MODEL CHECKING

- Exhaustive exploration of the state-space of a program.
  - If an error state is not reached, then model checking outputs safe.
  - If an error state is reached, then the path to the error state can be reconstructed, resulting in a counterexample.
- Model Checking for sequential programs comes in many variants:
  - Concrete Model Checking
  - Symbolic Model Checking
  - Bounded Model Checking
  - Abstract Model Checking

CONCRETE MODEL CHECKING

```
ConcreteModelChecking(Γ_c,P)
  worklist := {(l_0,σ) | σ ∈ P};
  reach := ∅;
  while worklist ≠ ∅ do{
    Choose (l,σ) ∈ worklist;
    worklist := worklist \ {(l,σ)};
    if ((l,σ) ∉ reach) then
    {
      reach := reach ∪ {(l,σ)};
      foreach ((l,c,l′) ∈ T)
        worklist := worklist ∪ {(l′,σ′) | σ′ ∈ sp({σ},c)};
    }
  }
  if ((l_err,_) ∈ reach) then
    return UNSAFE
  else
    return SAFE
```

CONCRETE MODEL CHECKING

WITH COUNTEREXAMPLE GENERATION

```
ConcreteModelChecking(Γ_c,P)
  worklist := {(l_0,σ) | σ ∈ P}; parents := λx.NR;
  reach := ∅;
  while worklist ≠ ∅ do{
    Choose (l,σ) ∈ worklist;
    worklist := worklist \ {(l,σ)};
    if ((l,σ) ∉ reach) then
    {
      reach := reach ∪ {(l,σ)};
      foreach ((l,c,l′) ∈ T ∧ (l′,σ′) ∈ sp({σ},c)){
        worklist := worklist ∪ {(l′,σ′)};
        parents((l′,σ′)) := (l,σ);
      }
    }
  }
  if ((l_err,_) ∈ reach) then
    return UNSAFE
  else
    return SAFE
```

SYMBOLIC MODEL CHECKING

```
SymbolicModelChecking(Γc,P)
  worklist := {(l0,P)};
  reach(l0) := P;
  foreach (l ∈ L\{l0}) reach(l) := false;
  while worklist ≠ ∅ do{
    Choose (l,F) ∈ worklist;
    worklist := worklist \ {(l,F)};
    if (reach(l) ⇏ F) then
    {
      reach(l) := reach(l) ∨ F;
      foreach ((l,c,l') ∈ T)
        worklist := worklist ∪ {(l',sp(F,c))};
    }
  }
  if (reach(l_err) ≠ false) then
    return UNSAFE
  else
    return SAFE
```

BOUNDED MODEL CHECKING

- Concrete/Symbolic model checking for a finite number of steps
  - Unroll loops in the program for a fixed number of iterations, and then do concrete/symbolic model checking on the resultant program.
- Alternatively, we can apply Static Single Assignment (SSA) transformation on the unrolled program, and directly encode the BMC problem in FOL.

ABSTRACT MODEL CHECKING

- All the previous approaches to model checking have severe limitations:
  - Concrete and Symbolic Model Checking may not terminate and are in general computationally expensive.
  - Bounded Model Checking can only be used to find bugs, and not for verification.
- Let's bring back abstraction!
  - Consider a sound Abstract Interpretation framework $(D, \leq, \alpha, \gamma, \hat{F})$.

ABSTRACT MODEL CHECKING

```
AbstractModelChecking(Γ_c,P)
  worklist := {(l_0,α(P))};
  reach := ∅;
  while worklist ≠ ∅ do{
    Choose (l,d) ∈ worklist;
    worklist := worklist \ {(l,d)};
    if ( ∄(l,d') ∈ reach . d ≤ d') then
    {
      reach := reach ∪ {(l,d)};
      foreach ((l,c,l') ∈ T)
         worklist := worklist ∪ {(l',d') | d' = f̂_c(d)};
    }
  }
  if ((l_err,d) ∈ reach ∧ d ≠ ⊥ ) then
    return UNSAFE
  else
    return SAFE
```

