What else can Satisfiability Modulo Theories do for us?

Philipp Rümmer
March 3, 2021

What happened so far ...

- Use of SMT in SymEx and DSE
- Satisfiability queries

In this lecture: more tools

- Models
- Unsat cores
- Quantifier elimination
- Craig interpolation

Solutions and Models

- **Task:**
  Produce a satisfying assignment for a given formula.

- **SMT-LIB commands:**
  - `(set-option :produce-models true)`
  - `(get-model)`, `(get-value (x y))`,
    called after `(get-model)` returns sat

Unsatisfiable Cores

- **Task:**
  Given an unsatisfiable set *F* of formulas,
  find a small unsatisfiable subset *F'* of *F*.

- **SMT-LIB commands:**
  - `(set-option :produce-unsat-cores true)`
  - `(assert (! … :named A))`
  - `(get-unsat-core)`,
    called after `(check-sat)` returns unsat

Example

```
; This example illustates extraction
; of unsatisfiable cores (a subset of assertions
; that are mutually unsatisfiable)
(set-option :produce-unsat-cores true)
(declare-fun p () Bool)
(declare-fun q () Bool)
(declare-fun r () Bool)
(declare-fun s () Bool)
; Z3 will only track assertions that are named.
(assert (! (or p q) :named a1))
(assert (! (=> r s) :named a2))
(assert (! (=> s (= q r)) :named a3))
(assert (! (or r p) :named a4))
(assert (! (or r s) :named a5))
(assert (! (not (and r q)) :named a6))
(assert (! (not (and s p)) :named a7))
(check-sat)
(get-unsat-core)
```

Permalink: http://logicrunch.it.uu.se:4096/~wv/princess/?ex=perma%2F1614771123_487382835

Unsatisfiable Cores (2)

- Computed cores are not guaranteed to be minimal (“best-effort”)
  - Idea is to make best use of the information a solver already has available
- Finding truly minimal cores is hard:
  - Repeated sat queries needed
  - Active research area: “Minimally unsatisfiable sets” (MUSes)

Cores in Symbolic Execution?

Quantifier Elimination

- **Task:**
  Given a formula *phi*, find an equivalent quantifier-free formula *phi'*.

- Not standardized in SMT-LIB, but supported by several solvers: Z3, CVC4, Princess, ...

Some Examples

Z3 QE Example

```
(declare-const x Int)
(assert (exists ((y Int))
           (and (> y 0) (or (> x y) (> x 42)))))
(apply qe)
```

Permalink: https://rise4fun.com/Z3/WC8ib

QE in Symbolic Execution?

```
int abs(int x) {
    if (x >= 0) {
        return x;
    } else {
        int t = -x;
        return t;
    }
}
```

Systematic QE

“Geometric” Approach to QE

1) Pick an innermost quantifier, make it existential
2) Push the quantifier down (mini-scoping), rewrite the matrix to DNF
3) Eliminate the quantified variable from each disjunct, drop the quantifier
4) Continue with 1)

“Geometric” Approach to QE

Formula underneath the quantifier

1) Pick an innermost quantifier, existential
2) Push the quantifier down (mini-scoping), rewrite the matrix to DNF
3) Eliminate the quantified variable from each disjunct, drop the quantifier
4) Continue with 1)

"Geometric" Approach to QE

1) Pick an innermost quantifier, make it existential
2) Push the quantifier down (mini-scoping), rewrite the matrix to DNF
3) Eliminate the quantified variable from each disjunct, drop the quantifier
4) Continue with 1)

Exponential blow-up 1

“Geometric” Approach to QE

1) Pick an innermost quantifier, make it existential
2) Push the quantifier down (mini-scoping), rewrite the matrix to DNF
3) Eliminate the quantified variable from each disjunct, drop the quantifier
4) Continue with 1)

Exponential blow-up 2
(over integers)

Exponential blow-up 1

Different Paradigms

- Geometric approach

- Instantiation-based approach:

$$\exists x.\phi[x] \quad \rightsquigarrow \quad \bigvee_{t \in T} \phi[t]$$

- Both can be implemented efficiently using SMT techniques (e.g., expand lazily)

Which Theories admit QE?

- Booleans
- LIA, NIA: integer arithmetic
- LRA, NRA: real arithmetic
- FP: floating-point arithmetic
- BV: bitvectors
- EUF: equality + uninterpr. functions
- Arrays
- ADTs: algebraic data-types
- Strings

Craig Interpolation

- **Task:**
  Given an unsatisfiable conjunction of formulas, extract binary/sequence/tree interpolants.

- Not standardized in SMT-LIB, but supported by several solvers: MathSAT, SMTInterpol, Princess, ...

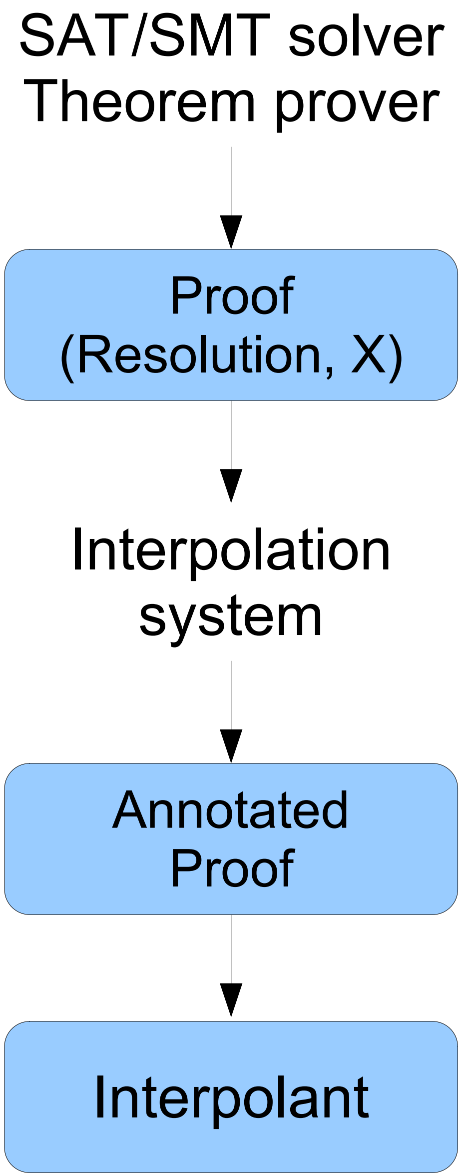
Binary Interpolants

Definition
Suppose a conjunction $A \wedge B$ is given.
A *binary interpolant* is a formula I such that
- $A \rightarrow I$ and $B \rightarrow \neg I$ are valid, and
- every non-logical symbol of I occurs in both A and B.

