

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.

Result

Interpolant-based software model checking for recursive programs

- ▶ avoid construction of an abstract program
- ▶ Hoare logic \rightsquigarrow nested words

Software model checking

Thomas Ball, Sriram K. Rajamani:

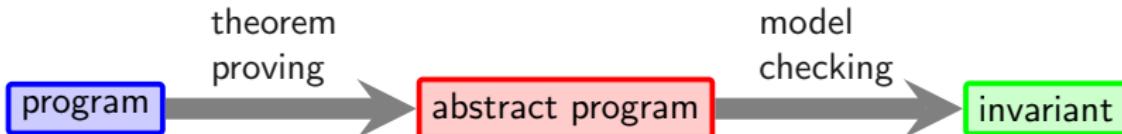
The SLAM project: debugging system software via static analysis. (POPL 2002)

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)



Software model checking

Thomas Ball, Sriram K. Rajamani:

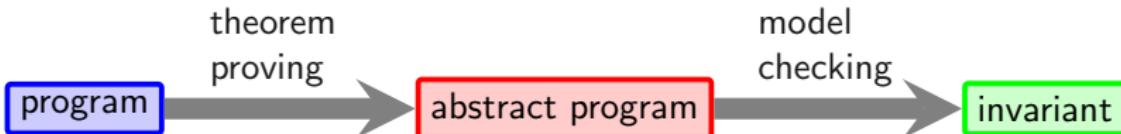
The SLAM project: debugging system software via static analysis. (POPL 2002)

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

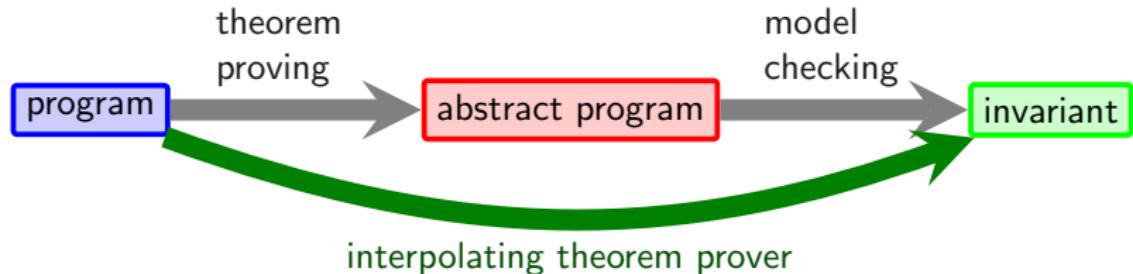
Proofs from tests (ISSTA 2008)

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

Automatically refining abstract interpretations (TACAS 2008)

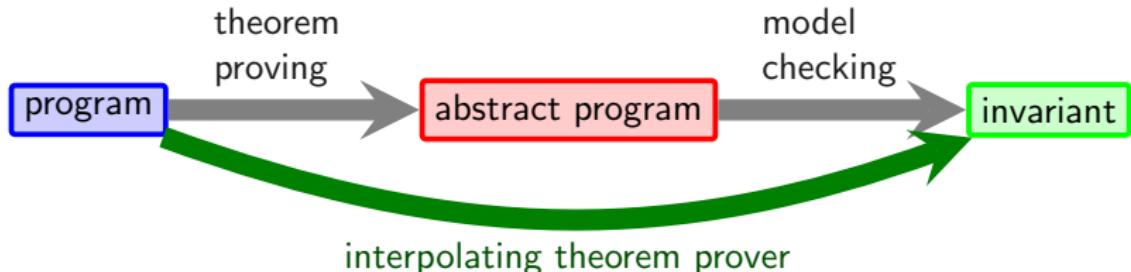
One idea:

Use interpolants to avoid construction of the abstract program



One idea:

Use interpolants to avoid construction of the abstract program



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

Interprocedural static analysis - motivation

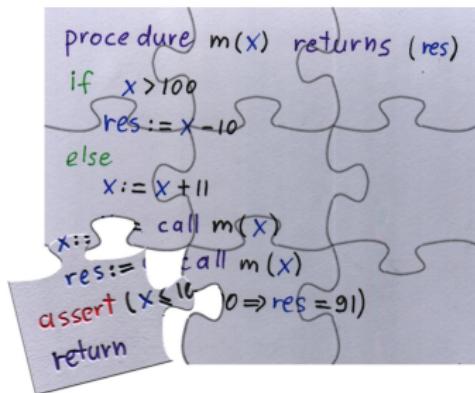
Recursive Programs?