ABSTRACT MODEL CHECKING

WITH COUNTEREXAMPLE GENERATION

```
AbstractModelChecking(Γ_c,P)
  worklist := {(l_0,α(P))}; parents := λx.NR;
  reach := ∅;
  while worklist ≠ ∅ do{
    Choose (l,d) ∈ worklist;
    worklist := worklist \ {(l,d)};
    if ( ∄(l,d') ∈ reach . d ≤ d') then
    {
      reach := reach ∪ {(l,d)};
      foreach ((l,c,l') ∈ T){
        worklist := worklist ∪ {(l',f̂_c(d))};
        parents((l',f̂_c(d))) := (l,d);
      }
    }
  }
  if ((l_err,d) ∈ reach ∧ d ≠ ⊥ ) then
    return UNSAFE
  else
    return SAFE
```

PREDICATE ABSTRACTION

- The predicate abstraction domain is parameterized by a fixed, finite set of predicates P.
  - Each predicate is a formula over the program variables.
  - Example: $P = \{x \leq 1, y = 0, x + y \leq -1\}$
- There are two predicate abstraction domains:
  - Boolean Predicate Abstraction
  - Cartesian Predicate Abstraction

CARTESIAN PREDICATE ABSTRACTION

- The abstract domain is $\mathbb{P}(P) \cup \{ \perp \}$
- The partial order relation $\sqsubseteq$ is defined as follows:
  - $\forall s \in \mathbb{P}(P) \, . \perp \sqsubseteq s$
  - $\forall s_1, s_2 \in \mathbb{P}(P) \, . \, s_1 \sqsubseteq s_2 \Leftrightarrow s_1 \supseteq s_2$
- Top element is $\varnothing$, bottom element is $\perp$
- Example: $P = \{x \leq 1, y = 0, x + y \leq -1\}$. Which of the following are true?
  - $\{x \leq 1\} \sqsubseteq \{x \leq 1, x + y \leq -1\}$
  - $\{x + y \leq = 1, y = 0\} \sqsubseteq \{y = 0\}$
  - $\{x \leq 1\} \sqsubseteq \varnothing$

