

Formal Verification Techniques

TSG Mini-Course (6)

Hasan Amjad

Automated Reasoning Group (ARG)
Computer Laboratory
University of Cambridge

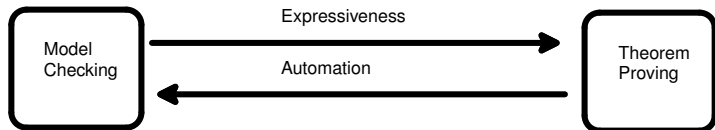
Review I - Formal Verification

- ▶ *Specification* What the system ought to do
- ▶ *Modelling* What the system actually does
- ▶ *Verification* Check that the model respects the specification

Review II - The story so far

	Sound	Complete	Decidable	Complexity
\mathcal{B}	YES	YES	YES	NP-complete 2EXP
\mathcal{PC}	YES	YES	SEMI	
QFPC	YES	YES	YES	
\mathcal{ENT}	YES	NO	NO	
\mathcal{PA}	YES	YES	YES	
\mathcal{PC}^ω	NO	NO	NO	
λ_τ	YES	NO	NO	
CTL	YES	YES	YES	NP-complete
LTL	YES	YES	YES	PSPACE-complete
L_μ	YES	YES	YES	EXPTIME-complete

Model Checking and Theorem Proving



- ▶ Theorem Proving
 - ▶ Specification: theorem statements
 - ▶ Modelling: formal definitions
 - ▶ Verification: Prove the theorems from the definitions
- ▶ Model Checking
 - ▶ Specification: properties of automaton
 - ▶ Modelling: automaton
 - ▶ Verification: show that desired states of the automaton have the required properties

Theorem proving is manual but expressive. Model checking is automatic and gives counterexamples, but is not as expressive.

Can we improve model checking expressiveness without sacrificing automation?

Kripke Models

- ▶ **Intuition** Kripke models are labelled transition systems
- ▶ A *Kripke model* or Kripke structure M , is a quintuple $\langle S_M, S0_M, T_M, P_M, L_M \rangle$ where,
 1. S_M is the set of *states*
 2. $S0_M \subseteq S_M$ is the set of *initial states*
 3. T_M is the finite set of *transitions* $a_i \subseteq S_M \times S_M$
 4. P_M is the finite set of *atomic propositions* p_0, \dots
 5. $L_M : S_M \rightarrow 2^{P_M}$ is a *labelling function* that, for every state, returns the set of p_i true in that state

Predicate abstraction

A simple example

Consider the program

```
x:=0; y:=1;
```

```
while (x<y) do {  
  x:=x+y;  
  y:=y+x;  
}
```

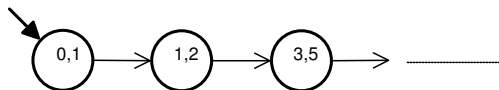
Predicate abstraction

A simple example

Consider the program

```
x:=0; y:=1;
```

```
while (x<y) do {  
  x:=x+y;  
  y:=y+x;  
}
```



Predicate abstraction

A simple example

Consider the program

```
x:=0; y:=1;
```

```
while (x<y) do {  
  x:=x+y;  
  y:=y+x;  
}
```

This can be formalised by the Kripke model

$$S = \mathbb{N} \times \mathbb{N}$$

$$S_0 = \{(0, 1)\}$$

$$T((x, y), (x', y')) = x < y \Rightarrow (x' = x + y \wedge y' = x + 2y)$$

$$P = \{ "x < y" \}$$

$$L((x, y)) = \text{if } x < y \text{ then } \{ "x < y" \} \text{ else } \emptyset$$

Predicate abstraction

A simple example

Consider the program

```
x:=0; y:=1;
```

```
while (x<y) do {  
  x:=x+y;  
  y:=y+x;  
}
```

Supposed we wish to check that **AG** ($x < y$). We can replace $x < y$ by a boolean B :

$$S = \mathbb{B}$$

$$S_0 = \{\mathbf{t}\}$$

$$T(B, B') = B \Rightarrow B'$$

$$P = \{\text{"B"}\}$$

$$L(B) = \text{if } B \text{ then } \{\text{"B"}\} \text{ else } \emptyset$$

Predicate abstraction

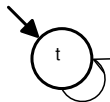
A simple example

Consider the program

```
x:=0; y:=1;
```

```
while (x<y) do {  
  x:=x+y;  
  y:=y+x;  
}
```