Interprocedural static analysis - motivation

Modularity!

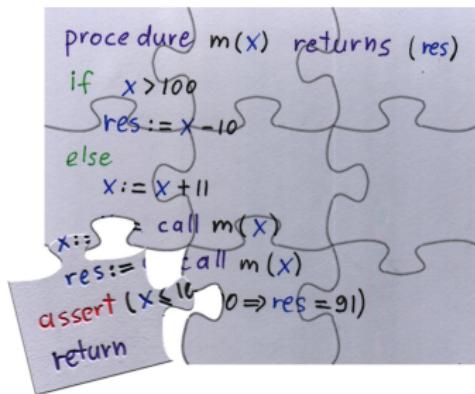
Recursive Programs?



Interprocedural static analysis - motivation

Modularity!

Recursive Programs?



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)

Interpolants

Interpolant - for a proof

Given:

Proof $\boxed{A} \Rightarrow \boxed{B}$

Interpolation:

$\boxed{A} \Rightarrow \boxed{I} \Rightarrow \boxed{B}$.

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

Interpolants

Interpolant - for a proof

Given:

Proof $\boxed{A} \Rightarrow \boxed{B}$

Interpolation:

$\boxed{A} \Rightarrow \boxed{I} \Rightarrow \boxed{B}$.

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

Interpolant - for an execution traces

Given:

Infeasible trace $\boxed{st_1} \dots \boxed{st_i} \boxed{st_{i+1}} \dots \boxed{st_n}$

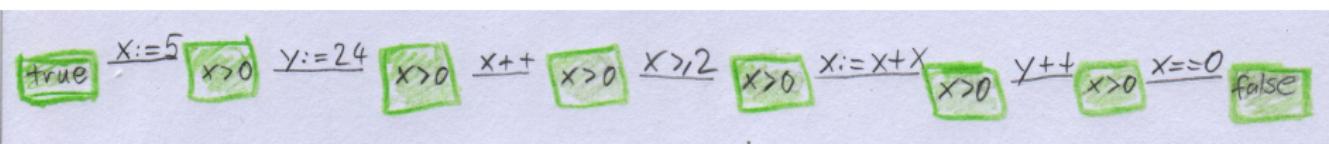
Interpolation: $post(\boxed{\text{true}}, \boxed{st_1} \dots \boxed{st_i}) \subseteq \boxed{I} \subseteq wp(\boxed{st_{i+1}} \dots \boxed{st_n}, \boxed{\text{false}})$