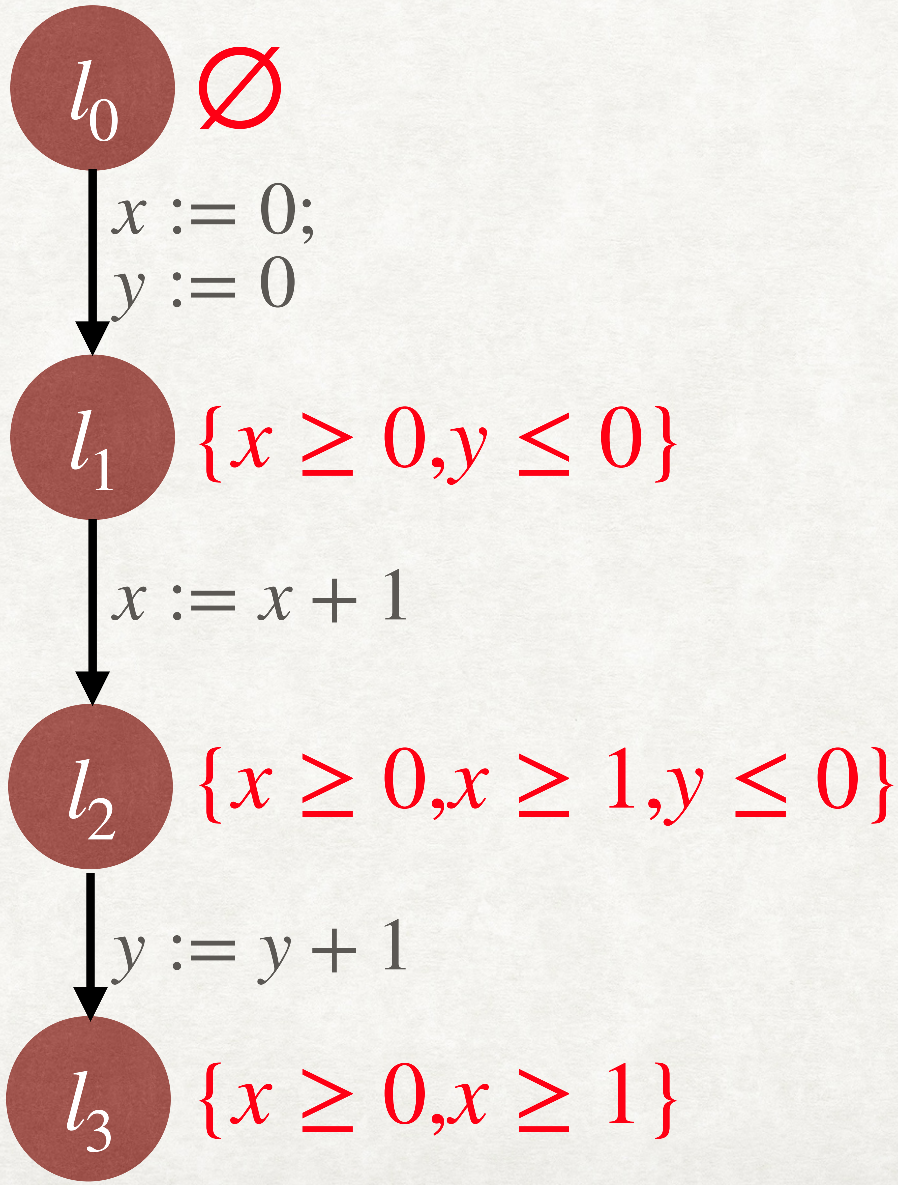
CARTESIAN PREDICATE ABSTRACTION

- Abstraction function: $\forall c \in \mathbb{P}(State) \,.\, c \neq \emptyset \Rightarrow \alpha(c) = \{p \in P \mid \forall \sigma \in c \,.\, \sigma \vDash p\}$
  - $\alpha(\emptyset) = \bot$
- Concretization function: $\forall s \in \mathbb{P}(P) \,.\, \gamma(s) = \{\sigma \mid \sigma \vDash \bigwedge_{p \in s} p\}$
  - $\gamma(\bot) = \emptyset$
- Examples $P = \{x \leq 1, y = 0, x + y \leq -1\}$
  - $\alpha(\{(0,0)\}) = \{x \leq 1, y = 0\}$
  - $\alpha(\{(0,0), (-1, -1)\}) = \{x \leq 1\}$
  - $\alpha(x \leq 0) = \{x \leq 1\}$
- Homework: Prove that $(\mathbb{P}(State), \subseteq) \underset{\gamma}{\overset{\alpha}{\rightleftarrows}} (\mathbb{P}(P) \cup \{\bot\}, \sqsubseteq)$ is an Onto Galois Connection.

