The Synergy of Precise and Fast Abstractions for Program Verification

Natasha Sharygina[1,3], Stefano Tonetta[2] and Aliaksei Tsitovich[1]

[1]University of Lugano, Switzerland

[2]Fondazione Bruno Kessler, Trento, Italy

[3]School of Computer Science, Carnegie Mellon University, USA

March 9, 2009

Talk Outline

- What fast and precise abstractions mean in context of CEGAR-based model checking
- Synergy algorithm – localization of the precise abstraction
- Implementation and Experiments

Software Model

Definition

A Transition System (TS) is a tuple $M = \langle V, I, T \rangle$, where

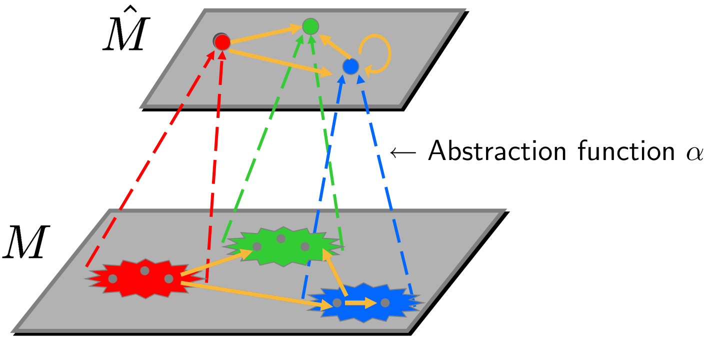
- V is a set of variables;
- $I(V)$ is a formula that represents the initial states;
- $T(V, V')$ is a formula that represents the transitions.

Execution of a program — path in M that starts at an initial state.

Counterexample — execution that reaches some “bad” state.

Abstraction

Abstraction is a key technique to scalable program verification.



The opposite to abstraction is called **concretization** γ

Program Abstraction

Definition

$\hat{M}$ is **abstraction** of M with an **abstraction function** α iff:

- every initial state of M corresponds to an initial state of $\hat{M}$ (if $s \models I(V)$ then $\exists \hat{s}. \hat{s} \models \hat{I}(\hat{V}) \wedge \alpha(s) = \hat{s}$);
- every transition of M corresponds to a transition of $\hat{M}$ ($\forall t \in T$ if $\exists t(s, s')$ then $\exists \hat{s}, \hat{s'}.\ \alpha(s) = \hat{s} \wedge \alpha(s') = \hat{s'} \wedge \hat{t}(\hat{s}, \hat{s'})$).

We also say that M **refines** $\hat{M}$ ($M \preceq \hat{M}$).

Concretization is the mapping in the opposite direction $\gamma : \hat{M} \rightarrow M$. It is not always possible, i.e., abstraction might introduce **spurious behaviors**, which have no mappings to M.

Spurious behaviors

Spurious transition — a transition $\hat{t}$ in $\hat{M}$, which has no concretization in M $(\forall t \in \gamma(\hat{t}) : \hat{t} \models \hat{T} \wedge t \not\models T)$.

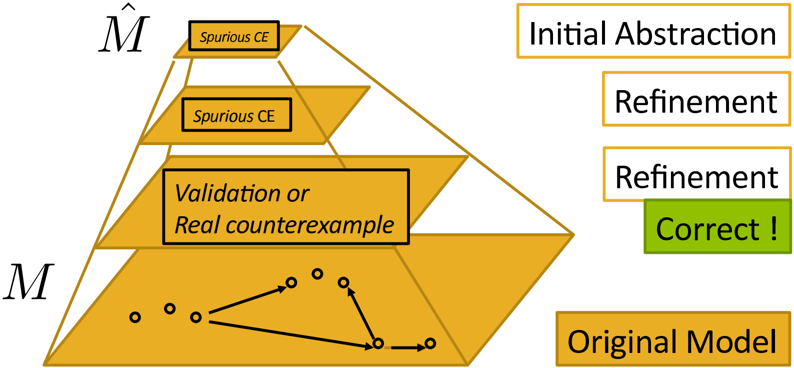
Spurious path — a sequence of spurious transitions, $\hat{\pi}$ in $\hat{M}$, which has no concretization in M $(\forall \pi \in \gamma(\hat{\pi}) : \hat{\pi} \models \hat{T} \wedge \pi \not\models T)$.