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

Inductive interpolants

Construct sequence of interpolants $I_0 \dots I_n$ inductively

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

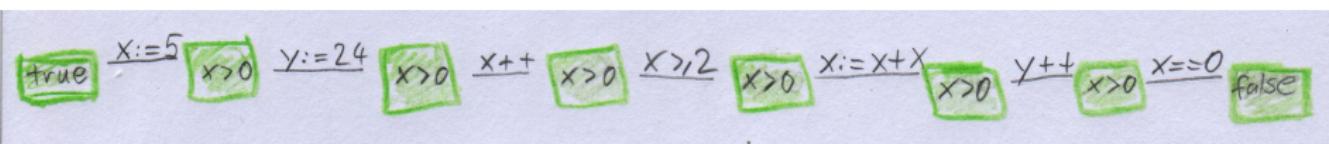


suitable **Hoare annotation** to prove infeasibility of program slice

Inductive interpolants

Construct sequence of interpolants $I_0 \dots I_n$ inductively

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

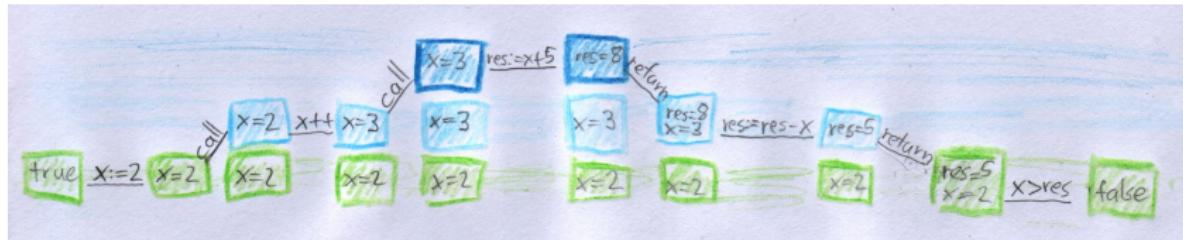


suitable **Hoare annotation** to prove infeasibility of program slice

What if execution trace contains procedure calls?

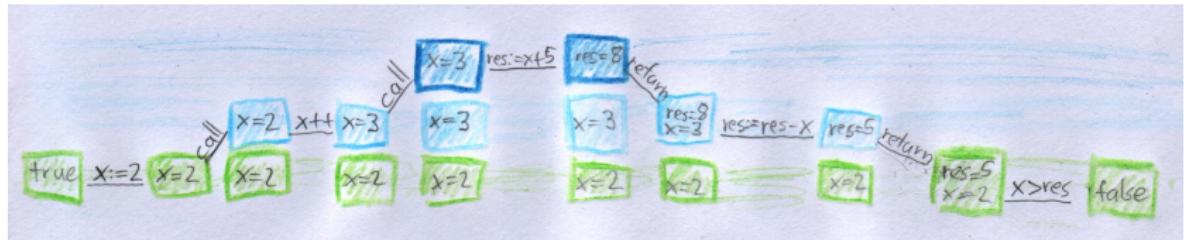
Interpolants for interprocedural analysis

What is an interpolant for an interprocedural execution?



Interpolants for interprocedural analysis

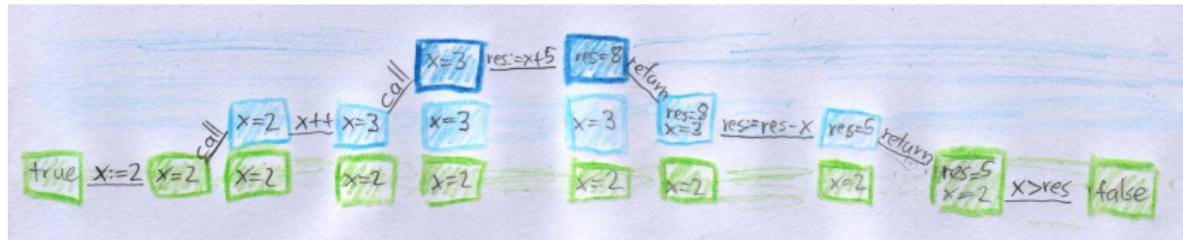
What is an interpolant for an interprocedural execution?



- ▶ 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?
 - ~~ locality of interpolant is lost
- ▶ only local valuations?
 - ~~ call/return dependency lost,
 - ~~ sequence of interpolants is not a proof

Our gordian knot

Picture of a
gordian knot.

Removed in online version
because of copyrights.

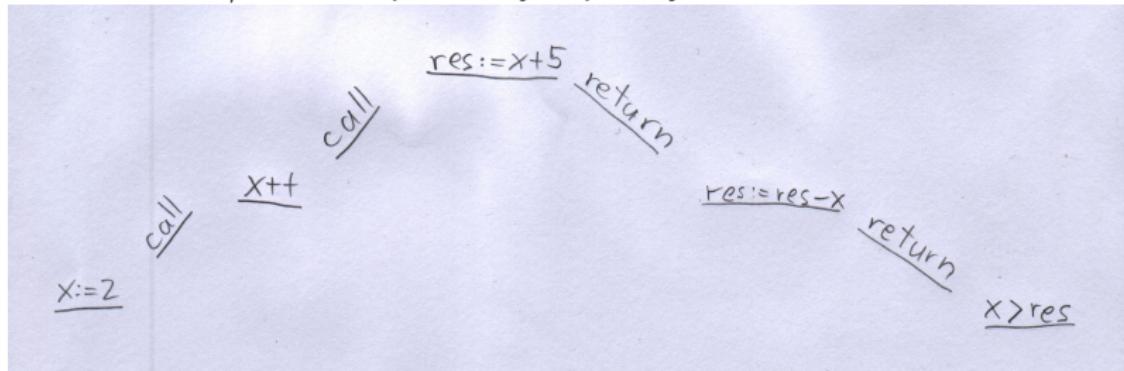
How can we keep track of the call/return dependency in a sequence of states without a stack?

