Reducing Behavioural to Structural Control flow-based Properties of Sequential Programs with Procedures

Dilian Gurov
KTH Stockholm, Sweden

Joint work with:

Marieke Huisman
University of Twente, The Netherlands

VMCAI 2009, Savannah, GA

20 January 2009

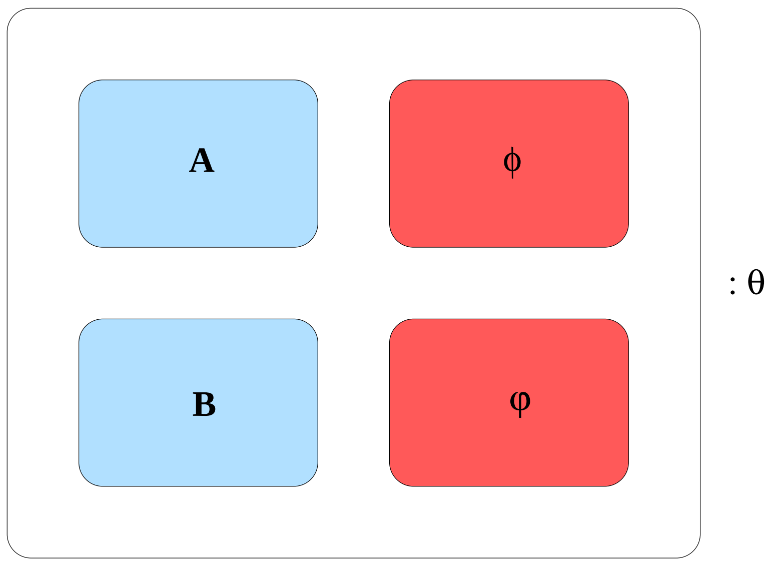
Overview

1. A Framework for Algorithmic Compositional Verification
   (a) General Framework based on Maximal Models
   (b) Program Model: Flow Graphs and Flow Graph Behaviour
   (c) Maximal Flow Graphs for Structural and Behavioural Properties
2. Property Translation
   (a) Example and Applications
   (b) Tableau Construction
   (c) Correctness
3. Conclusions and Future Work

1. Framework for Model Checking Open Systems

Open system: some components are only given by a specification:

abstract components



General Method [Grumberg-Long-94]: replace every abstract component by a concrete representative: maximal model

Refinement Preorder:

$$\mathcal{M}_1 \preceq \mathcal{M}_2 \overset{\text{def}}{\iff} \forall \phi.\, (\mathcal{M}_2 \models \phi \Rightarrow \mathcal{M}_1 \models \phi)$$ (simulation)

Framework Conditions:

1. for any formula ψ, the set of models for ψ has a greatest element $Max(\psi)$ w.r.t. the preorder: maximal model
2. preorder preserved by model composition

Our Set-up:

- **Models**: Labelled Transition Systems with Valuations
- **Logic**: $\phi ::= p \mid \neg p \mid X \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid [a]\,\phi \mid \nu X.\phi$

Program Model

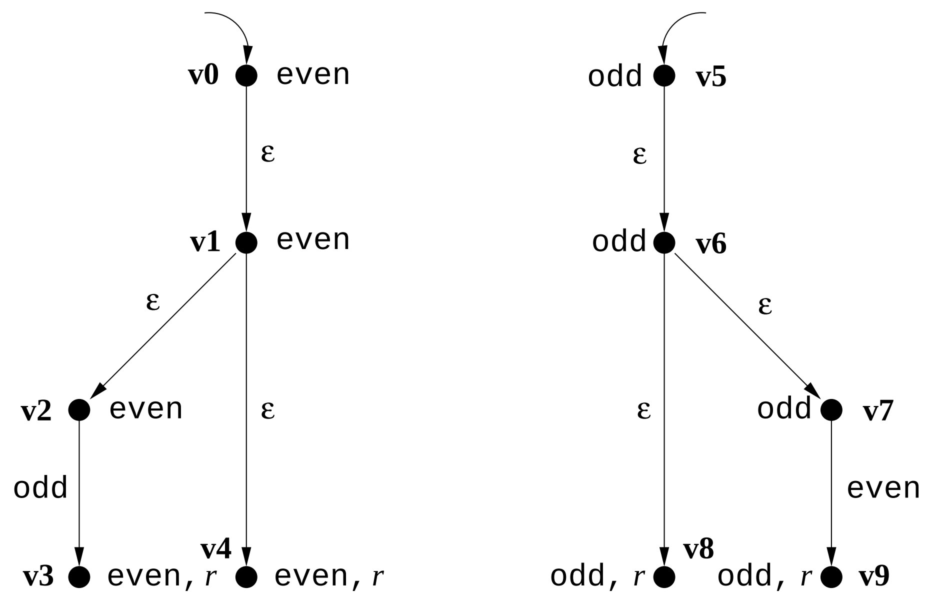
Control Flow Structure: Flow Graphs

```
class Number {

   public static boolean even(int n){
      if (n == 0)
         return true;
      else
         return odd(n-1);
   }

   public static boolean odd(int n){
      if (n == 0)
         return false;
      else
         return even(n-1);

   }
}
```