Spurious counterexample — a counterexample in $\hat{M}$, whose path π has no concretization in M.

Real counterexample — a counterexample in $\hat{M}$, which has concretization in M.

Refinement — an update of $\hat{M}$, which removes some spurious behavior.

Abstraction & Refinement



Counterexample-driven Abstraction Refinement (CEGAR)

Predicate abstraction by Graf/Saidi 97[1]

Idea:

Use predicates on data $p_1(s); ...; p_n(s)$ to cluster states of M

Abstraction function:
$\alpha(s) := p_1(s); ...; p_n(s)$

Abstract transition relation $\hat{T}$ can be:

- minimal (**precise abstraction**)
- over-approximated (**fast abstraction**)

[1] commonly used in program verification due to its full automation

Precise Abstraction[2]

$$\hat{T}_\alpha(\hat{V}, \hat{V}') = \exists V, V' : T(V, V') \wedge \alpha(V) = \hat{V} \wedge \alpha(V') = \hat{V}'$$

- Minimal number of abstract transitions (no spurious transitions)
- Adding new predicates is enough to refine spurious path
- Different computational engines: theorem provers, SAT/SMT-solvers, mixed BDD/SMT solvers
- But... Very slow computation (exponential in the number of predicates).

Reducing complexity:

Over-approximation makes computation of $\hat{T}$ faster

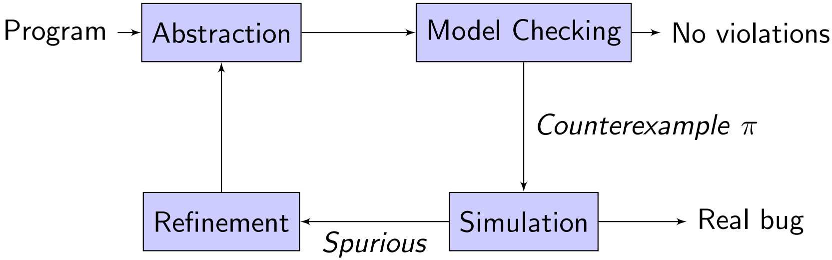
[2] Also known as **minimal** or **existential** or **exact** or **eager** abstraction

Fast Abstraction[3]

- Usually very fast computation
- Many ways to approximate the abstraction:
  - Cartesian abstraction (loses the relation among predicates)
  - Predicate partitioning (loses the relation among subsets of predicates of different clusters)
- But:
  - Introduces spurious transitions (abstraction now contains both spurious transitions and spurious paths)
  - Requires more refinement iterations

[3] Also known as **lazy** or **approximated** abstraction

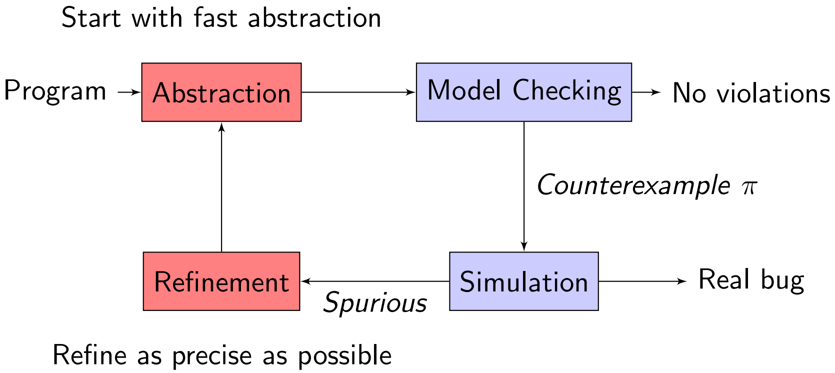
CEGAR-Loop



Key concern:

Large number of abstract-refine iterations makes verification impractical

Our Solution: Combination of fast and precise abstraction



Components of our algorithm

- `FastAbstraction` : given a set of predicates, Π, and a concrete transition relation T, computes program *over-approximation*, $\hat{T}_{\Pi}$.

- `PreciseAbstraction` : given a set of predicates, Π, and a concrete transition relation, T, computes the *minimal abstraction* $\hat{T}_{\Pi}$.

- `SpuriousTransition` (σ_{ST}): given a path π, maps every transition t in π to a set of predicates P, s.t. $P \subseteq \Pi$ and $t \not\models \hat{T}_{P}$.