- “Non-logical” symbols: variables, uninterpreted functions, etc.
- Clearly: if I exists, then the conjunction $A \wedge B$ is unsat
- Interpolation property: the converse

Examples, Intuition

Interpolants from proofs



Model Checking

- *Safety for finite-state systems:*
  Transition system: $I(\bar{s}), T(\bar{s}, \bar{s}')$
  Property: $P(\bar{s})$
- Bounded model checking:

$$I(\bar{s}_0) \wedge \neg P(\bar{s}_0) \quad ?$$

$$I(\bar{s}_0) \wedge T(\bar{s}_0, \bar{s}_1) \wedge \neg P(\bar{s}_1) \quad ?$$

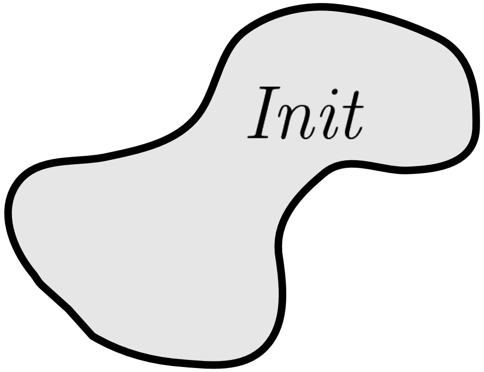
$$I(\bar{s}_0) \wedge T(\bar{s}_0, \bar{s}_1) \wedge T(\bar{s}_1, \bar{s}_2) \wedge \neg P(\bar{s}_2) \quad ?$$

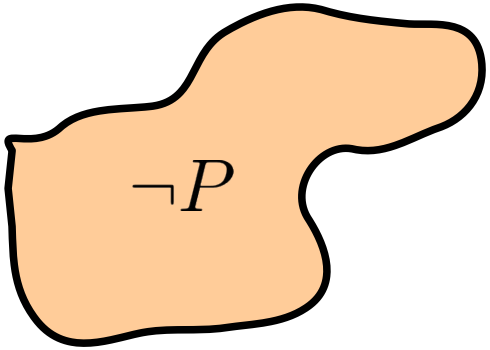
$$\vdots$$

Interpolation-based MC (simp.)

[McMillan, 2003]

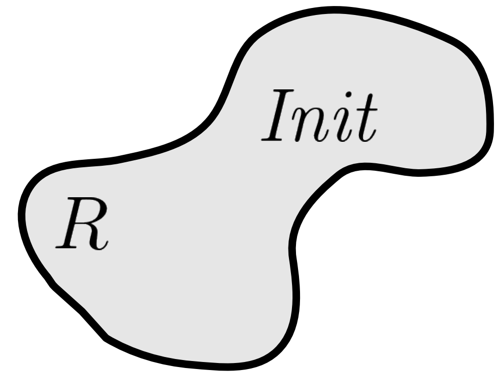
If $Init(\bar{s}) \wedge \neg P(\bar{s})$ is satisfiable
    return Unsafe
$R = Init(\bar{s}_{-1})$
while (**true**) {
    $A = R \wedge T(\bar{s}_{-1}, \bar{s})$
    $B = T(\bar{s}, \bar{s}_1) \wedge (\neg P(\bar{s}) \vee \neg P(\bar{s}_1))$
    if $A \wedge B$ is satisfiable {
        return Unknown
    } **else** {
        $R' = R \vee \mathsf{Itp}(A, B)[\bar{s}/\bar{s}_{-1}]$
        if $R == R'$
            return Safe
        else
            $R = R'$
    }
}

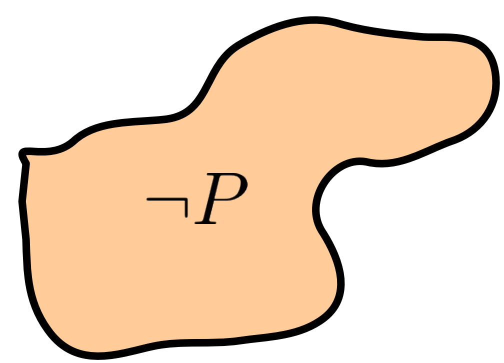




Interpolation-based MC (simp.)

[McMillan, 2003]



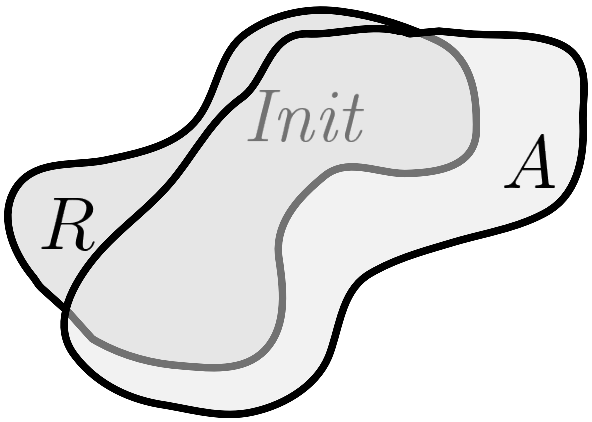


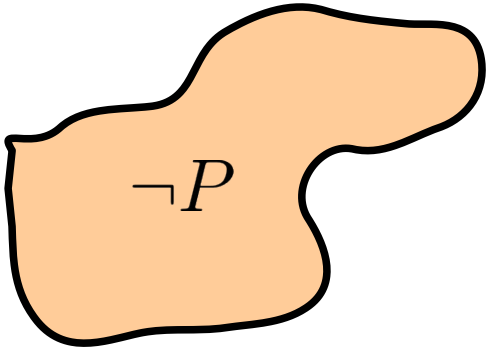

```
If  Init(s̄) ∧ ¬P(s̄) is satisfiable
   return Unsafe
R = Init(s̄_{-1})
while (true) {
   A = R ∧ T(s̄_{-1}, s̄)
   B = T(s̄, s̄_1) ∧ (¬P(s̄) ∨ ¬P(s̄_1))
   if A ∧ B is satisfiable {
      return Unknown
   } else {
      R' = R ∨ Itp(A, B)[s̄/s̄_{-1}]
      if  R == R'
         return Safe
      else
         R = R'
   }
}
```

Interpolation-based MC (simp.)