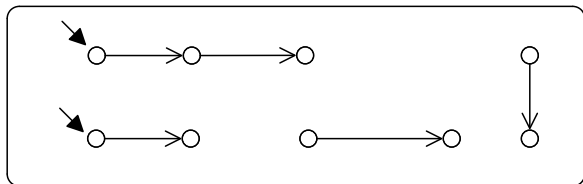
And now check **AG B**



Abstraction and refinement

Intuition

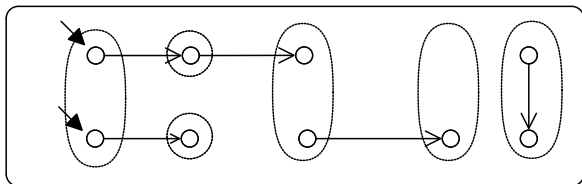
Basic idea is to *overapproximate* the set of reachable states



Abstraction and refinement

Intuition

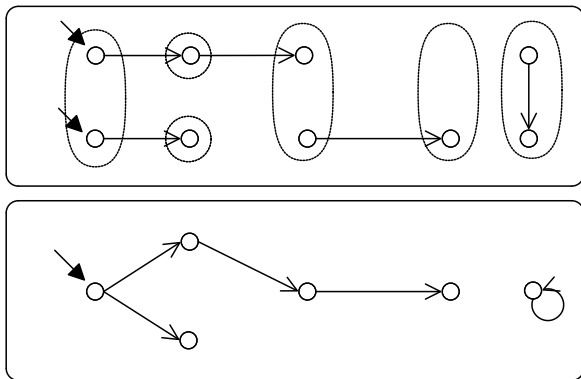
Basic idea is to *overapproximate* the set of reachable states



Abstraction and refinement

Intuition

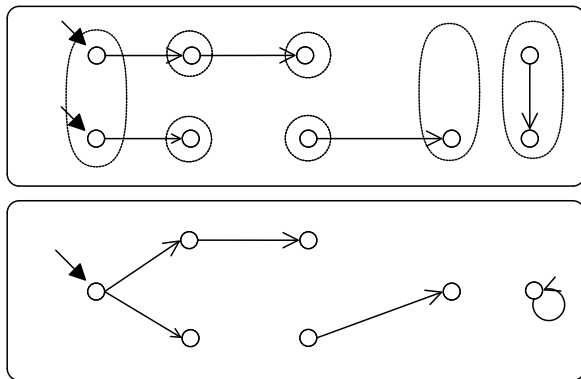
Basic idea is to *overapproximate* the set of reachable states



Abstraction and refinement

Intuition

Basic idea is to *overapproximate* the set of reachable states

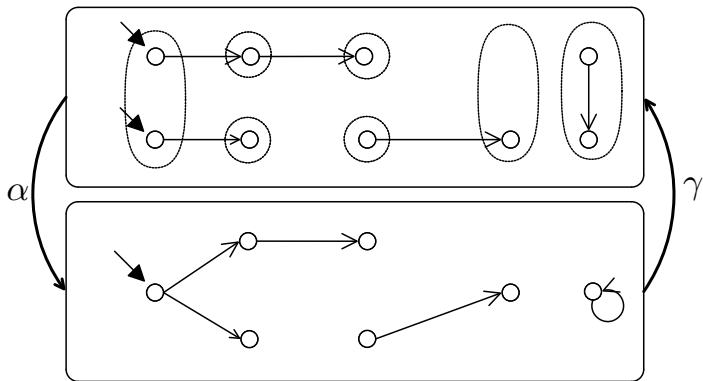


Then *refine* the abstraction if it was too coarse

Abstraction and refinement

Intuition

Basic idea is to *overapproximate* the set of reachable states



Then *refine* the abstraction if it was too coarse

Predicate Abstraction

Preliminaries

- ▶ We use ϕ to refer both to a formula on state variables \bar{v} or the set of states in which ϕ holds
- ▶ Transitions $a_i(\bar{v}, \bar{v}')$ as guarded commands

$$g_i(\bar{v}) \mapsto \bar{v} := \text{ass}_i(\bar{v})$$

Then $a_i(\bar{v})$ is all states to which the state \bar{v} has a transition

- ▶ Let $R \subseteq S \times S$ and $\phi \in \mathcal{P}(S)$. Then,
 1. **post** $[R](\phi) = \exists q. R(q, q') \wedge \phi(q)$
 2. **pre** $[R](\phi) = \forall q'. R(q, q') \Rightarrow \phi(q')$
- ▶ Weakest preconditions on transitions

$$\mathbf{pre}[a_i](\phi) = (g_i(\bar{v}) \Rightarrow \phi[\text{ass}_i(\bar{v})/\bar{v}])$$