- `SpuriousPath` (σ_{SP}): given a path π, maps every transition t in π to a set of predicates P, s.t. $\pi \not\models \hat{T}_{\sigma_{SP}(t)}$. Note that $\Pi \subseteq P$, i.e., `SpuriousPath` introduces new predicates.

The "synergy" algorithm

Let's proceed stepwise

```
MixCegarLoop(TransitionSystem M, Property F)
begin
    Π = InitialPredicates(F,T);
    α = FastAbstraction(T,Π);
    while not TIMEOUT do
        π = ModelCheck(α,F);
        if π = ∅ then return CORRECT;
        else
            σ_ST = SpuriousTransition(π);
            if σ_ST ≠ ∅ then
                foreach t ∈ π do
                    C = PreciseAbstraction(T,σ_ST(t));
                    α = α ∧ C;
            else
                σ_SP = SpuriousPath(π);
                if σ_SP ≠ ∅ then return INCORRECT;
                else
                    foreach t ∈ π do
                        Π = Π ∪ σ_SP(t);
                        C = PreciseAbstraction(T,σ_SP(t));
                        α = α ∧ C;
end
```

The "synergy" algorithm

```
MixCegarLoop(TransitionSystem M, Property F)
begin
    Π = InitialPredicates(F,T);
    α = FastAbstraction(T,Π);
    while not TIMEOUT do
        π = ModelCheck(α,F);
        if π = ∅ then return CORRECT;
        else
            σ_ST = SpuriousTransition(π);
            if σ_ST ≠ ∅ then
                foreach t ∈ π do
                    C = PreciseAbstraction(T,σ_ST(t));
                    α = α ∧ C;
            else
                σ_SP = SpuriousPath(π);
                if σ_SP ≠ ∅ then return INCORRECT;
                else
                    foreach t ∈ π do
                        Π = Π ∪ σ_SP(t);
                        C = PreciseAbstraction(T,σ_SP(t));
                        α = α ∧ C;
end
```

Choose initial predicates Π and use them for fast abstraction

The "synergy" algorithm

```
MixCegarLoop(TransitionSystem M, Property F)
begin
    Π = InitialPredicates(F,T);
    α = FastAbstraction(T,Π);
    while not TIMEOUT do
        π = ModelCheck(α,F);
        if π = ∅ then return CORRECT;
        else
            σ_ST = SpuriousTransition(π);
            if σ_ST ≠ ∅ then
                foreach t ∈ π do
                    C = PreciseAbstraction(T,σ_ST(t));
                    α = α ∧ C;
            else
                σ_SP = SpuriousPath(π);
                if σ_SP ≠ ∅ then return INCORRECT;
                else
                    foreach t ∈ π do
                        Π = Π ∪ σ_SP(t);
                        C = PreciseAbstraction(T,σ_SP(t));
                        α = α ∧ C;
end
```

Perform Model Checking and obtain counterexample π (if it exists)

The "synergy" algorithm

```
MixCegarLoop(TransitionSystem M, Property F)
begin
    Π = InitialPredicates(F,T);
    α = FastAbstraction(T,Π);
    while not TIMEOUT do
        π = ModelCheck(α,F);
        if π = ∅ then return CORRECT;
        else
            σ_ST = SpuriousTransition(π);
            if σ_ST ≠ ∅ then
                foreach t ∈ π do
                    C = PreciseAbstraction(T,σ_ST(t));
                    α = α ∧ C;
            else
                σ_SP = SpuriousPath(π);
                if σ_SP ≠ ∅ then return INCORRECT;
                else
                    foreach t ∈ π do
                        Π = Π ∪ σ_SP(t);
                        C = PreciseAbstraction(T,σ_SP(t));
                        α = α ∧ C;
end
```

Compute spurious transitions ($\sigma_{ST} : \forall t \in \pi \rightarrow P \subseteq \Pi \wedge t \not\models \hat{T}_P$)