Picture of a
gordian knot.

Removed in online version
because of copyrights.

Nested words

Idea: Add call/return dependency explicitly to the word



Rajeev Alur

Marrying words and trees (PODS 2007)

Rajeev Alur, P. Madhusudan

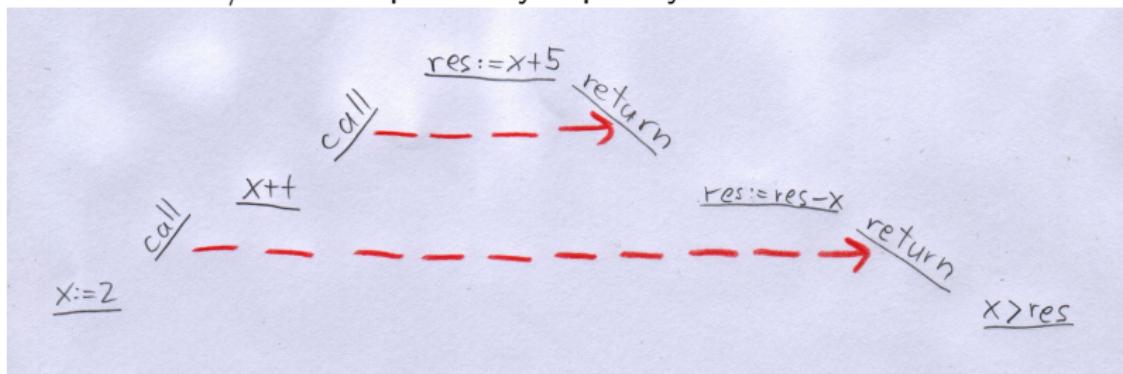
Adding nesting structure to words (DLT 2006, J. ACM 56(3) 2009)

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)

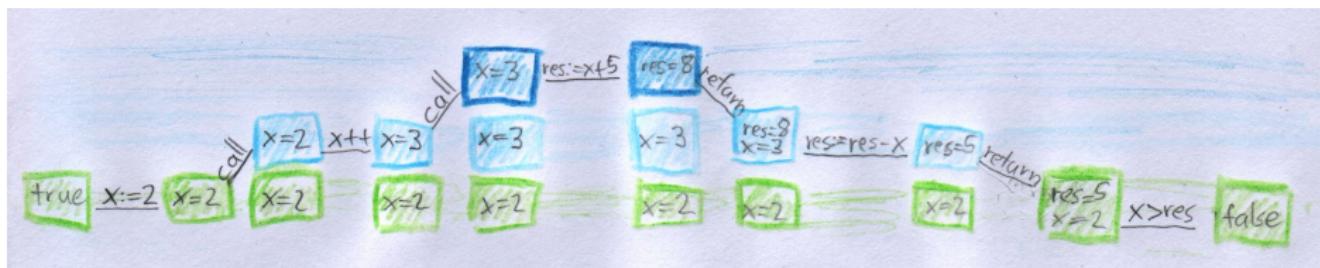
Rajeev Alur, Swarat Chaudhuri

Temporal reasoning for procedural programs (VMCAI 2010)

Nested interpolants

What is a sequence of interpolants for an interprocedural execution?