Flow graph composition: (disjoint) union of graphs

Flow Graph Behaviour

- flow graph induces pushdown automaton (PDA):
  - configurations (v, σ) are pairs of control point v and call stack σ
  - productions induced by:
    - ☞ non-call edges
    - ☞ call edges
    - ☞ return nodes
- flow graph behaviour is behaviour of induced PDA

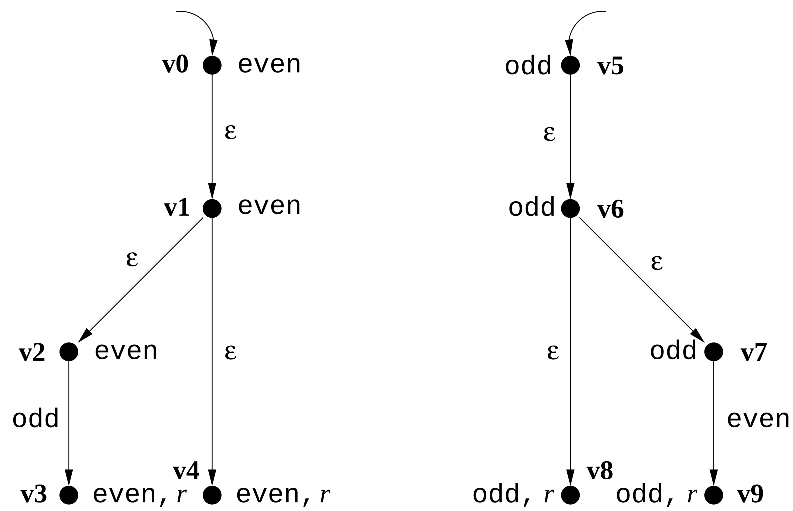
Example Flow Graph:

```
class Number {

   public static boolean even(int n){
      if (n == 0)
         return true;
      else
         return odd(n-1);
   }

   public static boolean odd(int n){
      if (n == 0)
         return false;
      else
         return even(n-1);

   }
}
```



Example Run:

$$(v_0, \epsilon) \xrightarrow{\tau}_b (v_1, \epsilon) \xrightarrow{\tau}_b (v_2, \epsilon) \xrightarrow{\text{even call odd}}_b (v_5, v_3) \xrightarrow{\tau}_b (v_6, v_3) \xrightarrow{\tau}_b$$

$$(v_7, v_3) \xrightarrow{\text{odd call even}}_b (v_0, v_9 \cdot v_3) \xrightarrow{\tau}_b (v_1, v_9 \cdot v_3) \xrightarrow{\tau}_b$$

$$(v_4, v_9 \cdot v_3) \xrightarrow{\text{even ret odd}}_b (v_9, v_3) \xrightarrow{\text{odd ret even}}_b (v_3, \epsilon)$$

Property Specification

Logic:

- fragment of μ-calculus: safety properties
- instantiated to structure and behaviour

Example structural property:

- program is tail–recursive: $\nu X.\ [\mathsf{even}]\, r \wedge [\mathsf{odd}]\, r \wedge [\varepsilon]\, X$

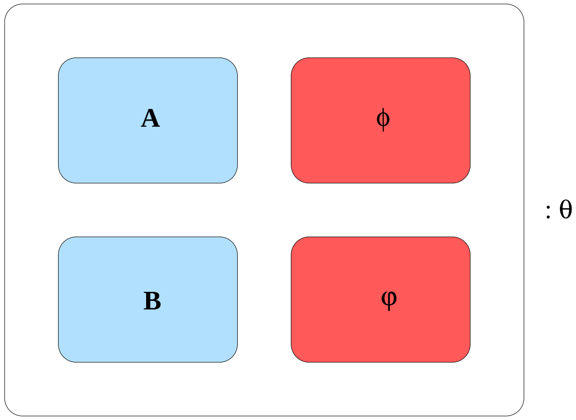
Example behavioural property:

- first call of **even** is not to itself: $\mathsf{even} \Rightarrow \nu X.\ [\mathsf{even\ call\ even}]\, \mathsf{ff} \wedge [\tau]\, X$

Model Checking Closed Systems

- Extract flow graph from program code
- For structural properties:
  1. cast flow graph as finite automaton
  2. apply standard, finite–state model checking
- For behavioural properties:
  1. cast flow graph as pushdown automaton
  2. apply PDA model checking

Model Checking Open Systems



Idea: replace every abstract component by a flow graph

Structural Properties: unique maximal flow graph for given sets of provided and required methods: flow graph interface, part of the specification

Behavioural Properties: more problematic

Problem

- in general: maximal flow graphs for behavioural properties not unique
- example: $[a\ \mathsf{call}\ b]\ r$ gives rise to two maximal flow graphs
- question: how can we compute these?

