Nested Interpolants

Matthias Heizmann  Jochen Hoenicke  Andreas Podelski

University of Freiburg, Germany

POPL 2010
Picture of a gordian knot.

Removed in online version because of copyrights.
Interpolant-based software model checking for recursive programs

- avoid construction of an abstract program
- Hoare logic $\leftrightarrow$ nested words
Thomas Ball, Sriram K. Rajamani:

*The SLAM project: debugging system software via static analysis.* (POPL 2002)

Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Grégoire Sutre

*Lazy abstraction.* (POPL 2002)

Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Kenneth L. McMillan

*Abstractions from proofs.* (POPL 2004)
Software model checking

Thomas Ball, Sriram K. Rajamani:

*The SLAM project: debugging system software via static analysis.* (POPL 2002)

Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Grégoire Sutre

*Lazy abstraction.* (POPL 2002)

Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Kenneth L. McMillan

*Abstractions from proofs.* (POPL 2004)

---

Bottleneck: Construction of abstract program
Recent approaches:
Avoid construction of abstract program

Franjo Ivancic, Ilya Shlyakhter, Aarti Gupta, Malay K. Ganai

*Model checking C programs using F-SOFT* (ICCD 2005)

Kenneth L. McMillan

*Lazy abstraction with interpolants* (CAV 2006)

Nels Beckman, Aditya V. Nori, Sriram K. Rajamani, Robert J. Simmons

*Proofs from tests* (ISSTA 2008)

Bhargav S. Gulavani, Supratik Chakraborty, Aditya V. Nori, Sriram K. Rajamani

*Automatically refining abstract interpretations* (TACAS 2008)
One idea:
Use interpolants to avoid construction of the abstract program

proof theorem checking

program abstract program invariant

interpolating theorem prover
One idea:
Use interpolants to avoid construction of the abstract program

interpolating theorem prover

Ranjit Jhala, Kenneth L. McMillan
A practical and complete approach to predicate refinement (TACAS 2006)

Kenneth L. McMillan
Lazy abstraction with interpolants (CAV 2006)
Quantified invariant generation using an interpolating saturation prover (TACAS 2008)

Open: Interpolants in interprocedural analysis
Recursive Programs?
Interprocedural static analysis - motivation

Recursive Programs?

Modularity!

```
procedure m(x) returns (res)
if x > 100
    res := x - 10
else
    x := x + 11
    call m(x)
    res := x + 10
assert (x <= 100 => res = 91)
return
```
Interprocedural static analysis - motivation

Recursive Programs?

Modularity!

Interprocedural analysis, a classical topic in programming languages

Micha Sharir, Amir Pnueli

Two approaches to interprocedural data flow analysis (1981)

Thomas W. Reps, Susan Horwitz, Shmuel Sagiv

Precise interprocedural dataflow analysis via graph reachability (POPL 1995)

Shaz Qadeer, Sriram K. Rajamani, Jakob Rehof

Summarizing procedures in concurrent programs (POPL 2004)
Given: \( A \Rightarrow B \)

Proof: \( A \Rightarrow B \)

Interpolation: \( A \Rightarrow I \Rightarrow B \).

... automatically generated by SMT solver (Craig interpolation)
Interpolants

Interpolant - for a proof

Given:
Proof \( A \Rightarrow B \)

Interpolation: \( A \Rightarrow I \Rightarrow B \).

... automatically generated by SMT solver (Craig interpolation)

Interpolant - for an execution traces

Given: Infeasible trace \( st_1 \ldots st_i \ldots st_{i+1} \ldots st_n \)

Interpolation: \( \text{post}(\text{true}, st_1 \ldots st_i) \subseteq I \subseteq \text{wp}(st_{i+1} \ldots st_n, \text{false}) \)

... can be new formula, not contained in program
Inductive interpolants

Construct sequence of interpolants $I_0 \ldots I_n$ inductively

$$\text{post}(I_i, st_i) \subseteq I_{i+1}$$

suitable Hoare annotation to prove infeasibility of program slice
Inductive interpolants

Construct sequence of interpolants $I_0 \ldots I_n$ inductively

$$\text{post}(I_i, st_i) \subseteq I_{i+1}$$

suitable Hoare annotation to prove infeasibility of program slice

What if execution trace contains procedure calls?
Interpolants for interprocedural analysis

What is an interpolant for an interprocedural execution?
Interpolants for interprocedural analysis

What is an interpolant for an interprocedural execution?

- state with a stack?
  ~ locality of interpolant is lost
Interpolants for interprocedural analysis

What is an interpolant for an interprocedural execution?

- state with a stack?
  - $\Rightarrow$ locality of interpolant is lost

- only local valuations?
  - $\Rightarrow$ call/return dependency lost,
  - $\Rightarrow$ sequence of interpolants is not a proof
Our gordian knot

Picture of a gordian knot.

Removed in online version because of copyrights.

How can we keep track of the call/return dependency in a sequence of states without a stack?
Picture of a gordian knot.

