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

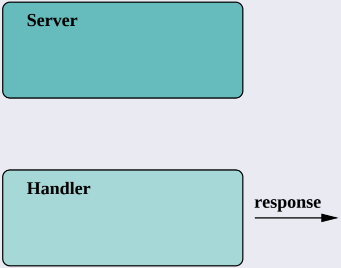
Concurrent Server

Server gets request

Concurrent Server

Server gets request

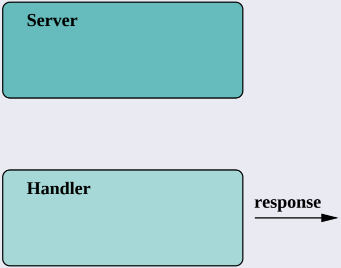
Server spawns off Handler



Concurrent Server

Server gets request

Server spawns off Handler



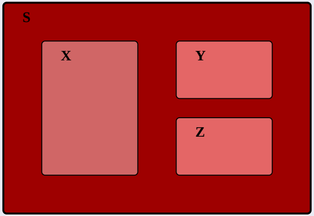
Property of Interaction Behaviour

Concurrent server always stabilizes (STAB)

Compositional Reasoning

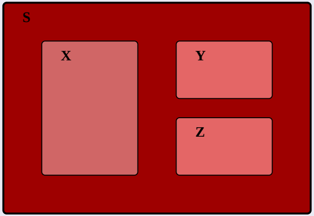
Compositional Reasoning

Pure approach: Composing properties

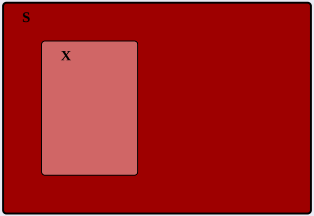


Compositional Reasoning

**Pure approach:
Composing properties**

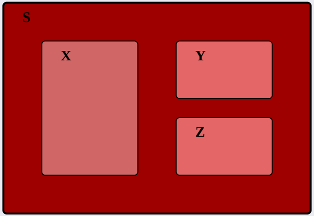


**More general approach:
Cut on component**

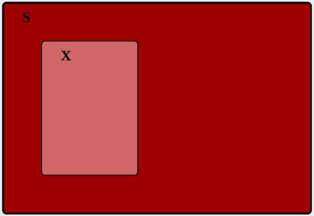


Compositional Reasoning

Pure approach: Composing properties



More general approach: Cut on component



Concurrent Server

How does compositional reasoning help?

Proving Stabilization of Concurrent Server

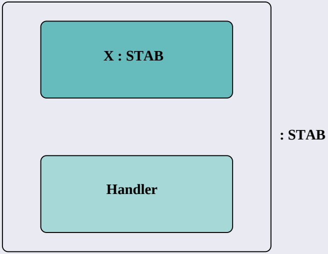
Proving Stabilization of Concurrent Server

Original goal:

Proving Stabilization of Concurrent Server

Original goal:

Reduces to:



Overview

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

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

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)

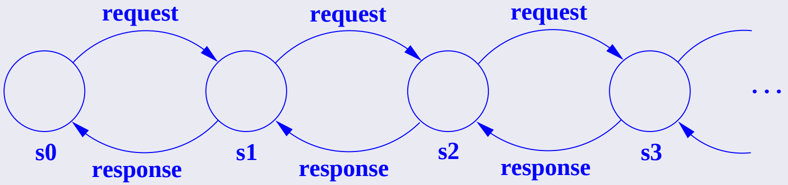
Interaction Behaviour

Semantic Domains

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

LTS Example: Concurrent Server

Interaction Behaviour of 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

$$E \ ::= \ \mathbf{0} \ | \ A \ | \ \alpha.E \ | \ E + E \ | \ E|E$$

Calculus of Communicating Systems (CCS)

CCS Syntax

$$E \ ::= \ \mathbf{0} \ | \ A \ | \ \alpha.E \ | \ E+E \ | \ E|E$$

CCS Semantics: Transition Rules (induce LTS)

$$\textsc{Prefix} \ \frac{-}{\alpha.E \xrightarrow{\alpha} E} \qquad \textsc{Def} \ \frac{E \xrightarrow{\alpha} F}{A \xrightarrow{\alpha} F} \ A \triangleq E$$

$$\textsc{Choice} \ \frac{E \xrightarrow{\alpha} E'}{E+F \xrightarrow{\alpha} E'} \qquad \textsc{Comm} \ \frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F}$$