ABSTRACT MODEL CHECKING

WITH CARTESIAN PREDICATE ABSTRACTION

$P = \{x \geq 0, y \leq 0, x \geq 1\}$



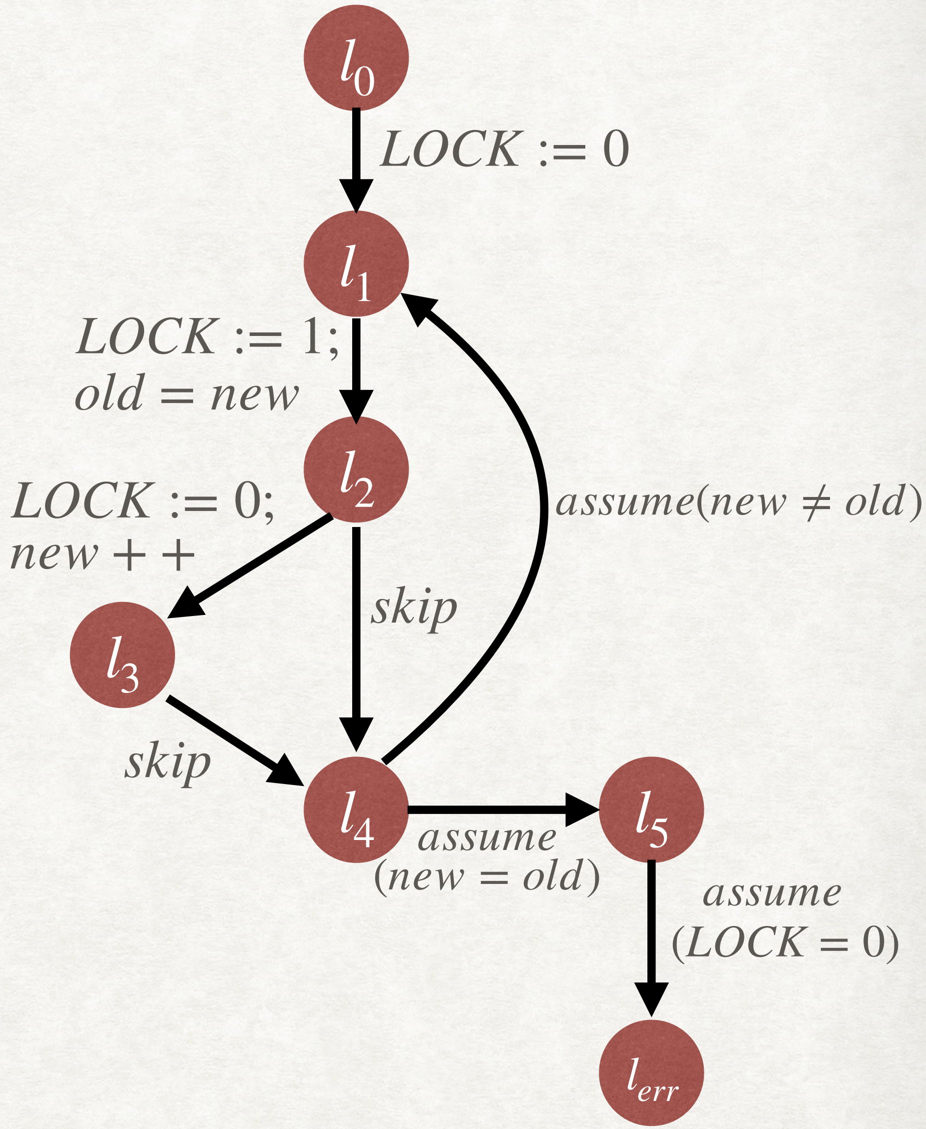
VERIFICATION USING CARTESIAN PREDICATE ABSTRACTION

```
0:   LOCK = 0;
1:   do {
       LOCK = 1;
       old = new;
2:     if (*) {
3:       LOCK = 0;
         new++;
       }
4:   } while (new != old);
5:   if (LOCK==0)
6:     error();
     LOCK = 0;
```

Software Model Checking. Ranjit Jhala and Rupak Majumdar. ACM Computing Surveys(2009)

VERIFICATION USING CARTESIAN PREDICATE ABSTRACTION

```
0:   LOCK = 0;
1:   do {
       LOCK = 1;
       old = new;
2:     if (*) {
3:       LOCK = 0;
         new++;
       }
4:   } while (new != old);
5:   if (LOCK==0)
6:     error();
     LOCK = 0;
```