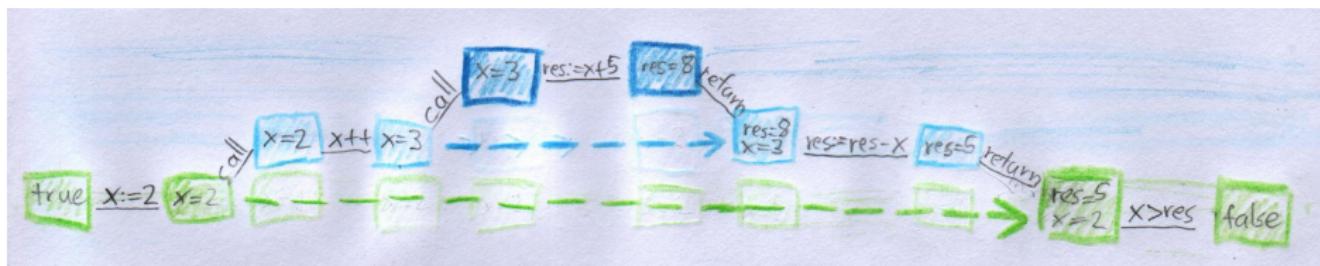
Idea: Define sequence interpolants with respect to nested trace



Nested interpolants

What is a sequence of interpolants for an interprocedural execution?

Idea: Define sequence interpolants with respect to nested trace



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

Control flow as nested word automata

```
procedure m(x) returns (res)

    l0: if x>100

        l1:   res:=x-10
        else

            l2:   xm := x+11

            l3:   call m

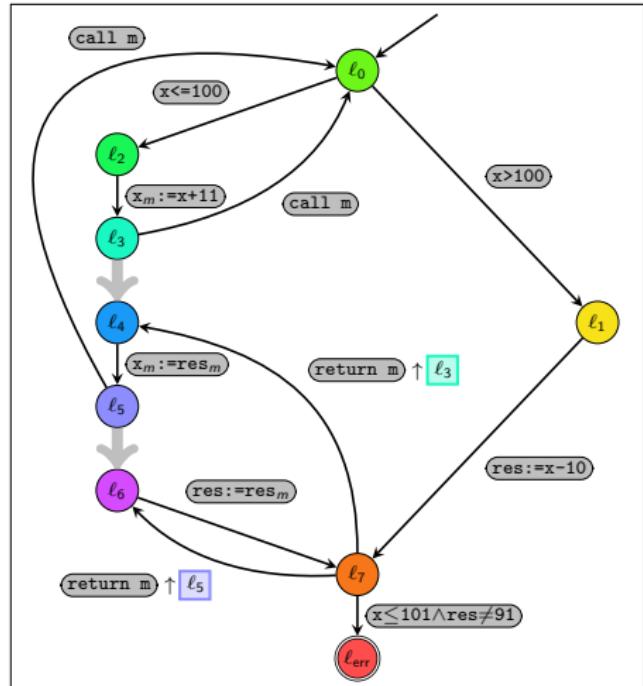
            l4:   xm := resm

            l5:   call m

            l6:   res := resm

            l7: assert (x<=101 -> res=91)
            return m
```