- ▶ **post** $[R](\phi) \Rightarrow \phi'$ iff $\phi \Rightarrow \mathbf{pre}[R](\phi')$
- ▶ A *Galois connection* is a pair of functions

$(\alpha : \mathcal{P}(S) \rightarrow \hat{S}, \gamma : \hat{S} \rightarrow \mathcal{P}(S))$ such that \hat{S} is a lattice and

1. $\alpha(\gamma(\hat{s} \in \hat{S})) = \hat{s}$ and $\phi \Rightarrow \gamma(\alpha(\phi))$
2. Given γ ,

$$\alpha(\phi) = \bigsqcap \{\hat{s} \in \hat{S} \mid \phi \Rightarrow \gamma(\hat{s})\}$$

Predicate Abstraction

Constructing the abstract model

- ▶ \hat{M} is an abstraction of M if
 1. $(\alpha : \mathcal{P}(S) \rightarrow \hat{S}, \gamma : \hat{S} \rightarrow \mathcal{P}(S))$ is a Galois connection
 2. $S_0 \subseteq \gamma(\hat{S}_0)$
 3. $\forall i \hat{s}. \mathbf{post}[a_i](\gamma(\hat{s})) \subseteq \gamma(\hat{a}_i(\hat{s}))$
- ▶ Suppose we have abstraction predicates ϕ_1, \dots, ϕ_l on concrete state variables. Then an abstract state is a monomial on (B_0, \dots, B_l) , over the lattice \mathbb{B}^l
- ▶ The concretization function is $\gamma(\hat{s}(\bar{B})) = \hat{s}[\bar{\phi}/\bar{B}]$
- ▶ Then the abstraction function is defined implicitly by

$$\alpha(\phi) = \bigwedge_{i=1}^l \{\hat{s} \mid \phi \Rightarrow \gamma(\hat{s})\}$$

- ▶ But we use the overapproximation

$$\alpha'(\phi) = \bigwedge_{i=1}^l \{B_i \mid \phi \Rightarrow \phi_i\}$$

- ▶ Then the successors $\hat{a}_i(\hat{s})$ of an abstract state are given by

$$\left\{ \begin{array}{l} \perp \\ \bigwedge_{j=1}^l \left\{ \begin{array}{ll} B_j & \text{if } \mathbf{post}[a_i](\gamma(\hat{s})) \Rightarrow \phi_j \\ \neg B_j & \text{if } \mathbf{post}[a_i](\gamma(\hat{s})) \Rightarrow \neg \phi_j \\ \mathbf{t} & \text{otherwise} \end{array} \right\} \end{array} \right\} \begin{array}{l} \text{if } \gamma(\hat{s}) \Rightarrow \neg g_i \\ \text{otherwise} \end{array}$$

Predicate Abstraction

Using the abstract model

- ▶ We can simplify the successor computation from

$$\bigwedge_{j=1}^I \begin{cases} B_j & \text{if } \mathbf{post}[a_j](\gamma(\hat{s})) \Rightarrow \phi_j \\ \neg B_j & \text{if } \mathbf{post}[a_j](\gamma(\hat{s})) \Rightarrow \neg\phi_j \\ \mathbf{t} & \text{otherwise} \end{cases}$$

to

$$\bigwedge_{j=1}^I \begin{cases} B_j & \text{if } \gamma(\hat{s}) \wedge g_j \Rightarrow \phi_j[\text{ass}_i(\bar{v})/\bar{v}] \\ \neg B_j & \text{if } \gamma(\hat{s}) \wedge g_j \Rightarrow \neg\phi_j[\text{ass}_i(\bar{v})/\bar{v}] \\ \mathbf{t} & \text{otherwise} \end{cases}$$

If the ϕ_j are in decidable theories, this can be computed using theorem proving decision procedures

- ▶ Set of abstract initial states $\hat{S}0$ is just $\alpha'(S0)$
- ▶ For each transition abstraction, we need to make 2^I calls to the decision procedure

Predicate Abstraction

Improving the abstract model

- ▶ The abstraction is too coarse: not enough monomials to go around
- ▶ Relax monomial restriction: \hat{s} can now correspond to any formula over B_1, \dots, B_l
- ▶ To compute an abstract transition, we need 2^{2^l} DP calls. However, \hat{S} is now a Boolean algebra, so in fact only $3^l - 1$ calls are needed
- ▶ So a transition $\hat{a}(\hat{s}) = \hat{g}(\hat{s}) \mapsto \bar{B} := a^{\hat{s}}(B)$ is computed by

$$B_i := \begin{cases} \mathbf{t} & \text{post}[a](\mathbf{t}) \Rightarrow \gamma(B_i) \\ \mathbf{f} & \text{post}[a](\mathbf{t}) \Rightarrow \neg\gamma(B_i) \\ ? & \text{otherwise} \end{cases}$$

and $\hat{g} = \alpha(g)$

- ▶ As before, the set of abstract initial states \hat{S}_0 is just $\alpha(S_0)$

Predicate Abstraction

Remarks

- ▶ Overapproximation preserves safety properties only
- ▶ *Symbolic approaches* are more efficient, but limited to theories that admit boolean encoding
- ▶ Abstraction predicates are usually initially those that occur in the guards
- ▶ Additionally we may use predicates occurring in the property to be verified

Refining the Abstraction

What happens if the property was not verified because the abstraction was too coarse?