[McMillan, 2003]

If $Init(\bar{s}) \wedge \neg P(\bar{s})$ is satisfiable
    return Unsafe
$R = Init(\bar{s}_{-1})$
while (**true**) {
    $A = R \wedge T(\bar{s}_{-1}, \bar{s})$
    $B = T(\bar{s}, \bar{s}_1) \wedge (\neg P(\bar{s}) \vee \neg P(\bar{s}_1))$
    if $A \wedge B$ is satisfiable {
        return Unknown
    } **else** {
        $R' = R \vee \mathsf{Itp}(A, B)[\bar{s}/\bar{s}_{-1}]$
        if $R == R'$
            return Safe
        else
            $R = R'$
    }
}

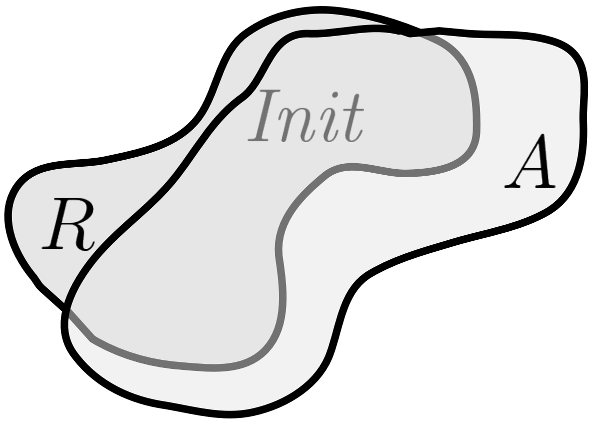


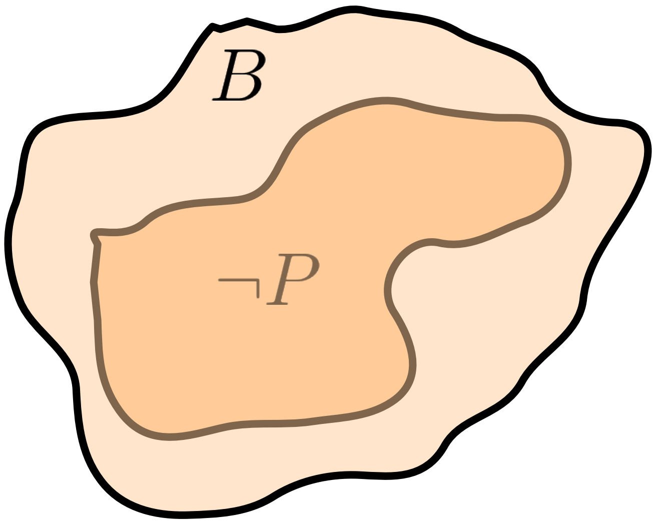


Interpolation-based MC (simp.)

[McMillan, 2003]

If $\mathit{Init}(\bar{s}) \wedge \neg P(\bar{s})$ is satisfiable
  return Unsafe
$R = \mathit{Init}(\bar{s}_{-1})$
while (**true**) {
  $A = R \wedge T(\bar{s}_{-1}, \bar{s})$
  $B = T(\bar{s}, \bar{s}_1) \wedge (\neg P(\bar{s}) \vee \neg P(\bar{s}_1))$
  if $A \wedge B$ is satisfiable {
    return Unknown
  } **else** {
    $R' = R \vee \mathsf{Itp}(A, B)[\bar{s}/\bar{s}_{-1}]$
    if $R == R'$
      return Safe
    else
      $R = R'$
  }
}

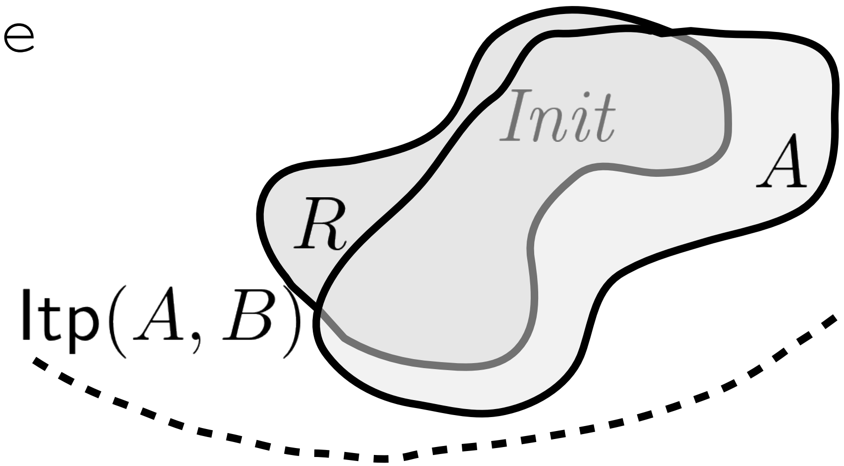


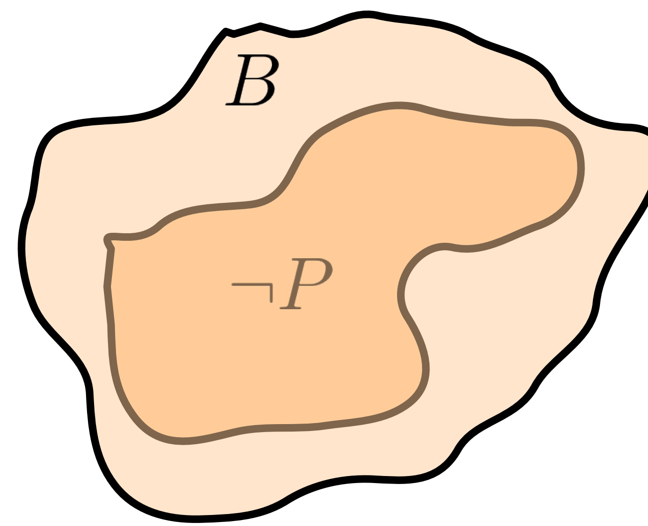


Interpolation-based MC (simp.)

