Game Semantics Supported Component Verification

Aleksandar Dimovski Ranko Lazić

Department of Computer Science
University of Warwick, UK

BCTCS, 2006

Objectives, motivations

Traditional approaches for software model checking

- Operational semantics:
  Program denotes a state transition system
- Finite models by predicate abstraction
- Tools:
  SLAM [Microsoft], BLAST [Berkeley], Magic [CMU]

Objectives, motivations

Game semantics yields algorithms for software model checking

- based on Regular languages for 2nd–order finitary IA [Abramsky, Ghica, Murawski and Ong, TACAS 2004]
- based on Visibly pushdown automata for 3rd–order finitary IA [Murawski,Walukiewicz and Ong, FoSSaCS 2005]

Advantages of Game semantics based approach for software model checking:

- model for any open program fragment with higher-order procedures
- fully abstract semantic model (i.e. the model is sound and complete)
- compositional as denotational semantics
- generation of minimal models

Goal

Compositional verification

- addresses state explosion problem
- "divide and conquer" approach
- assume–guarantee reasoning
- $[\![\Gamma \vdash \text{let } f_1 \text{ be } N_1 \text{ in}$
  $\text{let } f_2 \text{ be } N_2 \text{ in}$
  $\ldots$
  $\text{let } f_k \text{ be } N_k \text{ in}$
  $M : T']\!] =$
  $[\![\Gamma \vdash N_1]\!] \fatsemi \Big([\![\Gamma, f_1 \vdash N_2]\!] \fatsemi \big(\ldots \fatsemi$
  $[\![\Gamma, f_1, \ldots, f_{k-1} \vdash N_k]\!] \fatsemi [\![\Gamma, f_1, \ldots, f_k \vdash M : T']\!] \big) \Big)$

Game semantics: the idea

- Computation seen as a play between a program – Player P and its environment (context) – Opponent O
- A play consists of a sequence of moves (questions and answers), alternating between players
- Model of a program is a *strategy* (set of plays) for Player P to respond to moves of Opponent O

Game semantics: Types

- Types are interpreted as *games*
- A game G is
  - a set of moves M_G
  - a labelling function $\lambda_G : M_G \rightarrow \{O, P\} \times \{Q, A\}$
  - an enabling relation, $\vdash_G$
  - legal plays, $P_G \subseteq M_G^*$

Game semantics: Programs

- Programs are interpreted as *strategies*
- A strategy σ for a game G is an even-length prefix-closed non empty set of legal plays of G
- Composition = "CSP-style parallel composition plus hiding"

Abstracted Idealized Algol (AIA)

- Imperative features, locally-scoped variables, call-by-name procedures
- Integer abstractions:
  $[] = \{\mathbb{Z}\}$
  $[n, m] = \{ < n, n, n+1, \ldots, m-1, m, > m \}, \text{ for } n \leq 0 \leq m$
- *abort* command causes abnormal termination
- Operational semantics:
  $1 +_{[0,1]} 1 \rightarrow \{2, 3, 4 \ldots\} \rightarrow > 1$
- For AIA_2, game-semantic models are regular languages

Assume-Guarantee proof rule

$$\frac{[\![\Gamma, f : T \vdash M : T']\!] \,\S\, \sigma \;\text{ is }\; \mathrm{SAFE} \qquad [\![\Gamma \vdash N : T]\!]^{\dagger} \leq \sigma}{[\![\Gamma \vdash \mathsf{let}\ f\ \mathsf{be}\ N\ \mathsf{in}\ M : T']\!] \;\text{ is }\; \mathrm{SAFE}}$$

- Given $[\![\Gamma, f : T \vdash M : T']\!]$, define a *weakest assumption strategy* σ_W, which contains legal plays from $P_{![\![T]\!]}$ which, when simulated on $[\![\Gamma, f : T \vdash M : T']\!]$ do not produce any unsafe plays
- For σ_W, AG proof rule is guaranteed to return a conclusive result
- Angluin's L^* algorithm is used for learning σ_W