CCS Example: Concurrent Server

CCS Example: Concurrent Server

Defining Concurrent Server

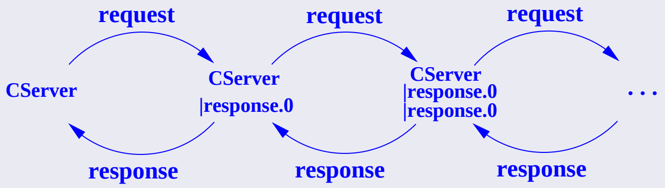
$$CServer \triangleq request.(CServer \mid response.\mathbf{0})$$

CCS Example: Concurrent Server

Defining Concurrent Server

$$CServer \triangleq request.(CServer \mid response.\mathbf{0})$$

Induced LTS



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 μ–calculus: HML + Recursion (μK)

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)

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: $\nu X.\ \mu Y.\ [\text{request}]\, X \wedge [-\text{request}]\ Y$

Hennessy-Milner Logic (HML)

Hennessy-Milner Logic (HML)

HML Syntax

$$\Phi \ ::= \ \mathbf{tt} \ | \ \mathbf{ff} \ | \ \Phi \vee \Phi \ | \ \Phi \wedge \Phi \ | \ \langle\alpha\rangle\,\Phi \ | \ [\alpha]\,\Phi$$

Hennessy-Milner Logic (HML)

HML Syntax

$$\Phi \;::=\; \mathbf{tt} \;\mid\; \mathbf{ff} \;\mid\; \Phi \vee \Phi \;\mid\; \Phi \wedge \Phi \;\mid\; \langle\alpha\rangle\,\Phi \;\mid\; [\alpha]\,\Phi$$

HML Semantics: Satisfaction Relation $s \models^{\mathcal{T}} \Phi$

$$s \models^{\mathcal{T}} \langle\alpha\rangle\,\Phi \;\stackrel{\text{def}}{\Leftrightarrow}\; \exists s' \in \mathcal{S}.\,(s \xrightarrow{\alpha} s' \wedge s' \models^{\mathcal{T}} \Phi)$$
$$s \models^{\mathcal{T}} [\alpha]\,\Phi \;\stackrel{\text{def}}{\Leftrightarrow}\; \forall s' \in \mathcal{S}.\,(s \xrightarrow{\alpha} s' \Rightarrow s' \models^{\mathcal{T}} \Phi)$$

Verifying Behavioural Properties: Interactive

Verifying Behavioural Properties: Interactive

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

$$\textsc{True}\ \frac{-}{s \vdash^{\mathcal{T}} \mathbf{tt}} \qquad \textsc{OrL}\ \frac{s \vdash^{\mathcal{T}} \Phi}{s \vdash^{\mathcal{T}} \Phi \vee \Psi} \qquad \textsc{OrR}\ \frac{s \vdash^{\mathcal{T}} \Psi}{s \vdash^{\mathcal{T}} \Phi \vee \Psi}$$

$$\textsc{And}\ \frac{s \vdash^{\mathcal{T}} \Phi \qquad s \vdash^{\mathcal{T}} \Psi}{s \vdash^{\mathcal{T}} \Phi \wedge \Psi} \qquad \textsc{Dia}\ \frac{s' \vdash^{\mathcal{T}} \Phi}{s \vdash^{\mathcal{T}} \langle \alpha \rangle \Phi}\ s' \in \partial_\alpha(s)$$

$$\textsc{Box}\ \frac{s_1 \vdash^{\mathcal{T}} \Phi \ldots s_n \vdash^{\mathcal{T}} \Phi}{s \vdash^{\mathcal{T}} [\alpha]\,\Phi}\ \partial_\alpha(s) = \{s_1, \ldots, s_n\}$$

Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^{\mathcal{T}} \Phi$

Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^{\mathcal{T}} \Phi$

- *local* techniques: execute s guided by Φ
  proof strategies give rise to MC algorithms

Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^{\mathcal{T}} \Phi$

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

Verifying Behavioural Properties: Algorithmic

Model Checking $s \models^{\mathcal{T}} \Phi$

- *local* techniques: execute s guided by Φ
  proof strategies give rise to MC algorithms
- *global* techniques: compute all Φ–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 $s \models^{\mathcal{T}} \Phi$

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

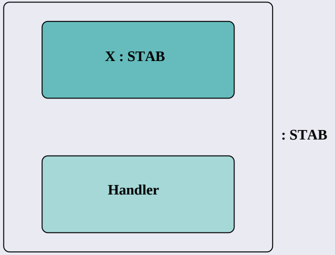
Complexity of Model Checking

- 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

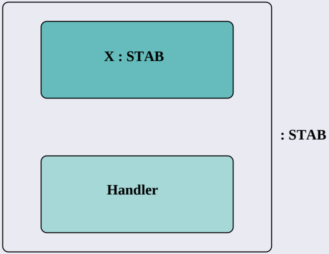
Compositional Verification

Task to prove:



Compositional Verification

Task to prove:

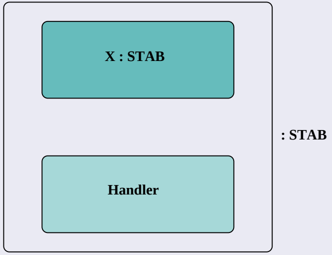


Notation:

$X : \mathrm{STAB} \models X | \mathrm{Handler} : \mathrm{STAB}$

Compositional Verification

Task to prove:



Notation:

$X : \mathrm{STAB} \models X | \mathrm{Handler} : \mathrm{STAB}$

Approaches:

- Interactive: proof systems
- Algorithmic: maximal models

Proof System for Compositional Verification

Proof System for Compositional Verification

Judgements

$\Gamma \vdash \Delta$ where Γ, Δ are sets of assertions

Proof System for Compositional Verification

Judgements

$\Gamma \vdash \Delta$ where Γ, Δ are sets of assertions

Term Cut Rule

$$\textsc{TermCut} \quad \frac{\vdash C : \Phi \qquad X : \Phi \vdash X|E : \Psi}{\vdash C|E : \Psi}$$

Proof System for Compositional Verification

Judgements

$\Gamma \vdash \Delta$ where Γ, Δ are sets of assertions

Term Cut Rule

$$\textsc{TermCut} \quad \frac{\vdash C : \Phi \qquad X : \Phi \vdash X|E : \Psi}{\vdash C|E : \Psi}$$

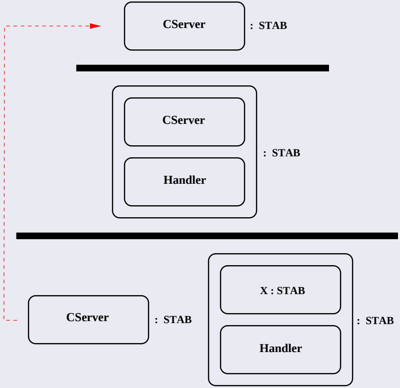
Global Discharge Rule

- 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

Proving Stabilization of Concurrent Server

Proof Outline



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 $X : \Phi, Y : \Psi \models X|Y : \Theta$ is undecidable for μK!

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 μK!
- 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 μK!
- 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 μK!
- complete for logic fragment:
  only variables as terms
- complete for model checking fragment:
  closed, regular CCS terms
- complete for pushdown automata

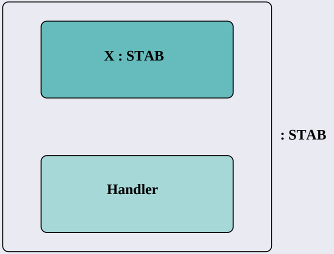
Maximal Models for Compositional Verification

Under certain conditions...

Maximal Models for Compositional Verification

Under certain conditions...

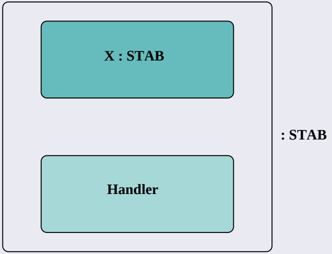
proof goal:



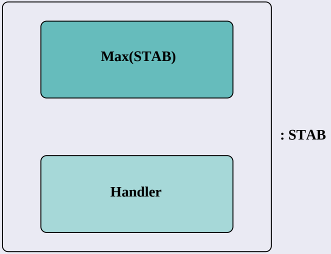
Maximal Models for Compositional Verification

Under certain conditions...

proof goal:



...reduces to model checking:



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:
   $C_1 \leq C_2$ and $\models C_2 : \Phi$ imply $\models C_1 : \Phi$
2. preserved by composition:
   $C_1 \leq C_2$ implies $C_1 | C_3 \leq C_2 | C_3$
3. the set of Φ–components has a maximal element w.r.t. $\leq$

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 $C_1|C_3 \leq C_2|C_3$
3. the set of Φ–components has a maximal element w.r.t. $\leq$

Maximal Model Principle

$$\textsc{MaxMod} \quad \frac{\models Max(\Phi)|E : \Psi}{X : \Phi \models X|E : \Psi}$$

Maximal Models for Compositional Verification

Derived Compositional Verification Principle

$$\textsc{Compos} \quad \frac{\models C : \Phi \qquad \models Max(\Phi) | E : \Psi}{\models C | E : \Psi}$$

Maximal Models for Compositional Verification

Derived Compositional Verification Principle

$$\textsc{Compos}\quad \frac{\models C : \Phi \qquad \models Max(\Phi)|E : \Psi}{\models C|E : \Psi}$$

Applies to:

1. ACTL (Kripke models)
2. Simulation Logic (Control Flow Graphs)
3. modal μ–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

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
- 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, μK
- 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, μK
- 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, μK
- 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