[McMillan, 2003]

If $Init(\bar{s}) \wedge \neg P(\bar{s})$ is satisfiable
    return Unsafe
$R = Init(\bar{s}_{-1})$
while (**true**) {
    $A = R \wedge T(\bar{s}_{-1}, \bar{s})$
    $B = T(\bar{s}, \bar{s}_1) \wedge (\neg P(\bar{s}) \vee \neg P(\bar{s}_1))$
    if $A \wedge B$ is satisfiable {
        return Unknown
    } **else** {
        $R' = R \vee \mathsf{Itp}(A, B)[\bar{s}/\bar{s}_{-1}]$
        if $R == R'$
            return Safe
        else
            $R = R'$
    }
}

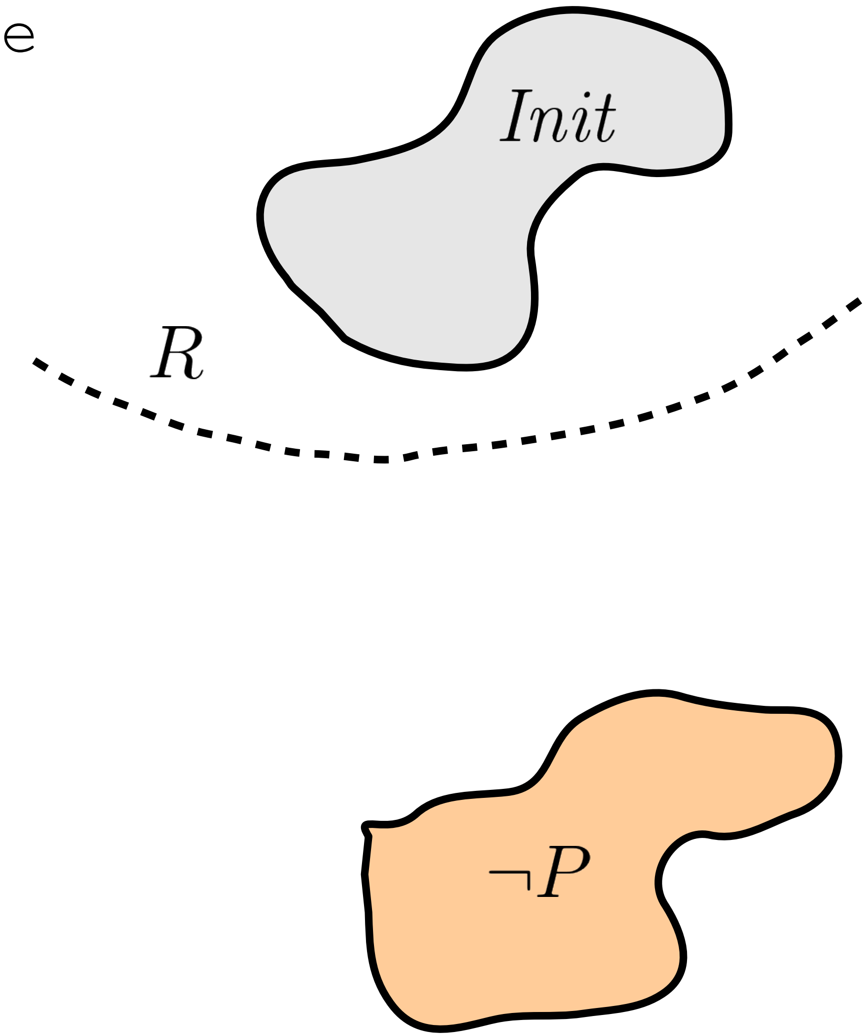




Interpolation-based MC (simp.)

[McMillan, 2003]

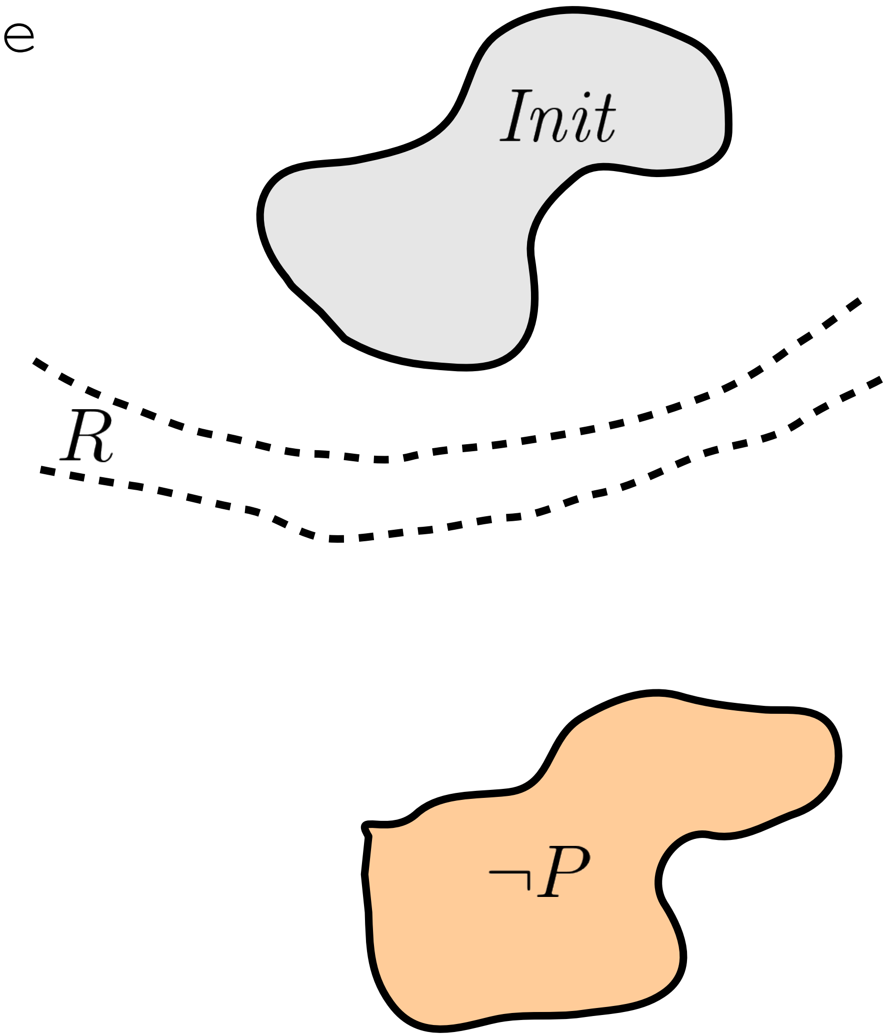
If $Init(\bar{s}) \wedge \neg P(\bar{s})$ is satisfiable
    return Unsafe
$R = Init(\bar{s}_{-1})$
while (**true**) {
    $A = R \wedge T(\bar{s}_{-1}, \bar{s})$
    $B = T(\bar{s}, \bar{s}_1) \wedge (\neg P(\bar{s}) \vee \neg P(\bar{s}_1))$
    if $A \wedge B$ is satisfiable {
        return Unknown
    } **else** {
        $R' = R \vee \mathsf{Itp}(A, B)[\bar{s}/\bar{s}_{-1}]$
        if $R == R'$
            return Safe
        else
            $R = R'$
    }
}



