interaction behaviour
- **Examples**:
  - teller machine (bankomat)
  - server accepting requests and sending responses
  - applications on a mobile device interacting via method calls
Interaction Behaviour

- **Computation**: Data Transformation + Interaction
- **Focus on**: on-going interaction behaviour
- **Examples**:
  - teller machine (bankomat)
  - server accepting requests and sending responses
  - applications on a mobile device interacting via method calls
- **Problem**:
  - how can we reason formally about interaction behaviour?
Dynamically Changing Architecture
## Dynamically Changing Architecture

- **Dynamic systems:**
  - components are generated dynamically
  - *open* systems: components dynamically join and leave system
Dynamically Changing Architecture

- **Dynamic systems:**
  - components are generated dynamically
  - *open* systems: components dynamically join and leave system

- **Examples:**
  - concurrent server spawns off component to handle request
  - application is loaded on a mobile device post-issuance
Dynamically Changing Architecture

- **Dynamic systems:**
  - components are generated dynamically
  - *open* systems: components dynamically join and leave system

- **Examples:**
  - concurrent server spawns off component to handle request
  - application is loaded on a mobile device post-issuance

- **Problem:**
  - how can we reason formally about the interaction behaviour of such systems?
  - *compositional* reasoning needed!
Concurrent Server
Concurrent Server

Server gets request

request

Server
Concurrent Server

Server gets request

request

Server

Server spawns off Handler

Handler

response
Concurrent Server

Property of Interaction Behaviour

Concurrent server always stabilizes (STAB)
Compositional Reasoning
Compositional Reasoning

Pure approach: Composing properties

S

- X
- Y
- Z
Compositional Reasoning

**Pure approach:**
Composing properties

**More general approach:**
Cut on component
Compositional Reasoning

**Pure approach:**
Composing properties

**More general approach:**
Cut on component

---

Concurrent Server

How does compositional reasoning help?
Proving Stabilization of Concurrent Server
Proving Stabilization of Concurrent Server

Original goal:

Server : STAB
Proving Stabilization of Concurrent Server

Original goal:

Server : STAB

Reduces to:

X : STAB
Handler

: STAB
Overview

1. Framework for Formal Reasoning
2. Interaction Behaviour
3. Behavioural Properties
   - Specification
   - Verification
4. Compositional Verification
   - Proof Systems
   - Maximal Models
5. Conclusion
Framework for Formal Reasoning: Ingredients
Framework for Formal Reasoning: Ingredients

Semantic Domains for Interaction Behaviour

- function from initial to final states: not suitable
- rather: sequences, or even trees, of interactions
Semantic Domains for Interaction Behaviour

- function from initial to final states: not suitable
- rather: sequences, or even trees, of interactions

Defining Interaction Behaviour

- semantic domain too low-level and unstructured
- composing behaviours
- meaning of behavioural definition given in semantic domain
Framework for Formal Reasoning: Ingredients

Specification and Verification

- *specification* captures desired behaviour
- *verification* establishes whether model/implementation meets specification
Framework for Formal Reasoning: Ingredients

**Specification and Verification**

- *specification* captures desired behaviour
- *verification* establishes whether model/implementation meets specification

**Compositional Verification**

- inferring system properties from component properties
Interaction Behaviour

Semantic Domains
Interaction Behaviour

Semantic Domains

- Traces (or runs, executions, paths)
Interaction Behaviour

Semantic Domains

- Traces (or runs, executions, paths)
- Computation trees
Interaction Behaviour

Semantic Domains

- Traces (or runs, executions, paths)
- Computation trees
- Labelled Transition Systems (LTS)
Interaction Behaviour

Semantic Domains

- Traces (or runs, executions, paths)
- Computation trees
- Labelled Transition Systems (LTS)
- Modal Transition Systems
LTS Example: Concurrent Server
Interaction Behaviour