Proposed Approach: via property translation (present contribution)

- characterise behavioural property through set of structural ones:
  - structural property: $a \Rightarrow [b]\ \mathsf{ff}$
  - structural property: $b \Rightarrow r$
- eliminate subsumed properties (optional)
- construct the maximal flow graphs for the structural properties

Verification Method for Open Systems:

1. for concrete components:
   - extract flow graphs
2. for abstract components, from specification:
   - if structural, construct maximal flow graph
   - if behavioural,
     (a) translate to equivalent set of structural properties
     (b) construct maximal flow graphs
3. for all compositions of extracted with constructed flow graphs:
   - model check system flow graph against system property

2. Property Translation

Example for programs with methods a and b only

- Behavioural property:
  - "method a never calls method b"

    $\nu X.\, [a\ \mathsf{call}\ b]\, \mathsf{ff} \wedge [\tau]\, X \wedge [a\ \mathsf{call}\ a]\, X \wedge [a\ \mathsf{ret}\ a]\, X$

- is characterised by the structural properties:
  - "in the text of method a there is no call–to–b instruction"

    $a \Rightarrow \nu X.\, [b]\, \mathsf{ff} \wedge [\varepsilon]\, X \wedge [a]\, X$

  - "in the text of method a every return instruction and every call–to–b instruction is preceded by some call–to–a instruction"

    $a \Rightarrow \nu X. \neg r \wedge [b]\, \mathsf{ff} \wedge [\varepsilon]\, X$

Applications of Translation

- Maximal flow graphs for
  - compositional verification of behavioural properties
  - synthesis of program skeletons from behavioural specifications
- Foundational value: structure $\leftrightarrow$ behaviour

  in terms of temporal logic
- Enforcing behavioural properties through structure
- Reducing infinite–state behavioural model checking

  to finite–state structural model checking

The Translation

Idea

- symbolic execution of behavioural formula
- accumulating structural constraints on the way
- by means of history stack: $(m, F) \cdot H$

For modal fragment

- simple mapping π_H

  defined inductively on the structure of the formula
- presented at: FESCA 2007

Modal Fragment: Mapping π_H

$$
\begin{aligned}
\pi_{(i,F)\cdot H}(p) &= \{i \Rightarrow [F]\, p\} \cup \{i' \Rightarrow [F']\, \mathsf{ff} \mid (i', F') \in H\} \\
\pi_{(i,F)\cdot H}(\neg p) &= \{i \Rightarrow [F]\, \neg p\} \cup \{i' \Rightarrow [F']\, \mathsf{ff} \mid (i', F') \in H\} \\
\pi_{(i,F)\cdot H}(\phi_1 \wedge \phi_2) &= \{\sigma_1 \wedge \sigma_2 \mid \sigma_1 \in \pi_{(i,F)\cdot H}(\phi_1),\ \sigma_2 \in \pi_{(i,F)\cdot H}(\phi_2)\} \\
\pi_{(i,F)\cdot H}(\phi_1 \vee \phi_2) &= \pi_{(i,F)\cdot H}(\phi_1) \cup \pi_{(i,F)\cdot H}(\phi_2) \\
\pi_{(i,F)\cdot H}([\tau]\, \phi) &= \pi_{(i,F\cdot\varepsilon)\cdot H}(\phi) \\
\pi_{(i,F)\cdot H}([a\ \mathsf{call}\ b]\, \phi) &= \begin{cases} \{\mathsf{tt}\} & \text{if } i \neq a \\ \pi_{(b,\epsilon)\cdot(i,F\cdot b)\cdot H}(\phi) & \text{if } i = a \end{cases} \\
\pi_{(i,F)\cdot H}([a\ \mathsf{ret}\ b]\, \phi) &= \begin{cases} \{\mathsf{tt}\} & \text{if } i \neq a\ \vee\ \ldots \\ \{i \Rightarrow [F]\, \neg r\} \cup \pi_H(\phi) & \text{if } i = a\ \wedge\ \ldots \end{cases}
\end{aligned}
$$