Interpolation-based MC (simp.)

[McMillan, 2003]

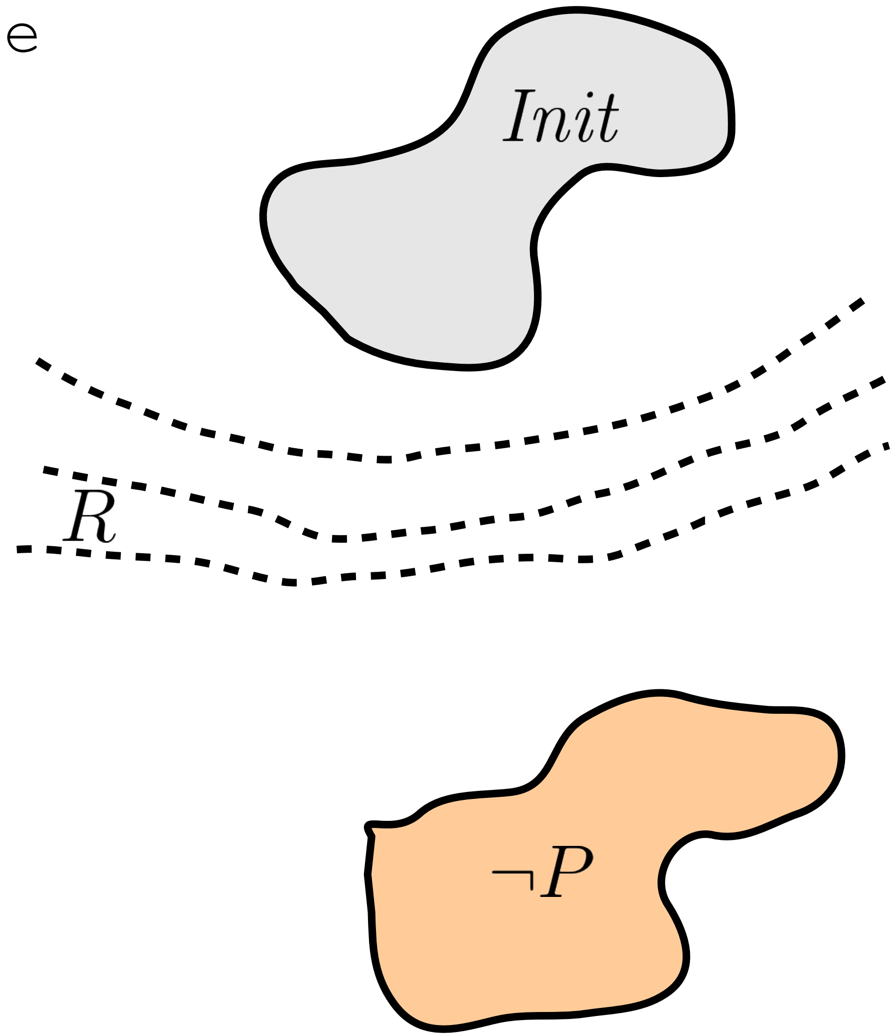
If $Init(\bar{s}) \wedge \neg P(\bar{s})$ is satisfiable
    return Unsafe
$R = Init(\bar{s}_{-1})$
while (**true**) {
    $A = R \wedge T(\bar{s}_{-1}, \bar{s})$
    $B = T(\bar{s}, \bar{s}_1) \wedge (\neg P(\bar{s}) \vee \neg P(\bar{s}_1))$
    if $A \wedge B$ is satisfiable {
        return Unknown
    } **else** {
        $R' = R \vee \mathsf{Itp}(A, B)[\bar{s}/\bar{s}_{-1}]$
        if $R == R'$
            return Safe
        else
            $R = R'$
    }
}



Interpolation-based MC (simp.)

[McMillan, 2003]

