SMT-AI: an Abstract Interpreter as Oracle for $k$-induction

Pierre Roux, Pierre-Loïc Garoche, Rémi Delmas

ONERA – DTIM, Toulouse

September 17, 2010
1 Introduction

2 \textit{k}-induction

3 Interprétation abstraite

4 Collaboration
Rice Theorem

Every non trivial property on programs is undecidable.

To cope with undecidability, we can withdraw:

- termination;
- completeness;
- automaticity.
Various methods

- model-checking: numerous methods based on a more or less explicit study of the state space of the system;
- abstract interpretation: overapproximation (abstraction) of the semantic of the program, can answer “don’t know”;
- deductive methods and proof assistants: everything can be proved... potentially by hand
A combination of analysis

**k-induction**
- Model-checking method based on SMT solvers.
  - Inductively used.
  - Requires to manually strengthen the property to prove with numerous lemmas.

**Abstract interpretation**
- Infers numerical invariants.
  - High level of automaticity.
  - Result can be imprecise.

Hence the idea of combining them.
A combination of analysis

Temporal system and specification (Lustre)

$k$-induction engine

- Counterexample analysis
  - false counterexample
  - parameters (packing,...)

- Additional invariants

Abstract interpreter

SMT-AI: an Abstract Interpreter as Oracle for $k$-induction
Synchronous systems

**Definition**

We define a synchronous system as:

- a (potential) state space $S$;
- a set of inputs $E$;
- an initial state $I \in S$;
- a transition relation $T \subseteq S \times E \times S$.

Synchronous languages: Lustre, Scade, Mathlab Simulink
\[ k\text{-induction method} \]

**Definitions**

\[
\text{Base}(k) := I(s_0) \land \bigwedge_{i=0}^{k-1} T(s_i, e_i, s_{i+1}) \land \neg P(s_k)
\]

\[
\text{Step}(k) := \bigwedge_{i=0}^{k} P(s_i) \land \bigwedge_{i=0}^{k} T(s_i, e_i, s_{i+1}) \land \neg P(s_{k+1})
\]

**Algorithm**

- \( k := 0 \)
- if \( \text{Base}(k) \) is satisfiable, we have a counterexample
- if \( \text{Base}(k) \) and \( \text{Step}(k) \) are both unsatisfiable, we proved \( P \) for all reachable states
- else (\( \text{Base}(k) \) unsatisfiable and \( \text{Step}(k) \) satisfiable), increment \( k \) and continue

SMT-AI: an Abstract Interpreter as Oracle for \( k\)-induction
SAT/SMT solvers are used to decide satisfiability.

SMT solvers allow to handle efficiently infinite systems.

If the property is false, we get a counterexample (bounded model-checking).
Remarks

- SAT/SMT solvers are used to decide satisfiability.
- SMT solvers allow to handle efficiently infinite systems.
- If the property is false, we get a counterexample (bounded model-checking).
- Under some hypotheses, the method can be complete.
- But in practice, it is often needed to strengthen $P$ with various invariants to terminate with reasonable values of $k$. 
Abstract interpretation principle

Collecting semantic $R \subseteq S$ of a synchronous system can be defined as a least fixpoint ($lfp$):

$$R = lfp \left( \lambda X. I \cup \{ s' \mid \exists s \in X. \exists e. T(s, e, s') \} \right)$$

But this is usually not computable.

Abstract interpretation framework

- an abstract domain $S^\#$;
- a concretization function $\gamma : S^\# \rightarrow \mathcal{P}(S)$ ($s^\# \in S^\#$ is an overapproximation of $s \in S$ if $s \in \gamma(s^\#$));
- an abstract transition relation $T^\# : S^\# \rightarrow S^\#$ computable and sound with respect to $T : \{ s' \mid \exists s \in \gamma(s^\#). \exists e. T(s, e, s') \} \subseteq \gamma(T^\#(s^\#))$.

We can compute an overapproximation of $R$:

$$R^\# = \bigsqcup_n T^\#(I^\#)^n \quad (+ \text{widening})$$
Examples of abstract domains

- Intervals
- Congruences
- Polyhedra
- Octagons

But relational domains are expensive.
We compile Lustre code to an extension of SMT-lib:

- close from input language of SMT solvers;
- simplicity (no more clocks, purely functional,…);
- well known compilation process;
- functional languages unusual in abstract interpretation literature.
Abstract interpretation analysis

- Forward/backward analysis of SMT-lib formulas:
  - forward: we evaluate expressions;
  - backward: we constrain environments in which the expression can evaluate to expected value;
- Greatest fixpoint computation;
- “paper” proof of soundness and termination.
Implementation

- relational domains thanks to APRON library;
- trace partitioning;
- interface for multiple analysis reusing previous results:
  - with various relational packs;
  - under different contexts;
  - ...
- 10 k lines of Caml, under GPL;
Exemple

On following code:

```haskell
node collaboration(a : bool) returns (OK : bool)
var pre_x : int; pre_y : int; x : int; y : int; z : int; n : int;
let
    pre_x = fby(0, 1, x); pre_y = fby(0, 1, y); n = 21;
x = 0 → if pre_x < 2*n then pre_x + 1 else pre_x;
y = 2*n → if pre_y > 0 then pre_y − 1 else pre_y;
z = x*y;
OK = z ≤ n*n; tel
```

- **k-induction**: doesn’t terminate
- **abstract interpreter**: also unable to prove invariant \( OK = true \)
  but infers \( x + y = 42 \ldots \)
- \ldots which allows \( k \)-induction to prove \( OK = true \)
Actual collaboration between $k$-induction and abstract interpreter.

Improve our abstract interpreter (better reduced product, . . .).

Testing on more examples and industrial case studies.
Questions