McCarthy 91 function



nested word automaton

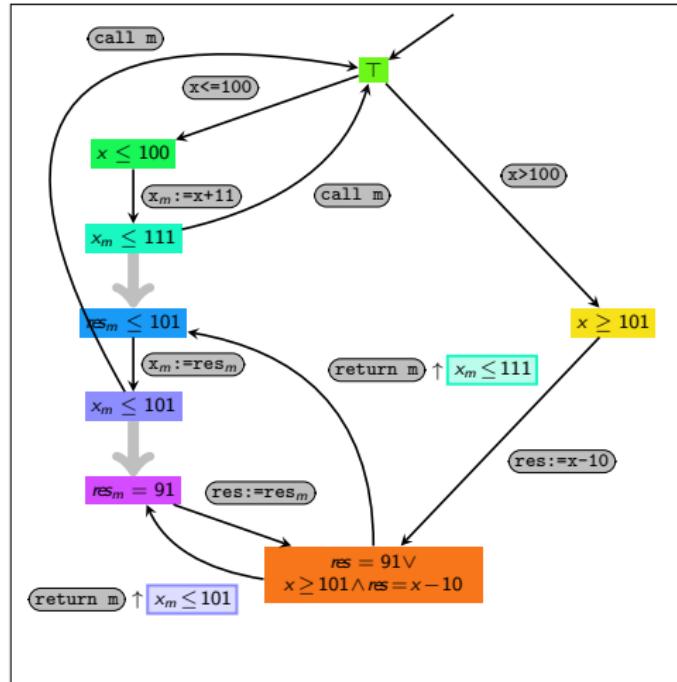
Floyd-Hoare proof as nested word automata

```

procedure m(x) returns (res)
    {T}
    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

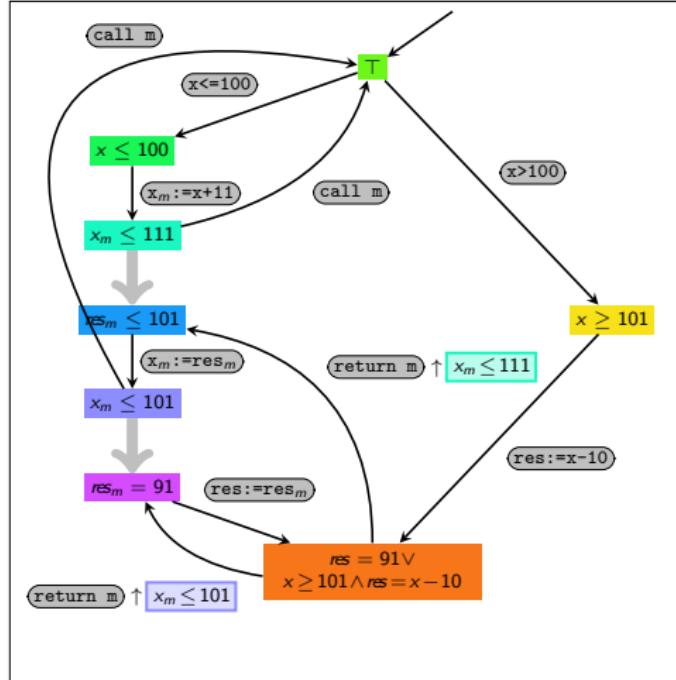
Floyd-Hoare proof as nested word automata

```

procedure m(x) returns (res)
    {T}
    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

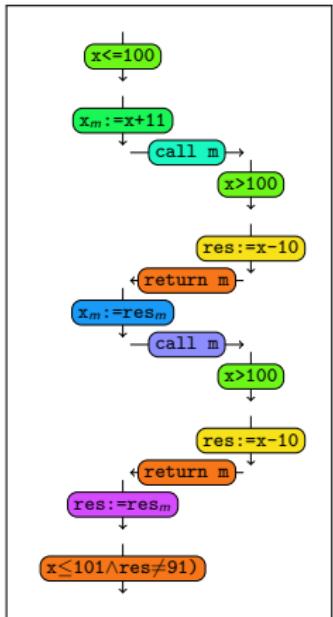


nested word automaton

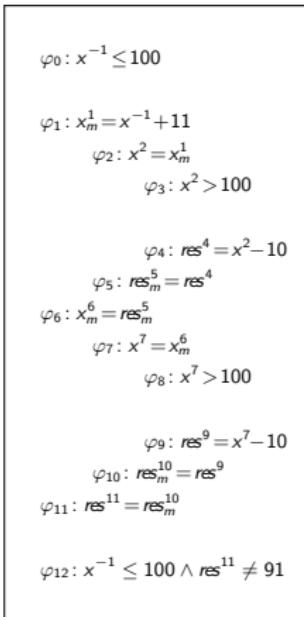
$$\text{e.g. } post(x \leq 100, (x_m := x + 11)) \subseteq x_m \leq 111$$