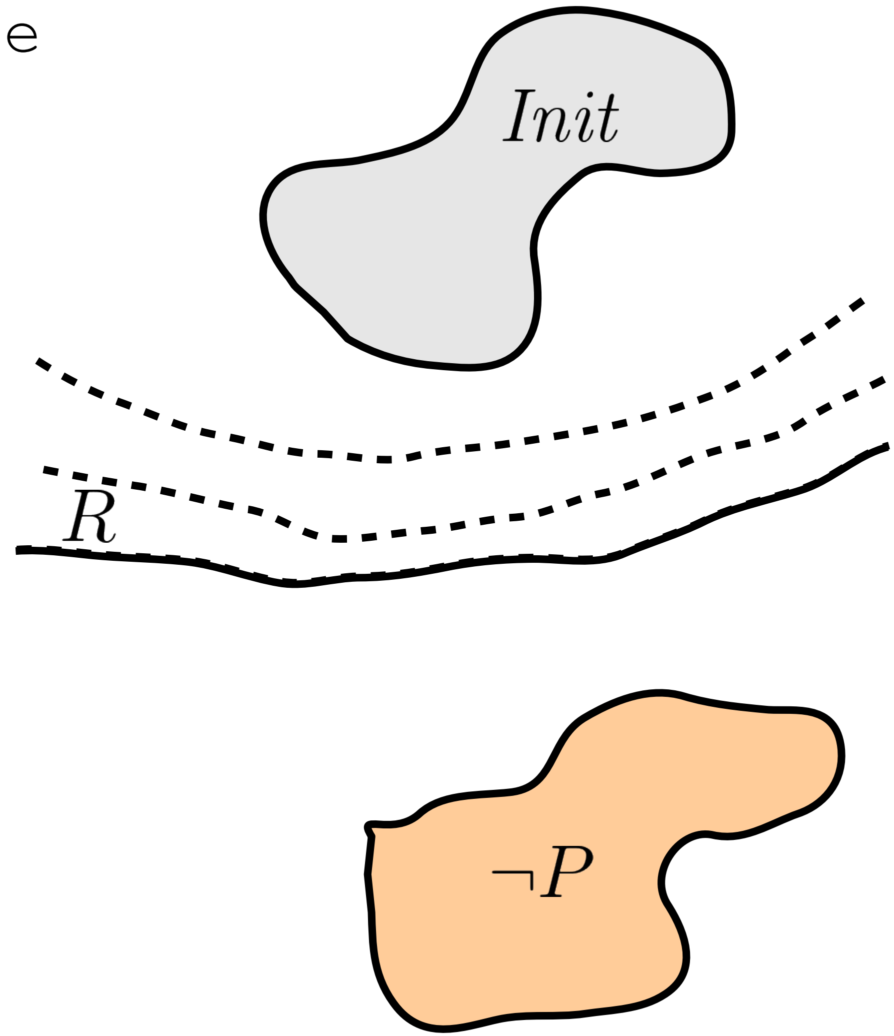
If $\mathit{Init}(\bar{s}) \wedge \neg P(\bar{s})$ is satisfiable
    return Unsafe
$R = \mathit{Init}(\bar{s}_{-1})$
while (**true**) {
    $A = R \wedge T(\bar{s}_{-1}, \bar{s})$
    $B = T(\bar{s}, \bar{s}_1) \wedge (\neg P(\bar{s}) \vee \neg P(\bar{s}_1))$
    if $A \wedge B$ is satisfiable {
        return Unknown
    } **else** {
        $R' = R \vee \mathsf{Itp}(A, B)[\bar{s}/\bar{s}_{-1}]$
        if $R == R'$
            return Safe
        else
            $R = R'$
    }
}



Interpolation-based MC (simp.)

[McMillan, 2003]

```
If  Init(s̄) ∧ ¬P(s̄) is satisfiable
   return Unsafe
R = Init(s̄_{-1})
while (true) {
   A = R ∧ T(s̄_{-1}, s̄)
   B = T(s̄, s̄_1) ∧ (¬P(s̄) ∨ ¬P(s̄_1))
   if A ∧ B is satisfiable {
      return Unknown
   } else {
      R' = R ∨ Itp(A, B)[s̄/s̄_{-1}]
      if  R == R'
         return Safe
      else
         R = R'
   }
}
```



Interpolant sequence

Definition
Given: a conjunction $T_1 \wedge \cdots \wedge T_n$.
An *interpolant sequence* is a sequence $I_0, \ldots, I_n$ of formulae such that

- $I_0 = \top$
- $I_n = \bot$
- $I_{i-1}, T_i \vdash I_i$ for each $i = 1, \ldots, n$
- for each $i = 1, \ldots, n$, the formula I_i only contains symbols common to $T_1 \wedge \cdots \wedge T_i$ and $T_{i+1} \wedge \cdots \wedge T_n$

Computation of sequence int.

Lemma
If a logic/theory admits binary interpolants, it also admits sequence interpolants.

Proof:
Solve a sequence of binary interpolation problems:

$$
\begin{array}{rcl}
 & & I_0 := \top \\
\big(I_0 \wedge T_1\big) \wedge \big(T_2 \wedge \cdots \wedge T_n\big) & \leadsto & I_1 \\
\big(I_1 \wedge T_2\big) \wedge \big(T_3 \wedge \cdots \wedge T_n\big) & \leadsto & I_2 \\
 & \vdots & \\
\big(I_{i-1} \wedge T_i\big) \wedge \big(T_{i+1} \wedge \cdots \wedge T_n\big) & \leadsto & I_i
\end{array}
$$

Computation of sequence int. (2)

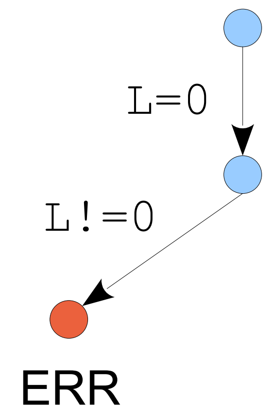
- In practice:
  - Compute a single SAT/SMT proof
  - Extract a sequence of interpolants directly from this proof
  - Meta-argument: this yields actual interpolant sequence

Software model checking
[McMillan, 2006]

```
L = 0;
do {
   assert(L==0);          lock()
   L = 1;
   old = new;
   if (*){
      L = 0;              unlock()
      new++;
   }
} while (new!=old);
```

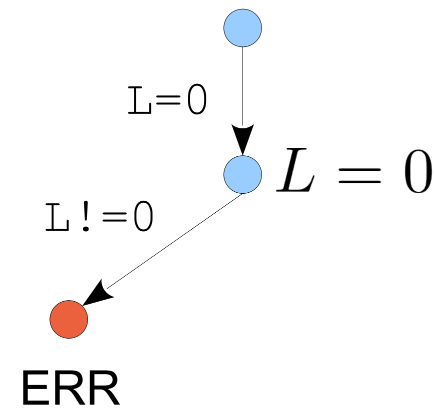
Software model checking
[McMillan, 2006]

```
L = 0;
do {
   assert(L==0);  } lock()
   L = 1;
   old = new;
   if (*){
      L = 0;      } unlock()
      new++;
   }
} while (new!=old);
```



Software model checking
[McMillan, 2006]