Modal Fragment: Examples

Example1

$$\begin{aligned} \pi_{(a,\epsilon)}([a \ \mathsf{call}\ b]\, r) &= \pi_{(b,\epsilon)\cdot(a,b)}(r) \\ &= \{b \Rightarrow r, a \Rightarrow [b]\, \mathsf{ff}\} \end{aligned}$$

Example2

$$\begin{aligned} \pi_{(a,\epsilon)}([a \ \mathsf{call}\ b]\, [a \ \mathsf{call}\ b]\, r) &= \pi_{(b,\epsilon)\cdot(a,b)}([a \ \mathsf{call}\ b]\, r) \\ &= \{\mathsf{tt}\} \end{aligned}$$

Full Logic

Dealing with fixed points: much more involved

- we need to identify termination conditions that guarantee:
  - structural constraints can be "folded" into fixed–point formulae
  - no new structural constraints will emerge

Approach

- in the frames, record also current formula
- use tableau construction, define global repeat conditions

  – allows correctness proof by viewing tableaux as proofs!
- from leaves, extract accumulated constraints

Tableau Construction

Tableau for behavioural formula: $\nu X.\,[a\ \mathsf{call}\ b]\,X \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge X)$

$$\vdash_{(a,\epsilon),\varnothing_U,\varnothing_C} \nu X.\,[a\ \mathsf{call}\ b]\,X \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge X) \quad \nu X$$

$$*\vdash_{(a,\epsilon),X=\phi,\varnothing_C} X \quad X\ \mathsf{unf}$$

$$\vdash_{(a,X_4),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\,X \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge X) \quad \wedge$$

$$\vdash_{(a,X_4\cdot X_1),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\,X \quad \mathsf{call}_1$$

$$\vdash_{(a,X_4\cdot X_1),X=\phi,\varnothing_C} [b\ \mathsf{ret}\ a]\,(\neg r \wedge X) \quad \mathsf{re}$$

$$-$$

$$\vdash_{(b,\epsilon)\cdot(a,X_4\cdot X_1\cdot X_2\cdot b),X=\phi,\varnothing_C} X \quad X\ \mathsf{unf}$$

$$_{X_4)\cdot(a,X_4\cdot X_1\cdot X_2\cdot b),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\,X \wedge [b\ \mathsf{ret}\ a]\,(\neg r \wedge X) \quad \wedge$$

$$_{(b,X_4\cdot X_1)\cdot(a,X_4\cdot X_1\cdot X_2\cdot b),X=\phi,\varnothing_C} [a\ \mathsf{call}\ b]\,X \quad \mathsf{call}_0 \qquad (*)$$

$$-$$

(*)

$$\dfrac{\dfrac{\dfrac{(*)}{\vdash_{(b, X_4 \cdot X_1) \cdot (a, X_4 \cdot X_1 \cdot X_2 \cdot b), X=\phi, \varnothing_C} [b \text{ ret } a] (\neg r \wedge X)} \wedge}{\vdash_{(a, X_4 \cdot X_1 \cdot X_2 \cdot b), X=\phi, \{(b, X_4 \cdot X_1, \neg r)\}} \neg r \wedge X} \text{ret}_1}{\dfrac{X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5), X=\phi, \{(b, X_4 \cdot X_1, \neg r)\} \; \neg r}{\begin{matrix}(a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5, \neg r) \\ (b, X_4 \cdot X_1, \neg r)\end{matrix}} \neg r \quad \dfrac{\vdash_{(a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5), X=\phi, \{(b, X_4 \cdot X_1, \neg r)\}} X}{\begin{matrix}(a, X_4 \cdot X_1 \cdot X_2 \cdot b \cdot X_5, X_4) \\ (b, X_4 \cdot X_1, \neg r)\end{matrix}} \text{IRep}(*)} \wedge$$

Extracted structural formulae

- $a \Rightarrow \nu X. [b] (\neg r \wedge X)$
- $b \Rightarrow \neg r$

Correctness of Tableau Construction

Idea

- view tableau rules as proof rules for proving that
  a set of structural properties χ entails a behavioural property ϕ

- a tableau for ϕ inducing χ converts to
  a proof that χ entails ϕ

Results

- soundness for full logic

- completeness for logic without disjunction

3. Conclusions

Achieved

- translation from behavioural to structural properties of program control flow
- implementation of translation, web–based interface
- application to compositional verification

Current limitations

- disjunction is over–approximated
- construction defined for closed interfaces

Future Work

We need to

- study disjunction: is there a complete translation?
- generalize construction to open interfaces, richer program models etc.
- study complexity of translation:
  - how many formulae?
  - of what size?
- study optimizations, subsumption checking etc.