Reach: Formal Verification in Go

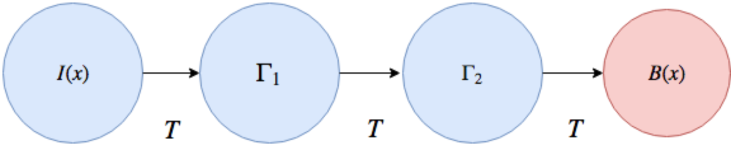
Scott Cotton

Independent

December 29, 2018

Reach: What's the Problem?

$$(x, I, T, B)$$



- Infinite traces.
- Symbolic, ergo big state space $2^{|x|}$.
- PSPACE complete.
- Fundamental and inescapable.

Core Approaches ($\sim$Orthogonal)

Sim

- Main tool for exploration.
- Non-exhaustive.
- Finds most bugs that BMC finds.
- *Very* fast.
- Very well understood.

BMC

- Exhaustive for finite prefix.
- Long prefix $\implies$ high confidence.
- Can produce counterexamples.
- Understood.
- Often very fast.

Proofs

- Complete safety.
- Certificates.
- Not well understood.
- Many methods (IMC, TI, ...)
- Incremental induction!

Reach: the tool

```
⌊⇨reach
Reach is a finite state reachability tool for binary systems.

usage: reach [gopts] <command> [args]

available commands:
        iic     iic is an incremental inductive checker.
        bmc     bmc performs SAT based bounded model checking.
        sim     sim simulates aiger.
        ck      ck checks traces and inductive invariants.
        stim    stim outputs an aiger stimulus from an output directory.
        aag     aag outputs an ascii aiger of the Reach internal aig.
        aig     aig outputs an binary aiger of the Reach internal aig.
        info    info provides summary information about an aiger or output.

global options:
  -cpuprof string
        file to output cpu profile

For help on a command, try "reach <cmd> -h".
```

Focus: Proofs *via* SAT solving

- Proofs are only true solution to the reachability problem.
- IIC: incremental inductive checking.
- Existential queries $\implies$ no space brick wall.
- Use Gini `http://www.github.com/irifrance/gini`.
- Exploit incremental scoping, activation literals.

Proofs: Incremental Induction

Good ideas are contagious.

- Aaron Bradley (2007-present)
- Alan Mischenko, Niklas Een ABC PDR (2010-present)
- Nikolai Bjorner Z3 PDR with Horn clauses (?)
- . . .

Incremental Induction

Induction

Find P:

$$\begin{aligned} I &\implies P \\ P &\implies \neg B \\ P \wedge T &\implies P' \end{aligned}$$

How to find P?
Temporal induction: use the post-image of a BMC prefix.
Interpolation: use interpolant of a BMC prefix with $\neg B$.

Incremental

Given A, find P:

$$\begin{aligned} I &\implies P \\ P &\implies \neg B \\ A \wedge P \wedge T &\implies P' \end{aligned}$$

Then update $A \leftarrow A \wedge P$.
In PDR/IC3, A is CNF, P is a clause which blocks states than can reach B.

Incremental Induction – CNFs

To check reachability with incremental induction, we maintain a levelled CNF

$$\Gamma \doteq \bigcup_i \Gamma_i, i \in [1 \dots K]$$

with

$$\Gamma_i \equiv \bigwedge_j \bigvee M_j, M_j \subseteq \{m | m \equiv v \text{ or } m \equiv \neg v, v \in x\}$$

- Each Γ_i represents an *overapproximation* of the reachable states from Γ_{i-1} (or from I for Γ_1).
- Syntactically, as a set of clauses, $\Gamma_i \supseteq \Gamma_{i+1}$
- Semantically, $\Gamma_i \subseteq \Gamma_{i+1}$
- Let $\lambda_i \doteq \Gamma_i \setminus (\bigcup_{j>i} \Gamma_j)$. These are clauses local to level i.
- Inversely, we can define Γ_i in terms of λ_i:

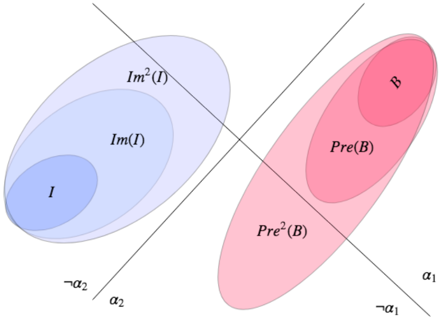
$$\Gamma_i \doteq \bigcup_{j \geq i} \lambda_j$$

Incremental Induction – Algorithm

- IC3 and PDR are 2-phase algorithms.
- Blocking phase: find inductive clauses incrementally from bad state backward reachable states.
- Propagation phase: identify when clauses become relatively inductive.

```
// check I => not(B), I and T => not(B')
initialize()
k := 1
for {
        if not block(k)  {
                return Reachable
        }
        if propagate(k) {
                return Unreachable
        }
        k++
}
```

Incremental Induction: Blocking



CNF		clause	enabling condition
λ_1	$\leftarrow$	$\top$	
λ_1	$\leftarrow$	$\lambda_1 \wedge \neg\alpha_1$	$\mathrm{sat}(\alpha_1 \wedge T \wedge B'), \mathrm{unsat}(I \wedge T \wedge \alpha_1')$
λ_2	$\leftarrow$	$\top$	$\mathrm{unsat}(\Gamma_1 \wedge T \wedge B')$
λ_2	$\leftarrow$	$\lambda_2 \wedge \neg\alpha_2$	$\mathrm{sat}(\alpha_2 \wedge T \wedge (B' \vee \alpha_1')), \mathrm{unsat}(\Gamma_1 \wedge T \wedge \alpha_2')$

Blocking: Lifting and Generalization

How do we find α_i?

1. Maintain a tree of proof obligations.
2. Each proof obligation is a term $o \equiv \bigwedge\{a_0, a_1, \ldots\}$.
3. Find σ s.t. σ can reach a bad state b from some Γ_j (SAT problems).
4. *Lift* σ to a small term o such every extension of o can transition to b, independent of Γ_j.
5. Generalize o by finding a small subclause which makes the query

   $$\Gamma_j \wedge \neg o \wedge T \wedge o'$$

   unsat.

α_i is then the result of generalization.

Incremental Induction: Propagation

Given K levels $[1 \dots K]$, and a set of clauses Γ_k for each level k such that

$$
\begin{aligned}
\Gamma_k &\implies \Gamma_{k+1}, && k \in [1 \dots K) \\
I &\implies \Gamma_k, && k \in [1 \dots K] \\
\Gamma_k \wedge T &\implies \Gamma_{k+1}, && k \in [1 \dots K)
\end{aligned}
$$

Find the closure of the following consecution rule (CR) ($k \in [1 \dots K]$)

$$
\begin{aligned}
\text{if } c \in \Gamma_k, \Gamma_k \wedge T &\implies c' \\
\text{then } c \in \Gamma_{k+1} &
\end{aligned}
$$

Incremental Induction: Termination

Suppose we propagate with $\Gamma_i, i \in [1 \ldots K]$, and the result is that $\Gamma_K = \Gamma_{k+1}$.
Then Γ_K is an inductive invariant proving unreachability.

$$
\begin{aligned}
I &\implies \Gamma_K \\
\Gamma_K &\implies \neg B \\
\Gamma_K \wedge T &\implies \Gamma'_K
\end{aligned}
$$

Incremental Induction: The Reach Way

- Justifying Proof Obligations.
- Sifting Generalization.
- Unified Queueing.
  - Consecutive Sifting
  - Obligation Filtering

Justifying Proof Obligations

Justification is a method which given

1. A circuit $C : x \to y$; and
2. An assignment σ to x; and
3. A valuation ν of the outputs y under σ.

Gives a minimal assignment $\mathcal{J}(x)$ such that x evaluates to y.
Example:

$$\begin{aligned} C &= & y \iff x_0 \wedge \neg x_1 \\ \sigma &= & \{x_0, x_1\} \\ \nu &= & \{\neg y\} \end{aligned}$$

Then $\mathcal{J}(x) = \{x_1\}$

Justifying Proof Obligations

Justification properties.