L^* algorithm

L^* learns a strategy σ for G via queries

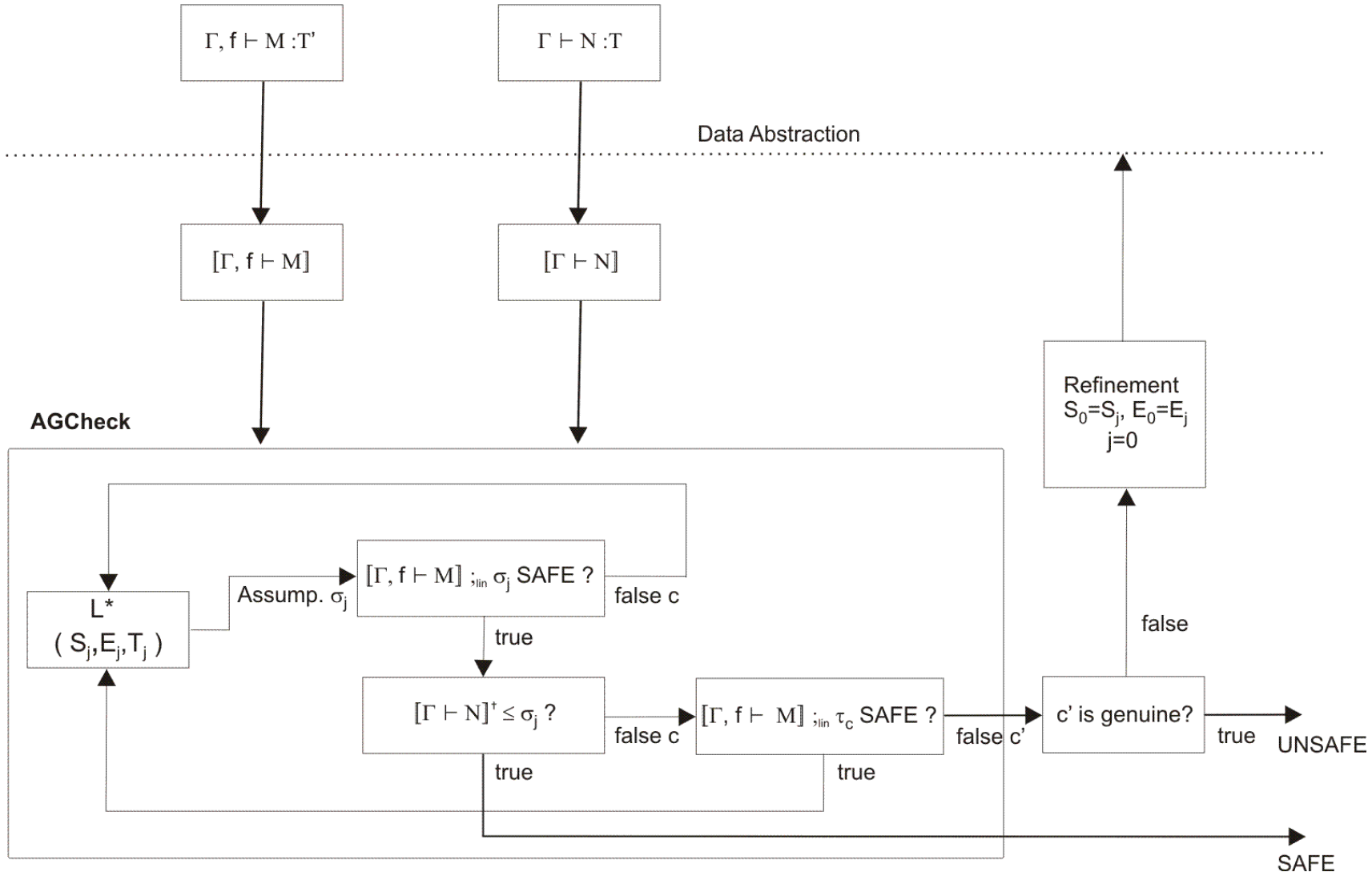
- *Membership query:* given $s \in (O_G P_G)^*$, whether $s \in \sigma$?
- *Equivalence query:* given a DFA D, whether $\mathcal{L}(D) = \sigma$?

L^* will generate the unknown strategy using at most $n - 1$ equivalence queries, where n is the number of states in the minimal DFA

Queries are implemented using model checking

- *Membership query:* given s, build a strategy $\sigma_s = \{s' \mid s' \sqsubseteq^{even} s\}$; then model check $[\![\Gamma, f \vdash M]\!] \wr \sigma_s$ for safety
- *Equivalence query:* given a DFA D, model check two premises of the AG rule

Verification procedure



Future work

- Concurrency, 3rd–order, call by value, pointers, ...
- Predicate abstraction