Defining Interaction Behaviour
Interaction Behaviour

Defining Interaction Behaviour

- Process Algebra: Calculus of Communicating Systems
Interaction Behaviour

Defining Interaction Behaviour

- Process Algebra: Calculus of Communicating Systems
- Programming language: Erlang
Interaction Behaviour

Defining Interaction Behaviour

- Process Algebra: Calculus of Communicating Systems
- Programming language: Erlang
- Control Flow Graph: extracted from Java bytecode
Interaction Behaviour

Defining Interaction Behaviour

- Process Algebra: Calculus of Communicating Systems
- Programming language: Erlang
- Control Flow Graph: extracted from Java bytecode

LTS Semantics

Induced by *transition rules*
Calculus of Communicating Systems (CCS)
Calculus of Communicating Systems (CCS)

CCS Syntax

\[ E ::= 0 | A | \alpha.E | E + E | E\|E \]
Calculus of Communicating Systems (CCS)

CCS Syntax

\[ E ::= 0 \mid A \mid \alpha.E \mid E + E \mid E|E \]

CCS Semantics: Transition Rules (induce LTS)

**PREFIX**

\[ \alpha.E \xrightarrow{\alpha} E \]

**DEF**

\[ E \xrightarrow{\alpha} F \quad A \triangleq E \]

**CHOICE**

\[ E \xrightarrow{\alpha} E' \quad E + F \xrightarrow{\alpha} E' \]

**COMM**

\[ E \xrightarrow{\alpha} E' \quad E|F \xrightarrow{\alpha} E'|F \]
CCS Example: Concurrent Server
### CCS Example: Concurrent Server

**Defining Concurrent Server**

\[ CServer \triangleq request.(CServer | response.0) \]
CCS Example: Concurrent Server

Defining Concurrent Server

\[
\text{CServer} \triangleq \text{request.}(\text{CServer} \mid \text{response.}0)
\]

Induced LTS

```
request
CServer
|response.0
response
request
CServer
|response.0
|response.0
response
request
CServer
|response.0
|response.0
response
```

\[
\ldots
\]
## Specifying Behavioural Properties

### Specifying Sets of Behaviours
Specifying Behavioural Properties

Specifying Sets of Behaviours

- Modal logic: Hennessy-Milner Logic (HML)
Specifying Behavioural Properties

Specifying Sets of Behaviours

- Modal logic: Hennessy-Milner Logic (HML)
- Temporal logic: Computation Tree Logic (CTL)
## Specifying Behavioural Properties

### Specifying Sets of Behaviours

- **Modal logic:** Hennessy-Milner Logic (HML)
- **Temporal logic:** Computation Tree Logic (CTL)
- **Modal \( \mu \)-calculus:** HML + Recursion (\( \mu K \))
Specifying Behavioural Properties

Specifying Sets of Behaviours

- Modal logic: Hennessy-Milner Logic (HML)
- Temporal logic: Computation Tree Logic (CTL)
- Modal $\mu$–calculus: HML + Recursion ($\mu$K)

Example: Formalizing STAB

- CTL: $\text{AG (AF stab)}$
Specifying Behavioural Properties

Specifying Sets of Behaviours

- Modal logic: Hennessy-Milner Logic (HML)
- Temporal logic: Computation Tree Logic (CTL)
- Modal μ-calculus: HML + Recursion (μK)

Example: Formalizing STAB

- CTL: AG (AF stab)
- μK: \( \nu X. \mu Y. \text{[request]} X \land \text{[¬request]} Y \)
<table>
<thead>
<tr>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hennessy-Milner Logic (HML)</strong></td>
</tr>
</tbody>
</table>

Dilian Gurov
Theoretical Computer Science Department, KTH Royal Institute of Technology

Compositional Verification of Interaction Behaviour
Hennessy-Milner Logic (HML)

HML Syntax

\[ \Phi ::= \texttt{tt} \mid \texttt{ff} \mid \Phi \lor \Phi \mid \Phi \land \Phi \mid \langle \alpha \rangle \Phi \mid [\alpha] \Phi \]
Hennessy-Milner Logic (HML)