1. Justification has simple linear time recursive algorithm.
2. Justification can break ties by heuristics which tend not to select certain inputs, such as state variables.
3. Justification is independent of any Γ_i component of the query.
4. Alternative for quadratic worst case ternary simulation in ABC PDR

Sifting Generalization

- SAT solvers ususally can provide a set of *failed literals* for an unsat problem under a set of assumptions $a_0, a_1, \ldots, a_m$, where the assumptions are just assignments to some of the variables.
- *Sifting* is a term for strengthening clauses in a sat problem.
- If φ is a sat problem, and $c \equiv a_0 \lor a_1 \lor \ldots \lor a_n$ is a clause in φ, then we know that $\varphi \land \neg c$ is unsat.
- We can test $\varphi \land \neg c$, and if it is unsat, then the solver will hand us a subset $\hat{c} \subseteq \{a | a \in c\}$ such that $\varphi \land \hat{c}$ is unsat.

Sifting Generalization

Sifting is the process of repeating the following

$$\begin{aligned} \hat{c} &\leftarrow \text{why}(\varphi \wedge \neg\text{shuffle}(c)) \\ c &\leftarrow \hat{c} \end{aligned}$$

until $|c| = |\hat{c}|$ for some number of iterations, or similar termination condition.

Sifting

- Is often faster than testing whether $\varphi \wedge \bigvee(c \setminus \{a\})$ is unsat.
- Is often effective since re-ording the assumptions can yield different results.
- Does not guarantee as strong a result as removing literals.
- Much better time/effectiveness ratio.

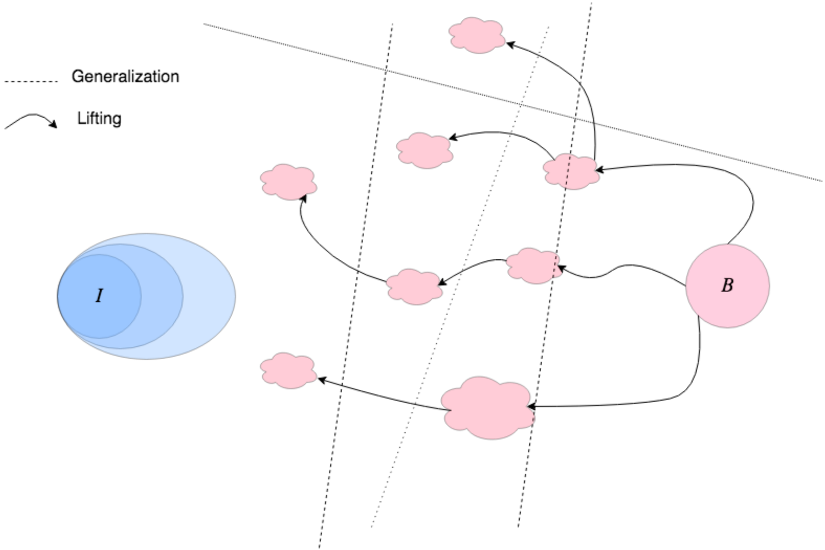
Consecutive Sifting

Consecutive Sifting is the process of applying sifting to clauses in Γ_i under consecution

$$\Gamma_{i-1} \wedge T \wedge \neg c', c \in \Gamma_i$$

- Unlike sifting, consecutive sifting strengthens Γ_i semantically.
- If a clause c can be properly strengthened by consecutive sifting, the result can *feedback*.
- Consecutive sifting is an alternative to and can augment generalization.
- Consecutive sifting can be interleaved with blocking.