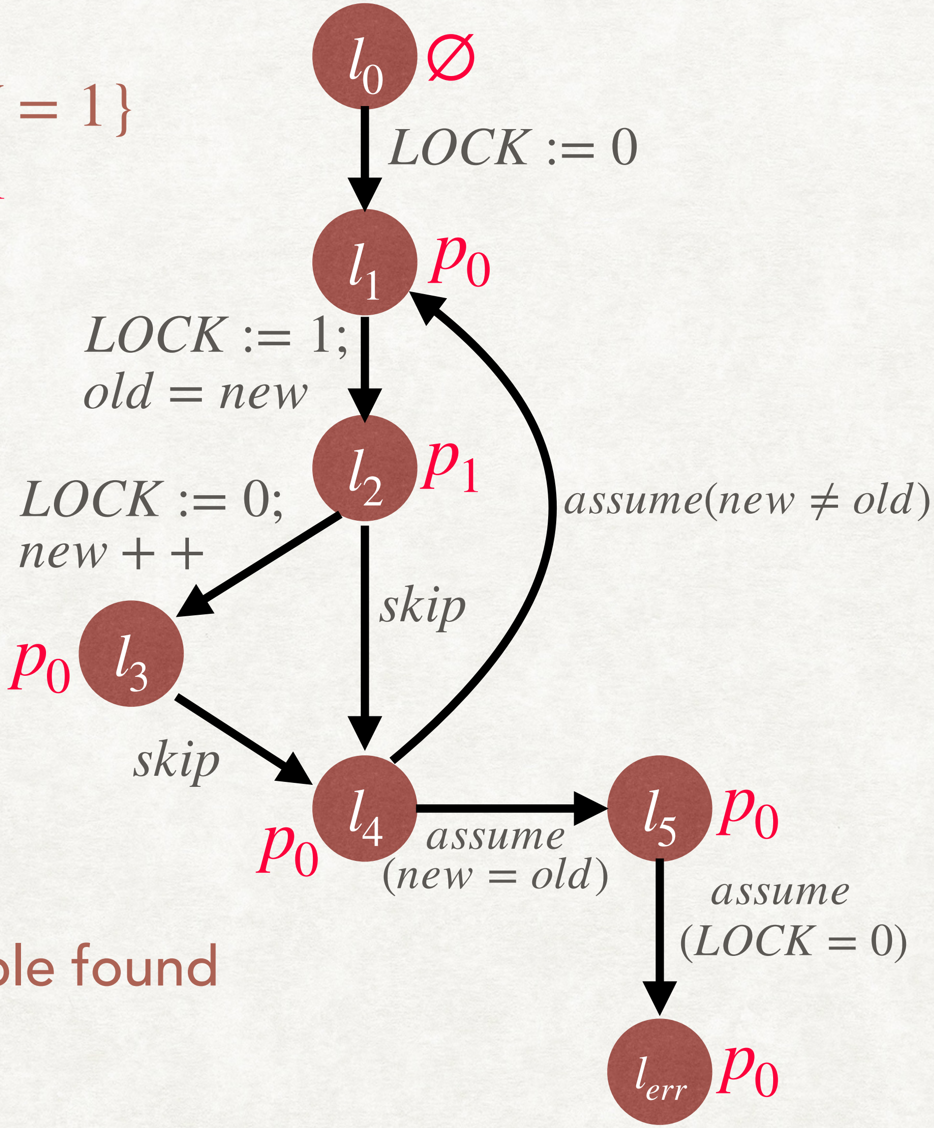


Software Model Checking. Ranjit Jhala and Rupak Majumdar. ACM Computing Surveys(2009)