Constructing a proof of correctness

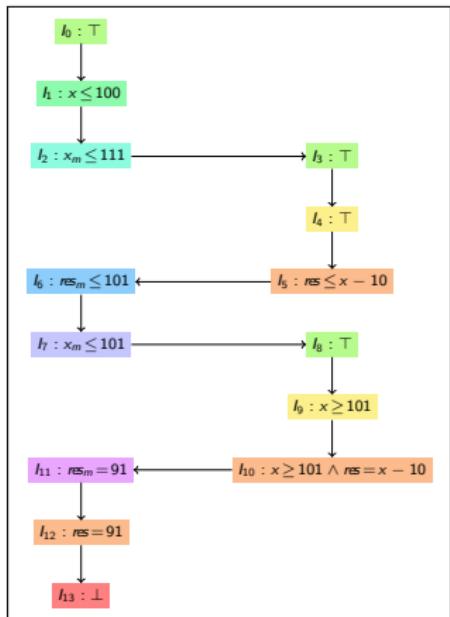
Compute sequence of nested interpolants



infeasible nested trace π



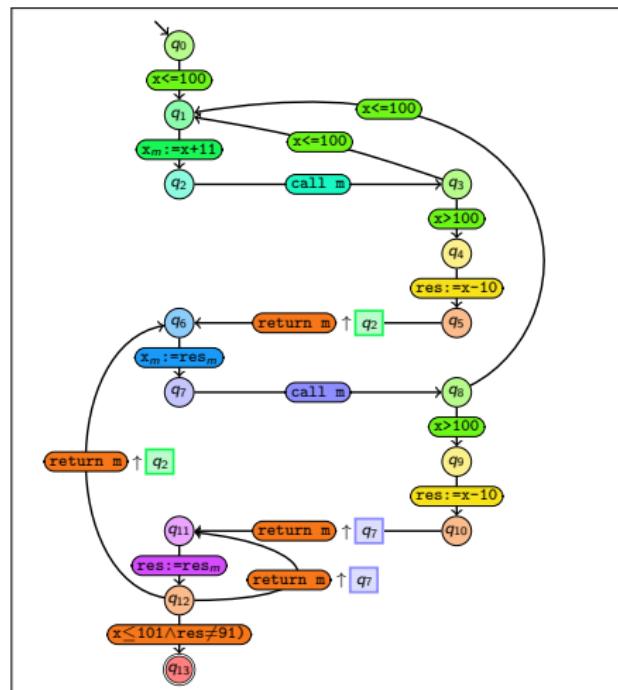
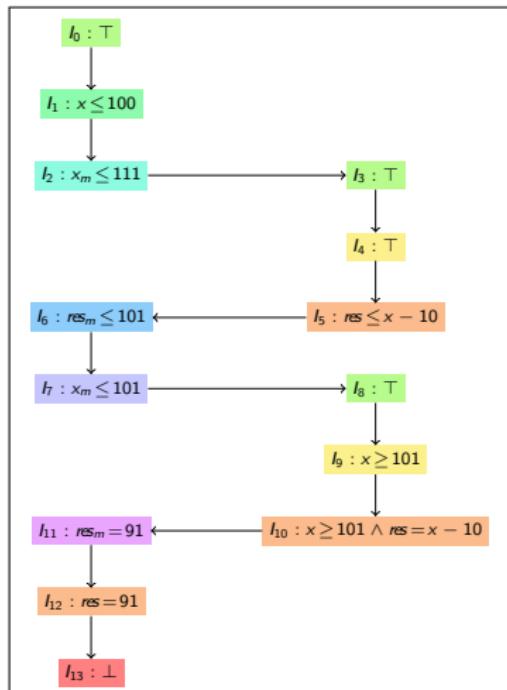
SSA of π