**HML Syntax**

\[ \Phi ::= \texttt{tt} \mid \texttt{ff} \mid \Phi \lor \Phi \mid \Phi \land \Phi \mid \langle \alpha \rangle \Phi \mid [\alpha] \Phi \]

**HML Semantics: Satisfaction Relation** \( s \models^T \Phi \)

\[ s \models^T \langle \alpha \rangle \Phi \iff \exists s' \in S. (s \xrightarrow{\alpha} s' \land s' \models^T \Phi) \]

\[ s \models^T [\alpha] \Phi \iff \forall s' \in S. (s \xrightarrow{\alpha} s' \Rightarrow s' \models^T \Phi) \]
Verifying Behavioural Properties: Interactive
Verifying Behavioural Properties: Interactive

Proof System Based: Judgements $s \vdash^T \Phi$

\[
\begin{align*}
\text{TRUE} & \quad \frac{\text{} }{s \vdash^T \tt} \\
\text{ORL} & \quad \frac{s \vdash^T \Phi}{s \vdash^T \Phi \lor \Psi} \\
\text{ORR} & \quad \frac{s \vdash^T \Psi}{s \vdash^T \Phi \lor \Psi} \\
\text{AND} & \quad \frac{s \vdash^T \Phi \quad s \vdash^T \Psi}{s \vdash^T \Phi \land \Psi} \\
\text{DIA} & \quad \frac{s' \vdash^T \Phi}{s \vdash^T \langle \alpha \rangle \Phi} \quad s' \in \partial_\alpha(s) \\
\text{Box} & \quad \frac{s_1 \vdash^T \Phi \ldots s_n \vdash^T \Phi}{s \vdash^T [\alpha] \Phi} \quad \partial_\alpha(s) = \{s_1, \ldots, s_n\}
\end{align*}
\]
Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^T \Phi$
Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^T \Phi$

- *local* techniques: execute $s$ guided by $\Phi$
  - proof strategies give rise to MC algorithms
Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^T \Phi$

- **local** techniques: execute $s$ guided by $\Phi$
  proof strategies give rise to MC algorithms
- **global** techniques: compute all $\Phi$–states, check membership
Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^T \Phi$

- *local* techniques: execute $s$ guided by $\Phi$
  - proof strategies give rise to MC algorithms
- *global* techniques: compute all $\Phi$–states, check membership

Complexity of Model Checking

- For Finite–State Systems:
  - polynomial in size of model, exponential in size of formula
Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^T \Phi$

- *local* techniques: execute $s$ guided by $\Phi$
  proof strategies give rise to MC algorithms
- *global* techniques: compute all $\Phi$–states, check membership

Complexity of Model Checking

- For Finite–State Systems:
  polynomial in size of model, exponential in size of formula
- For Pushdown Automata:
  exponential in number of non–terminals and in size of formula
Compositional Verification
Compositional Verification

Task to prove:

$X : \text{STAB}$

$\text{Handler}$
Compositional Verification

**Task to prove:**

\[ X : \text{STAB} \]

**Notation:**

\[ X : \text{STAB} \models X|\text{Handler} : \text{STAB} \]

Dilian Gurov
Theoretical Computer Science Department
KTH Royal Institute of Technology

Compositional Verification of Interaction Behaviour
Compositional Verification

Task to prove:

\[ X : \text{STAB} \models X | \text{Handler} : \text{STAB} \]

Notation:

\[ X : \text{STAB} \models X | \text{Handler} : \text{STAB} \]

Approaches:

- Interactive: proof systems
- Algorithmic: maximal models
Proof System for Compositional Verification
Proof System for Compositional Verification

Judgements

\[ \Gamma \preceq \Delta \]  where \( \Gamma, \Delta \) are sets of assertions
Proof System for Compositional Verification