Removed in online version because of copyrights.
Nested words

Idea: Add call/return dependency explicitly to the word

\[\text{Rajeev Alur}
\]

*Marrying words and trees* (PODS 2007)

*Rajeev Alur, P. Madhusudan*

*Adding nesting structure to words* (DLT 2006, J. ACM 56(3) 2009)

*Rajeev Alur, Swarat Chaudhuri*

*Temporal reasoning for procedural programs* (VMCAI 2010)
Nested words

Idea: Add call/return dependency explicitly to the word

Rajeev Alur

Marrying words and trees (PODS 2007)

Rajeev Alur, P. Madhusudan

Adding nesting structure to words (DLT 2006, J. ACM 56(3) 2009)

Rajeev Alur, Swarat Chaudhuri

Temporal reasoning for procedural programs (VMCAI 2010)
What is a sequence of interpolants for an interprocedural execution?

Idea: Define sequence interpolants with respect to nested trace.
Nested interpolants

What is a sequence of interpolants for an interprocedural execution?

Idea: Define sequence interpolants with respect to nested trace

\[ \text{post} (L_i, L_k, \text{return}) \subseteq L_{i+1} \]
Control flow as nested word automata

procedure m(x) returns (res)

\[ \ell_0 : \text{if } x > 100 \]

\[ \ell_1 : \text{res} := x - 10 \]

\[ \ell_2 : x_m := x + 11 \]

\[ \ell_3 : \text{call m} \]

\[ \ell_4 : x_m := \text{res}_m \]

\[ \ell_5 : \text{call m} \]

\[ \ell_6 : \text{res} := \text{res}_m \]

\[ \ell_7 : \text{assert } (x \leq 101 \rightarrow \text{res} = 91) \]

return m

McCarthy 91 function

nested word automaton
Floyd-Hoare proof as nested word automata

procedure m(x) returns (res)
   {⊤}
   ℓ₀: if x>100
      {x ≥ 101}
   ℓ₁: res:=x-10
      else
      {x ≤ 100}
   ℓ₂: xₘ := x+11
      {xₘ ≤ 111}
   ℓ₃: call m
      {resₘ ≤ 101}
   ℓ₄: xₘ := resₘ
      {xₘ ≤ 101}
   ℓ₅: call m
      {resₘ = 91}
   ℓ₆: res := resₘ
      {res = 91 ∨ (x ≥ 101 ∧ res = x – 10)}
   ℓ₇: assert (x≤101 -> res=91)
       return m

McCarthy 91 function

nested word automaton
procedure m(x) returns (res)
   {⊤}
   ℓ₀: if x>100
   {x ≥ 101}
   ℓ₁: res:=x-10
   else
   {x ≤ 100}
   ℓ₂: xₘ := x+11
   {xₘ ≤ 111}
   ℓ₃: call m
   {resₘ ≤ 101}
   ℓ₄: xₘ := resₘ
   {xₘ ≤ 101}
   ℓ₅: call m
   {resₘ = 91}
   ℓ₆: res := resₘ
   {res = 91 ∨ (x ≥ 101 ∧ res = x – 10)}
   ℓ₇: assert (x<=101 -> res=91)
   return m

McCarthy 91 function

nested word automaton

e.g. post(x ≤ 100, xₘ:=x+11) ⊆ xₘ ≤ 111
Constructing a proof of correctness

Compute sequence of nested interpolants

\[ \varphi_0 : x^{-1} \leq 100 \]
\[ \varphi_1 : x_m^1 = x^{-1} + 11 \]
\[ \varphi_2 : x^2 = x_m^1 \]
\[ \varphi_3 : x^2 > 100 \]
\[ \varphi_4 : res^4 = x^2 - 10 \]
\[ \varphi_5 : res_m^5 = res^4 \]
\[ \varphi_6 : x_m^6 = res_m^5 \]
\[ \varphi_7 : x^7 = x_m^6 \]
\[ \varphi_8 : x^7 > 100 \]
\[ \varphi_9 : res^9 = x^7 - 10 \]
\[ \varphi_{10} : res_m^{10} = res^9 \]
\[ \varphi_{11} : res^{11} = res_m^{10} \]
\[ \varphi_{12} : x^{-1} \leq 100 \land res^{11} \neq 91 \]

infeasible nested trace \( \pi \)

SSA of \( \pi \)

sequence of interpolants for \( \pi \)
Constructing a proof of correctness

Nested interpolant automaton

sequence of interpolants for \( \pi \)

nested interpolant automaton
Constructing a proof of correctness

Nested interpolant automaton

sequence of interpolants for \( \pi \)

nested interpolant automaton
Constructing a proof of correctness

CEGAR

recursive program $\mathcal{P}$

start with $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) \supseteq \mathcal{L}(\mathcal{A}_\Sigma)$

return refined abstraction $\mathcal{A} := \mathcal{A} \cap \overline{\mathcal{A}_\mathcal{I}}$

where

$\mathcal{A}_\mathcal{I}$ is a nested interpolant automaton

such that

$\pi \in \mathcal{L}(\mathcal{A}_\mathcal{I})$

$L(\mathcal{A}) \cap L(\mathcal{A}_P) = \emptyset$ ?

$\pi \in \mathcal{L}(\mathcal{A}_\Sigma)$ ?

no

no

return nested error trace $\pi$

such that

$\pi \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_P)$

no

yes

yes

$\mathcal{P}$ is correct

$\mathcal{P}$ is incorrect
Conclusion

Interpolant-based software model checking for recursive programs

- avoid construction of an abstract program
- Hoare logic $\leftrightarrow$ nested words