sequence of interpolants for π

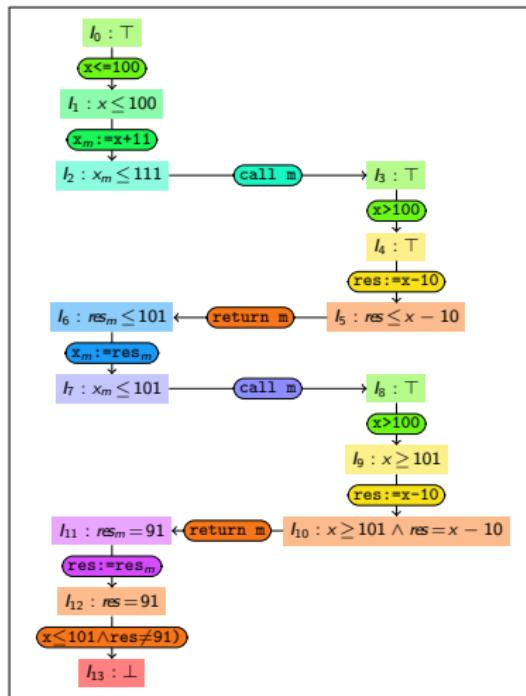
Constructing a proof of correctness

Nested interpolant automaton

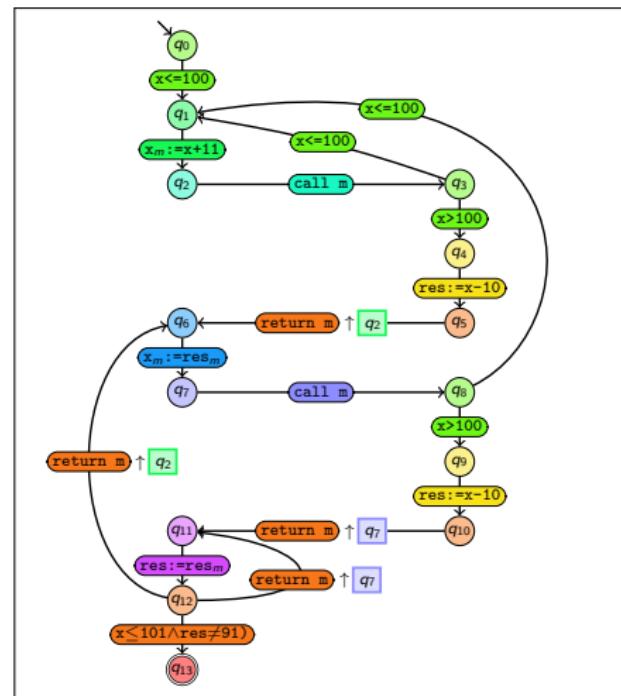


Constructing a proof of correctness

Nested interpolant automaton



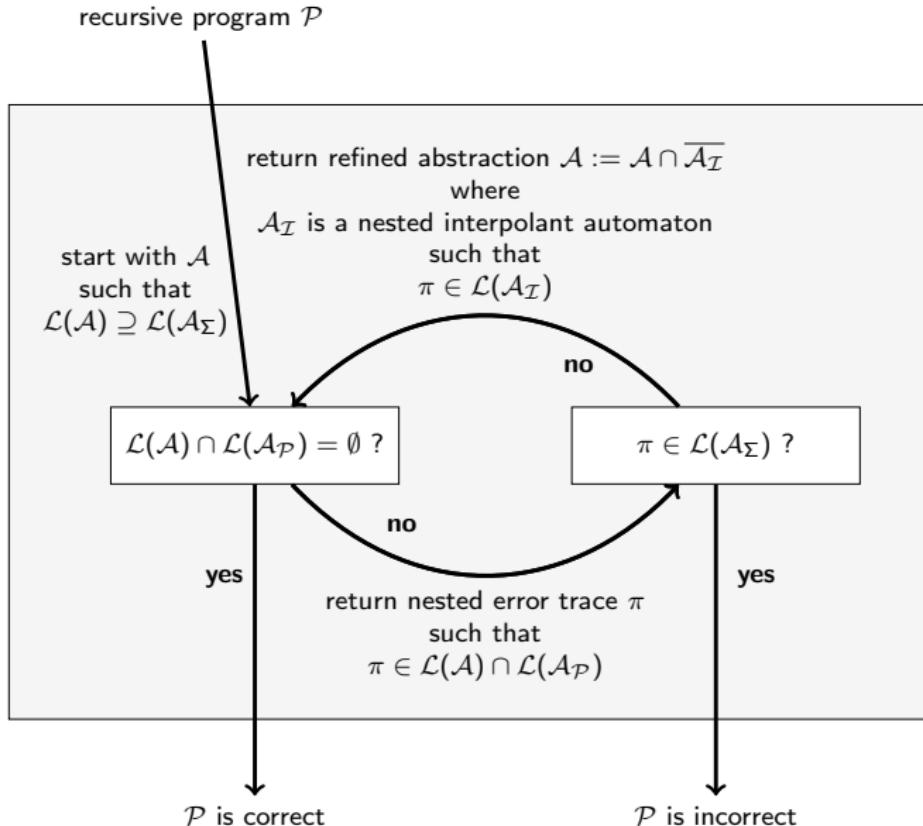
sequence of interpolants for π



nested interpolant automaton

Constructing a proof of correctness

CEGAR



Conclusion

Interpolant-based software model checking
for recursive programs

- ▶ avoid construction of an abstract program
- ▶ Hoare logic \rightsquigarrow nested words