Judgements

Γ ⊨ Δ where Γ, Δ are sets of assertions

Term Cut Rule

TERM Cut

\[ \frac{\vdash C : \Phi \quad X : \Phi \vdash X | E : \Psi}{\vdash C | E : \Psi} \]
Proof System for Compositional Verification

**Judgements**

\[ \Gamma \vdash \Delta \] where \( \Gamma, \Delta \) are sets of assertions

**Term Cut Rule**

\[
\begin{align*}
\text{TERM\textsc{Cut}} & : \quad \vdash C : \Phi \quad X : \Phi \vdash X \mid E : \Psi \\
& \quad \vdash C \mid E : \Psi
\end{align*}
\]

**Global Discharge Rule**

- explicit ordinal approximation
- proof tree embodies a valid proof by well-founded induction
- powerful mechanism for inductive and co-inductive proofs
# Proving Stabilization of Concurrent Server

Proof Systems

Dilian Gurov  
Theoretical Computer Science Department  
KTH Royal Institute of Technology  

Compositional Verification of Interaction Behaviour
Proving Stabilization of Concurrent Server

Proof Outline

Proof Systems

Dilian Gurov
Theoretical Computer Science Department
KTH Royal Institute of Technology

Compositional Verification of Interaction Behaviour
Proof System for Compositional Verification

Properties
Proof System for Compositional Verification

Properties

- sound: only valid judgements are derivable
Proof System for Compositional Verification

Properties

- sound: only valid judgements are derivable
- incomplete in general:
  even $X : \Phi, Y : \Psi \models X \mid Y : \Theta$ is undecidable for $\mu K$!
Proof System for Compositional Verification

Properties

- sound: only valid judgements are derivable
- incomplete in general:
  even $X : \Phi, Y : \Psi \models X|Y : \Theta$ is undecidable for $\mu K!$
- complete for logic fragment:
  only variables as terms
Proof System for Compositional Verification

Properties

- sound: only valid judgements are derivable
- incomplete in general: even $X : \Phi, Y : \Psi \models X | Y : \Theta$ is undecidable for $\mu K$!
- complete for logic fragment: only variables as terms
- complete for model checking fragment: closed, regular CCS terms
Proof System for Compositional Verification

Properties

- sound: only valid judgements are derivable
- incomplete in general: even $X : \Phi, Y : \Psi \models X|Y : \Theta$ is undecidable for $\mu K$!
- complete for logic fragment: only variables as terms
- complete for model checking fragment: closed, regular CCS terms
- complete for pushdown automata
Maximal Models for Compositional Verification

Under certain conditions...
Maximal Models for Compositional Verification

Under certain conditions...

proof goal:

\[
\text{X : STAB} \\
\text{Handler} \\
: \text{STAB}
\]
Maximal Models for Compositional Verification

Under certain conditions...

proof goal:

\[ X : \text{STAB} \]

: \text{STAB}

Handler

...reduces to model checking:

\[ \text{Max(STAB)} \]

: \text{STAB}

Handler
Maximal Models for Compositional Verification

Conditions

There is a (simulation) pre-order \( \leq \) on components:
Maximal Models for Compositional Verification

Conditions

There is a (simulation) pre–order \( \leq \) on components:

1. **property preserving:**
   
   \( C_1 \leq C_2 \) and \( \models C_2 : \Phi \) imply \( \models C_1 : \Phi \)
Maximal Models for Compositional Verification

**Conditions**

There is a (simulation) pre-order $\leq$ on components:

1. **property preserving:**
   
   $C_1 \leq C_2$ and $\models C_2 : \Phi$ imply $\models C_1 : \Phi$

2. **preserved by composition:**

   $C_1 \leq C_2$ implies $C_1 | C_3 \leq C_2 | C_3$
Maximal Models for Compositional Verification

Conditions

There is a (simulation) pre–order \( \leq \) on components:

1. property preserving:
   \[ C_1 \leq C_2 \text{ and } \models C_2 : \Phi \text{ imply } \models C_1 : \Phi \]

2. preserved by composition:
   \[ C_1 \leq C_2 \text{ implies } C_1|C_3 \leq C_2|C_3 \]

3. the set of \( \Phi \)-components has a maximal element w.r.t. \( \leq \)
Maximal Models

Conditions

There is a (simulation) pre–order \( \leq \) on components:

1. property preserving:
   \[ C_1 \leq C_2 \text{ and } \models C_2 : \Phi \text{ imply } \models C_1 : \Phi \]

2. preserved by composition:
   \[ C_1 \leq C_2 \text{ implies } C_1|C_3 \leq C_2|C_3 \]

3. the set of \( \Phi \)–components has a maximal element w.r.t. \( \leq \)

Maximal Model Principle

\[ \text{MaxMod} \quad \models \text{Max}(\Phi)|E : \Psi \quad \frac{X : \Phi \models X|E : \Psi}{\text{MaxMod}} \]
Maximal Models for Compositional Verification

**Derived Compositional Verification Principle**

\[
\begin{align*}
\text{COMPOS} & \quad \models C : \Phi \quad \models \text{Max}(\Phi) | E : \Psi \\
& \quad \models C | E : \Psi
\end{align*}
\]
Maximal Models for Compositional Verification

 Derived Compositional Verification Principle

\[
\text{COMPOS} \quad \models C : \Phi \quad \models \text{Max}(\Phi) \mid E : \Psi \quad \models C \mid E : \Psi
\]

Applies to:

1. ACTL (Kripke models)
2. Simulation Logic (Control Flow Graphs)
3. modal $\mu$–calculus (EMTS)
Conclusions

Interaction Behaviour

Interaction behaviour can be:
Conclusions

Interaction Behaviour

Interaction behaviour can be:

- captured elegantly through LTS
Conclusions

Interaction Behaviour

Interaction behaviour can be:

- captured elegantly through LTS
- defined in various ways: CCS, Erlang, Control Flow Graphs
Conclusions

Interaction Behaviour

Interaction behaviour can be:

- captured elegantly through LTS
- defined in various ways: CCS, Erlang, Control Flow Graphs
- specified in various logics: HML, CTL, μK
Interaction Behaviour

Interaction behaviour can be:

- captured elegantly through LTS
- defined in various ways: CCS, Erlang, Control Flow Graphs
- specified in various logics: HML, CTL, $\mu$K
- verified algorithmically or interactively
Conclusions

Interaction Behaviour

Interaction behaviour can be:

- captured elegantly through LTS
- defined in various ways: CCS, Erlang, Control Flow Graphs
- specified in various logics: HML, CTL, $\mu$K
- verified algorithmically or interactively

Compositional Verification

- good for modular design
Conclusions

**Interaction Behaviour**

Interaction behaviour can be:

- captured elegantly through LTS
- defined in various ways: CCS, Erlang, Control Flow Graphs
- specified in various logics: HML, CTL, $\mu$K
- verified algorithmically or interactively

**Compositional Verification**

- good for modular design
- needed for verifying open systems
Conclusions

Interaction Behaviour

Interaction behaviour can be:

- captured elegantly through LTS
- defined in various ways: CCS, Erlang, Control Flow Graphs
- specified in various logics: HML, CTL, $\mu$K
- verified algorithmically or interactively

Compositional Verification

- good for modular design
- needed for verifying open systems
- algorithmically or interactively
Future Challenges
Future Challenges

Interactive Verification

How to reason about *complex phenomena* such as:

- failure and recovery
- self–stabilization

in open, dynamic systems?
Future Challenges

Interactive Verification

How to reason about *complex phenomena* such as:
- failure and recovery
- self–stabilization
in open, dynamic systems?

Algorithmic Verification

How to achieve *scalability* of verification?
- separating concerns
- abstraction mechanisms