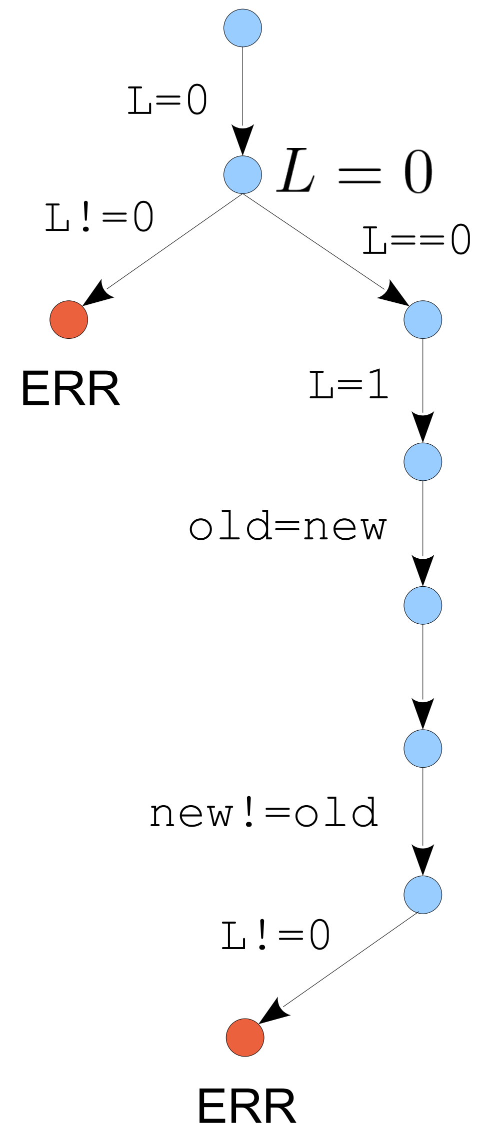
```
L = 0;
do {
   assert(L==0);
   L = 1;           } lock()
   old = new;
   if (*){
      L = 0;        } unlock()
      new++;
   }
} while (new!=old);
```



Software model checking

[McMillan, 2006]

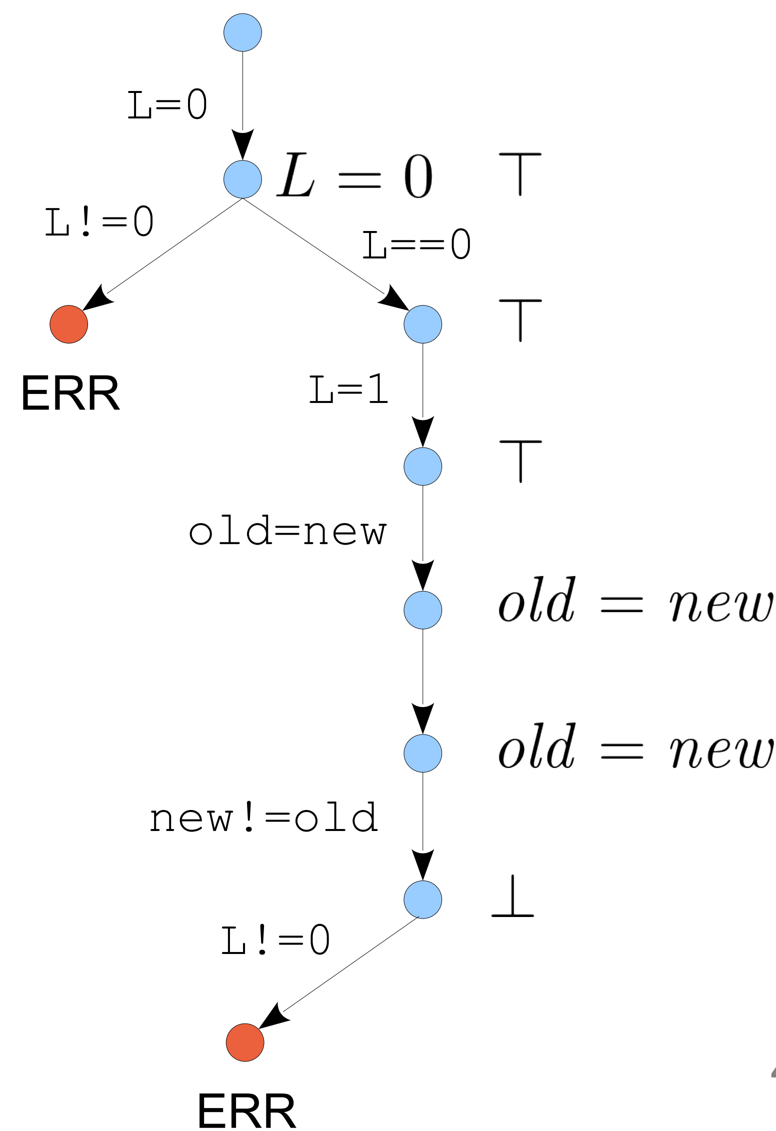
```
L = 0;
do {
   assert(L==0);    } lock()
   L = 1;
   old = new;
   if (*){
      L = 0;        } unlock()
      new++;
   }
} while (new!=old);
```



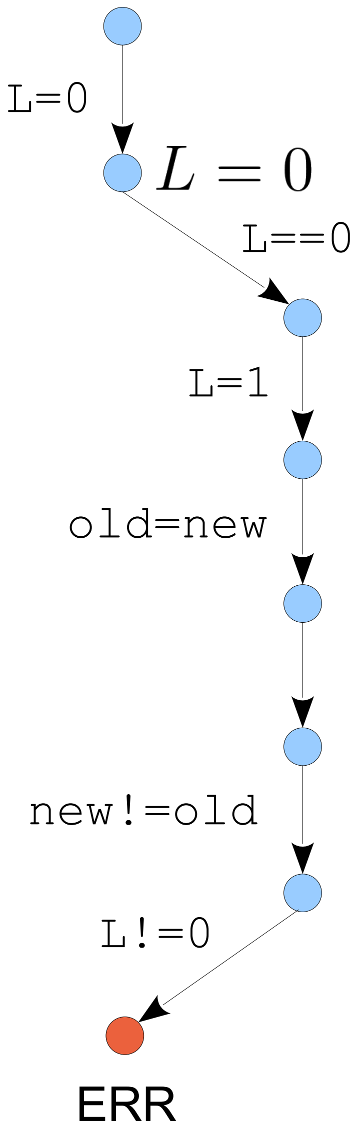
Software model checking

[McMillan, 2006]