Filtering Proof Obligations



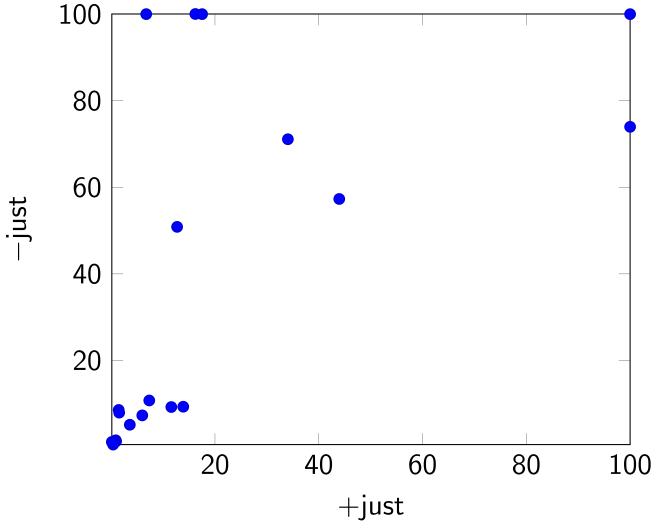
Filtering Proof Obligations

- Generalization and consecutive sifting makes some SAT queries irrelevant.
- Irrelevant queries lead to redundant CNFs, extra work.
- Reach: keep only obligations which aren't blocked.
- Uses a fast SAT subsumption algorithm relating proof obligations to clauses.

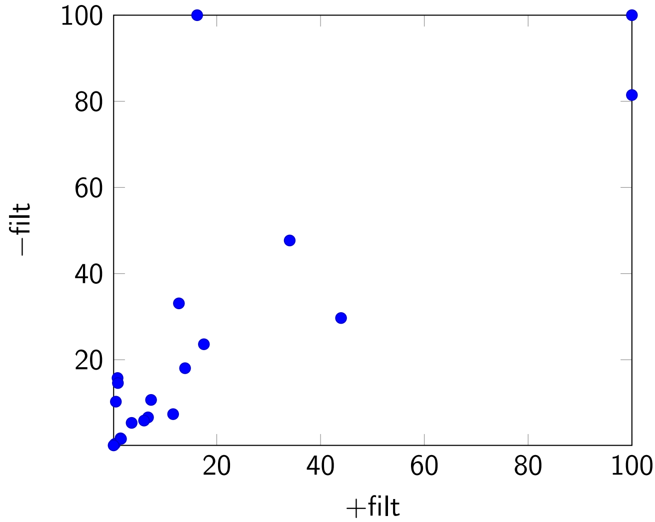
Reach Sneak Peak

- Reach is under development.
- Still some corner case bugs to work out (aiger format).
- Still often not competitive with ABC/PDR.
- Bmc, simulation, result checking stable and fast enough.
- IIC functional and much faster than baseline IC3.
- Developed with TIP and HWMCC16 benchmarks.
- Lots of easy problems. Not enough medium difficulty problems.

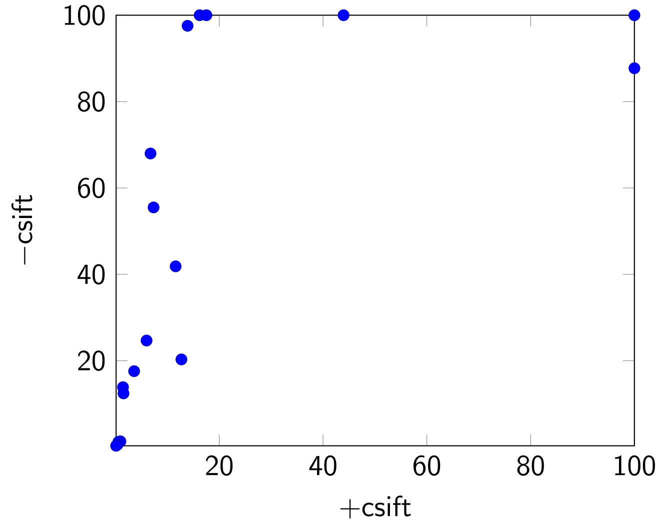
Justification Benchmarks



Proof Obligation Filtering Benchmarks



Consecutive Sifting Benchmarks



Reach Conclusions

- A new tool and library for symbolic finite state reachability.
- Written in Go.
- Uses Gini extensively.
- Implements some new effective ideas.

Thanks

Thanks for your interest in this work.