The "synergy" algorithm

MixCegarLoop(**TransitionSystem M, Property F**)
begin
  Π = InitialPredicates(**F**,**T**);
  α = FastAbstraction(**T**,Π);
  while not TIMEOUT do
    π = ModelCheck(α,**F**);
    if $\pi = \emptyset$ **then** return CORRECT;
    else
      σ_{ST} = SpuriousTransition(π);
      if $\sigma_{ST} \neq \emptyset$ **then**
        foreach $t \in \pi$ **do**
          C = PreciseAbstraction(**T**,$\sigma_{ST}(t)$);
          $\alpha = \alpha \wedge C$;
      else
        σ_{SP} = SpuriousPath(π);
        if $\sigma_{SP} \neq \emptyset$ **then** return INCORRECT;
        else
          foreach $t \in \pi$ **do**
            $\Pi = \Pi \cup \sigma_{SP}(t)$;
            C = PreciseAbstraction(**T**,$\sigma_{SP}(t)$);
            $\alpha = \alpha \wedge C$;
end

1. Perform Precise-Abstraction for predicates P related to spurious transitions $\forall t \in \pi$.
2. Remove detected spurious transitions by refining original abstraction

Note, **All** spurious transitions related to detected predicates are removed at once!

The "synergy" algorithm

```
MixCegarLoop(TransitionSystem M, Property F)
begin
    Π = InitialPredicates(F,T);
    α = FastAbstraction(T,Π);
    while not TIMEOUT do
        π = ModelCheck(α,F);
        if π = ∅ then return CORRECT;
        else
            σ_ST = SpuriousTransition(π);
            if σ_ST ≠ ∅ then
                foreach t ∈ π do
                    C = PreciseAbstraction(T,σ_ST(t));
                    α = α ∧ C;
            else
                σ_SP = SpuriousPath(π);
                if σ_SP ≠ ∅ then return INCORRECT;
                else
                    foreach t ∈ π do
                        Π = Π ∪ σ_SP(t);
                        C = PreciseAbstraction(T,σ_SP(t));
                        α = α ∧ C;
end
```

Otherwise check if π has any spurious path ($\sigma_{SP} : t \in \pi \rightarrow \Pi \subseteq P \wedge \pi \not\models \hat{T}_{\sigma_{SP}(t)}$)

The "synergy" algorithm

```
MixCegarLoop(TransitionSystem M, Property F)
begin
    Π = InitialPredicates(F,T);
    α = FastAbstraction(T,Π);
    while not TIMEOUT do
        π = ModelCheck(α,F);
        if π = ∅ then return CORRECT;
        else
            σ_ST = SpuriousTransition(π);
            if σ_ST ≠ ∅ then
                foreach t ∈ π do
                    C = PreciseAbstraction(T,σ_ST(t));
                    α = α ∧ C;
            else
                σ_SP = SpuriousPath(π);
                if σ_SP ≠ ∅ then return INCORRECT;
                else
                    foreach t ∈ π do
                        Π = Π ∪ σ_SP(t);
                        C = PreciseAbstraction(T,σ_SP(t));
                        α = α ∧ C;
end
```

1. Add new predicates to Π from SpuriousPath(π).
2. Perform Precise-Abstraction for predicates P related to transitions $\forall t \in \pi$.
3. Remove spurious path by refining the original abstraction

Advantages of our algorithm

Summary:

Computes abstraction quickly but keeps it precise enough to avoid too many refinement iterations

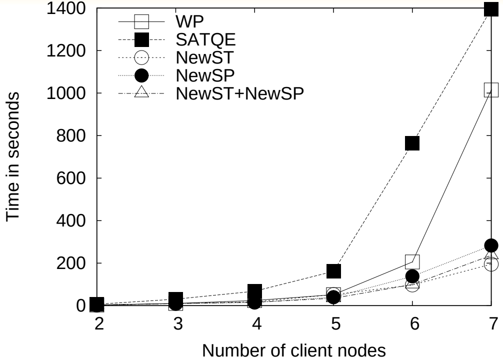
- Expensive precise abstraction is limited to a small number of predicates.
- Multiple spurious behaviors are removed at each refinement iteration (reduces CEGAR iterations)
- Synergy can be localized to some parts of the program (for every location of the control-flow graph)

Implementation

Synergy is implemented in our software model checker, SATABS:

- `FastAbstraction`— computation of the weakest precondition;
- `PreciseAbstraction`— enumeration of possible transitions by means of a SAT solver;
- `SpuriousTransition`— a SAT solver is used to check if a transition is spurious; if it is, UNSAT proof is used to find the relevant predicates;
- `SpuriousPath`— the weakest preconditions of the current predicates are computed along the transitions of the spurious path to find a set of current and new predicates