- ▶ We can add more abstraction predicates to refine the abstraction
- ▶ Where do these come from?
 1. Unused guard predicates
 2. Unused property predicates
- ▶ We can derive new predicates from the counterexample itself

Counterexample guided refinement

Suppose all guard and property predicates have been used.

If we get an abstract counterexample then,

- ▶ Either it has a corresponding concrete trace
- ▶ Or, due to non-determinism, the abstract trace is spurious

We can check this by attempting to concretize the trace

If trace is spurious,

1. Some abstract state \hat{s} has more than one successor on some transition \hat{a}_i . So some B_i in \hat{a}_i is being assigned "?"
2. Find $\mathbf{pre}[a_i](\gamma(B_i))$ and assign it to new variable B_{i+1}
3. *Refine* as follows

Let

$$b_i^{\mathbf{t}} = \bigvee \{ \hat{s} \mid \mathbf{post}[a_i](\gamma(\hat{s})) \Rightarrow \gamma(B_i) \}$$

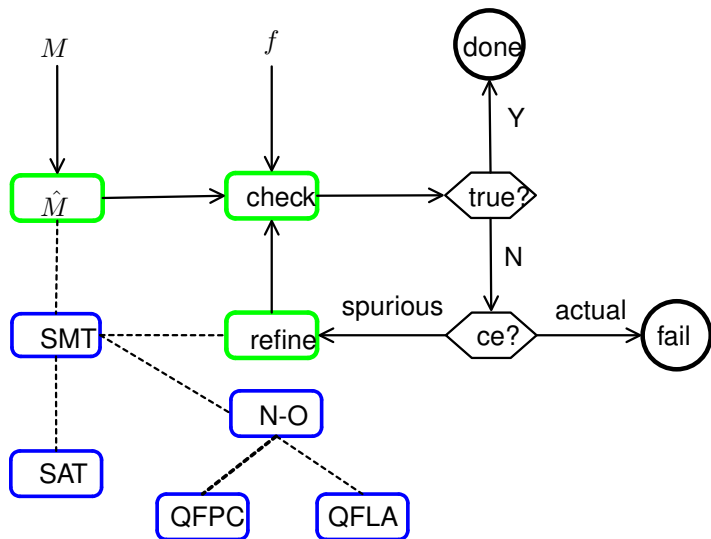
and

$$b_i^{\mathbf{f}} = \bigvee \{ \hat{s} \mid \mathbf{post}[a_i](\gamma(\hat{s})) \Rightarrow \neg \gamma(B_i) \}$$

Then for the "?" case of B_i for \hat{a}_i ,

$$B_i := \text{if } b_i^{\mathbf{t}} \text{ then } \mathbf{t} \text{ else if } b_i^{\mathbf{f}} \text{ then } \mathbf{f} \text{ else ?}$$

The Complete Automated Framework



Integration with manual provers

- ▶ As a decision procedure for temporal logics
- ▶ Embedding can be deep or shallow
- ▶ Execution may allow for manual intervention

Further Developments

- ▶ Bounded Model Checking (Biere1999)
- ▶ Reduction to boolean formulas (Lahiri2003)
- ▶ SAT-boosted Presburger Arithmetic (Kroening2004)

Synopsis

- ▶ Formal Verification: automating formal proofs of systems
- ▶ Theorem proving: semi-automatic proof over all logics
- ▶ Model checking: automatic proof and debugging over decidable automata
- ▶ Research
 - ▶ Logics and automata
 - ▶ Proof procedures
 - ▶ Combinations
 - ▶ Frameworks

Further Reading

1. H. Saidi. Model Checking Guided Abstraction and Analysis. SAS 2000.
2. S. Lahiri et al. A Symbolic Approach to Predicate Abstraction. CAV 2003.
3. E. Clarke et al. SAT-based Abstraction Refinement using ILP and Machine Learning Techniques. CAV 2002.
4. A. Biere et al. Model Checking without BDDs. TACAS 1999.