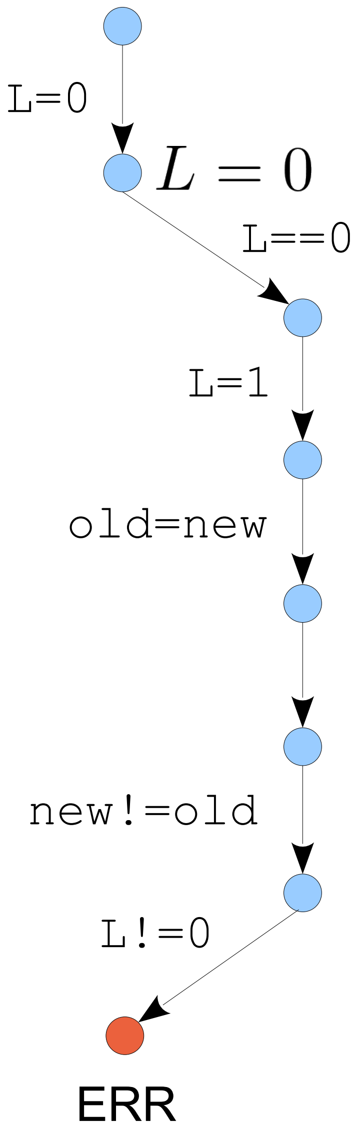
```
L = 0;
do {
    assert(L==0);    } lock()
    L = 1;
    old = new;
    if (*){
        L = 0;       } unlock()
        new++;
    }
} while (new!=old);
```



In the Example



In the Example



$$T_1 : \quad L_0 = 0$$

$$T_2 : \quad L_0 = 0$$

$$T_3 : \quad L_1 = 1$$

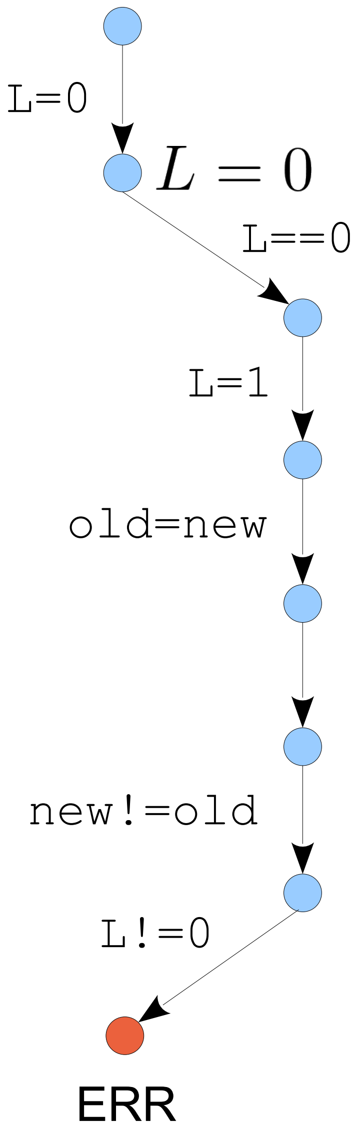
$$T_4 : \quad old_1 = new_0$$

$$T_5 : \quad \top$$

$$T_6 : \quad new_0 \neq old_1$$

$$T_7 : \quad L_1 \neq 0$$

In the Example



$$T_1 : \quad L_0 = 0$$
$$T_2 : \quad L_0 = 0$$
$$T_3 : \quad L_1 = 1$$
$$T_4 : \quad old_1 = new_0$$
$$T_5 : \quad \top$$
$$T_6 : \quad new_0 \neq old_1$$
$$T_7 : \quad L_1 \neq 0$$

$$I_0 : \quad \top$$
$$I_1 : \quad \top$$
$$I_2 : \quad \top$$
$$I_3 : \quad \top$$
$$I_4 : \quad old_1 = new_0$$
$$I_5 : \quad old_1 = new_0$$
$$I_6 : \quad \bot$$
$$I_7 : \quad \bot$$

Permalink: http://logicrunch.it.uu.se:4096/~wv/princess/?ex=perma%2F1614771057_1126223040

Interpolation in SymEx?

- [McMillan, 2010]

Which Theories/Logics admit Interpolation?

Interpolants and Quantifiers

$$A[\bar{x}, \bar{z}] \wedge B[\bar{z}, \bar{y}]$$

- Strongest interpolant: $\exists \bar{x}.\ A[\bar{x}, \bar{z}]$
  Weakest interpolant: $\forall \bar{y}.\ \neg B[\bar{z}, \bar{y}]$

  (where x, y are the local symbols)

Interpolants and Quantifiers

$$A[\bar{x}, \bar{z}] \wedge B[\bar{z}, \bar{y}]$$

- Strongest interpolant: $\exists \bar{x}.\ A[\bar{x}, \bar{z}]$
  Weakest interpolant: $\forall \bar{y}.\ \neg B[\bar{z}, \bar{y}]$

  (where x, y are the local symbols)

- If we allow quantifiers, there are always interpolants!

Interpolants and Quantifiers

$$A[\bar{x}, \bar{z}] \wedge B[\bar{z}, \bar{y}]$$

- Strongest interpolant: $\exists \bar{x}.\ A[\bar{x}, \bar{z}]$
  Weakest interpolant: $\forall \bar{y}.\ \neg B[\bar{z}, \bar{y}]$

  (where x, y are the local symbols)

- If we allow quantifiers, there are always interpolants!
- **When can we compute quantifier-free interpolants?**

Interpolation through QE

- **Observation:**
  Theories that admit **quantifier elimination** also admit quantifier-free interpolation. E.g.
    - Presburger arithmetic
    - Real arithmetic
    - Quantified Boolean logic

Which Theories/Logics admit Interpolation?

Further tools (not discussed)

- MaxSAT/MaxSMT
  - Find maximal satisfiable subsets of a set of inconsistent formulas
- Abduction
  - Suppose $T \models O$ does **not** hold
  - Find explanations *E* such that

$$T \cup E \models O \qquad T \cup E \not\models false$$

- Syntax-guided synthesis

Thank you for your attention!

Questions?