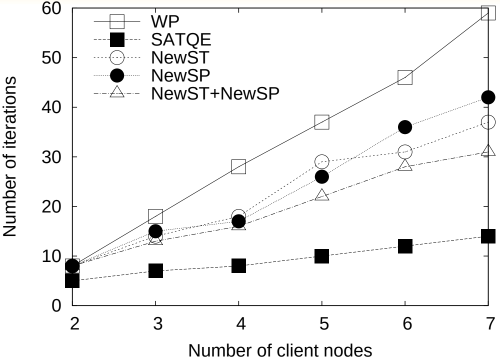
Client/server shopping agent[4]



WP – weakest-precondition-based abstraction; SATQE – SAT-based existential abstraction; NewST – refined abstraction to remove spurious transitions; NewSP – refined abstraction to remove spurious path.

[4] This example is particularly interesting because the fast abstraction produces a number of spurious transitions exponential in the number of predicates.

Client/server shopping agent[4]



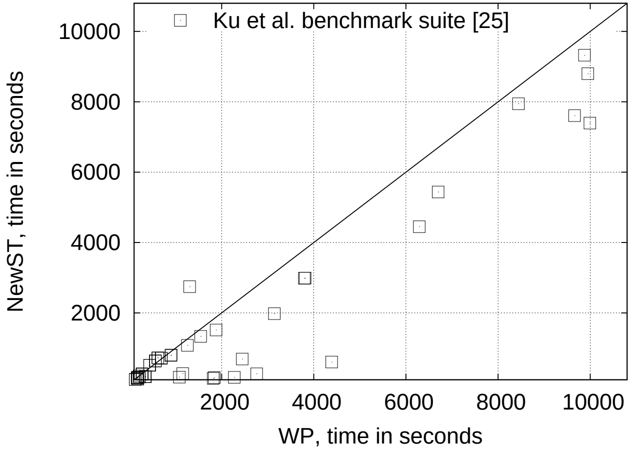
WP – weakest-precondition-based abstraction; SATQE – SAT-based existential abstraction; NewST – refined abstraction to remove spurious transitions; NewSP – refined abstraction to remove spurious path.

[4] This example is particularly interesting because the fast abstraction produces a number of spurious transitions exponential in the number of predicates.

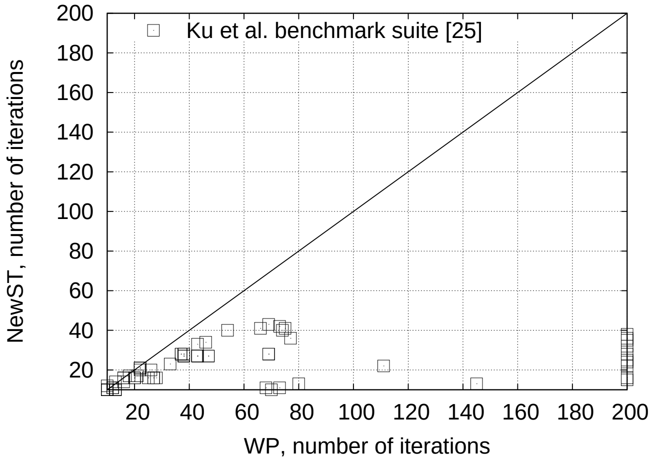
Results

- As expected, SATQE does not perform efficiently on large real programs because of the large number of predicates involved.
- Although `NewSP` outperforms SATQE, it still generates too many predicates and fails in scaling against `NewST` and WP
- The most interesting — comparison between WP and `NewST`, which gives a deeper understanding the advantage of our techniques.

Comparison of WP and NewST on a benchmark suite by Ku et. al



Comparison of WP and NewST on a benchmark suite by Ku et. al



To conclude:

- A new abstraction refinement technique that combines precise and approximated abstraction.
- Our approach outperforms both precise and imprecise techniques and reduces the number of CEGAR iterations.
- Implemented and evaluated on a number of benchmarks — **http://www.verify.inf.unisi.ch/projects/synergy**

Future Work

1. Integrate synergy with interpolation-based approaches for predicate discovery
2. Investigate trade-offs between precise and approximated approaches in the context of purely interpolation-based model checking

Thanks for listening!

Questions ?