VERIFICATION USING CARTESIAN PREDICATE ABSTRACTION

$P = \{LOCK = 0, LOCK = 1\}$

p_0 p_1



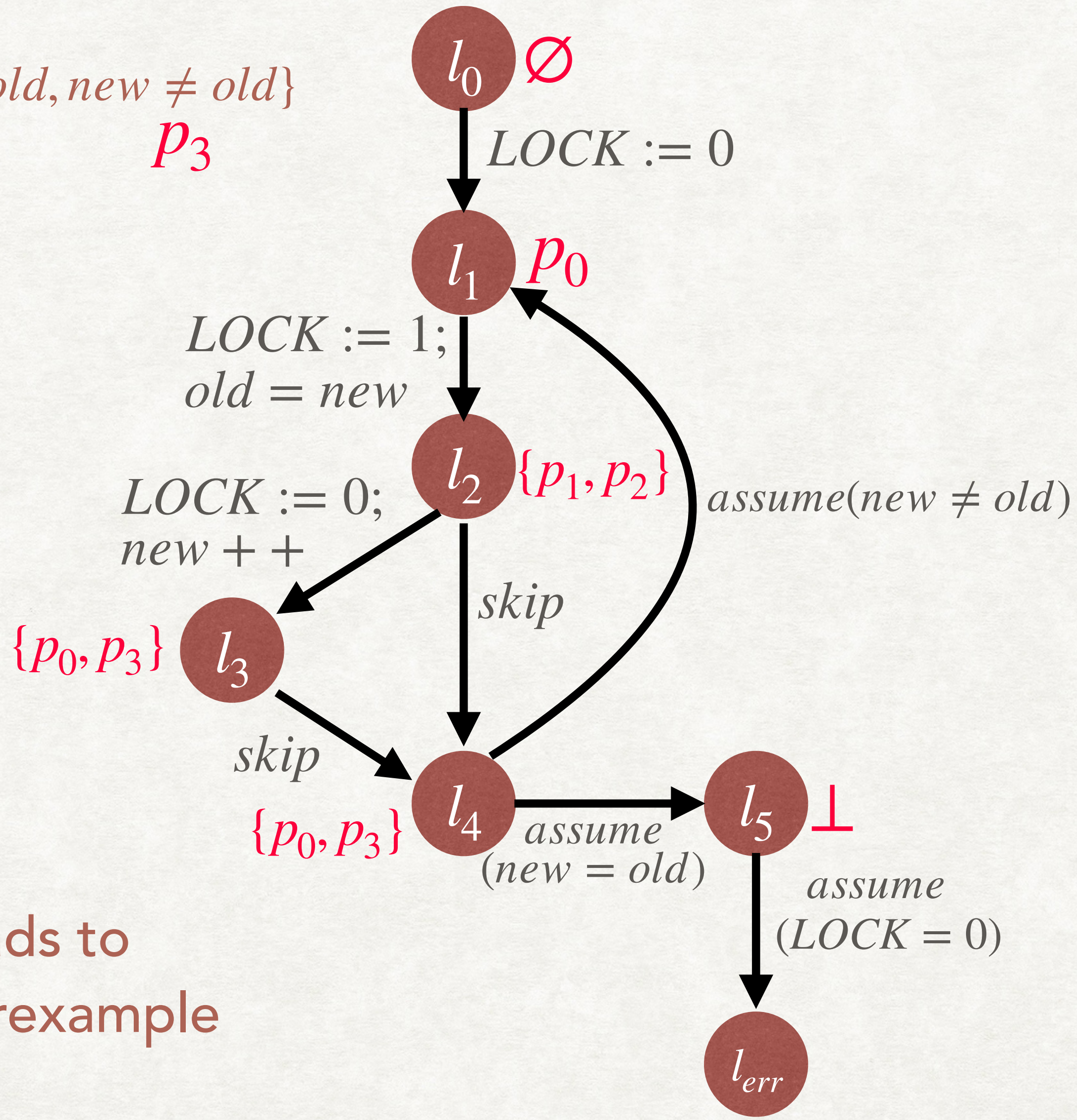
Spurious Counterexample found

Software Model Checking. Ranjit Jhala and Rupak Majumdar. ACM Computing Surveys(2009)

VERIFICATION USING CARTESIAN PREDICATE ABSTRACTION

$P = \{LOCK = 0, LOCK = 1, new = old, new \neq old\}$

p_0 p_1 p_2 p_3



Abstraction Refinement leads to elimination of spurious counterexample

Software Model Checking. Ranjit Jhala and Rupak Majumdar. ACM Computing Surveys(2009)

ABSTRACTION REFINEMENT

- Given two abstract domains $(D_1, \leq_1, \alpha_1, \gamma_1)$ and $(D_2, \leq_2, \alpha_2, \gamma_2)$, we say that D_2 refines D_1 if $\forall c \in \mathbb{P}(State) \,.\, \gamma_2(\alpha_2(c)) \subseteq \gamma_1(\alpha_1(c))$.
- Intuitively, D_2 introduces lower over-approximation during abstraction, leading to more refined abstractions.

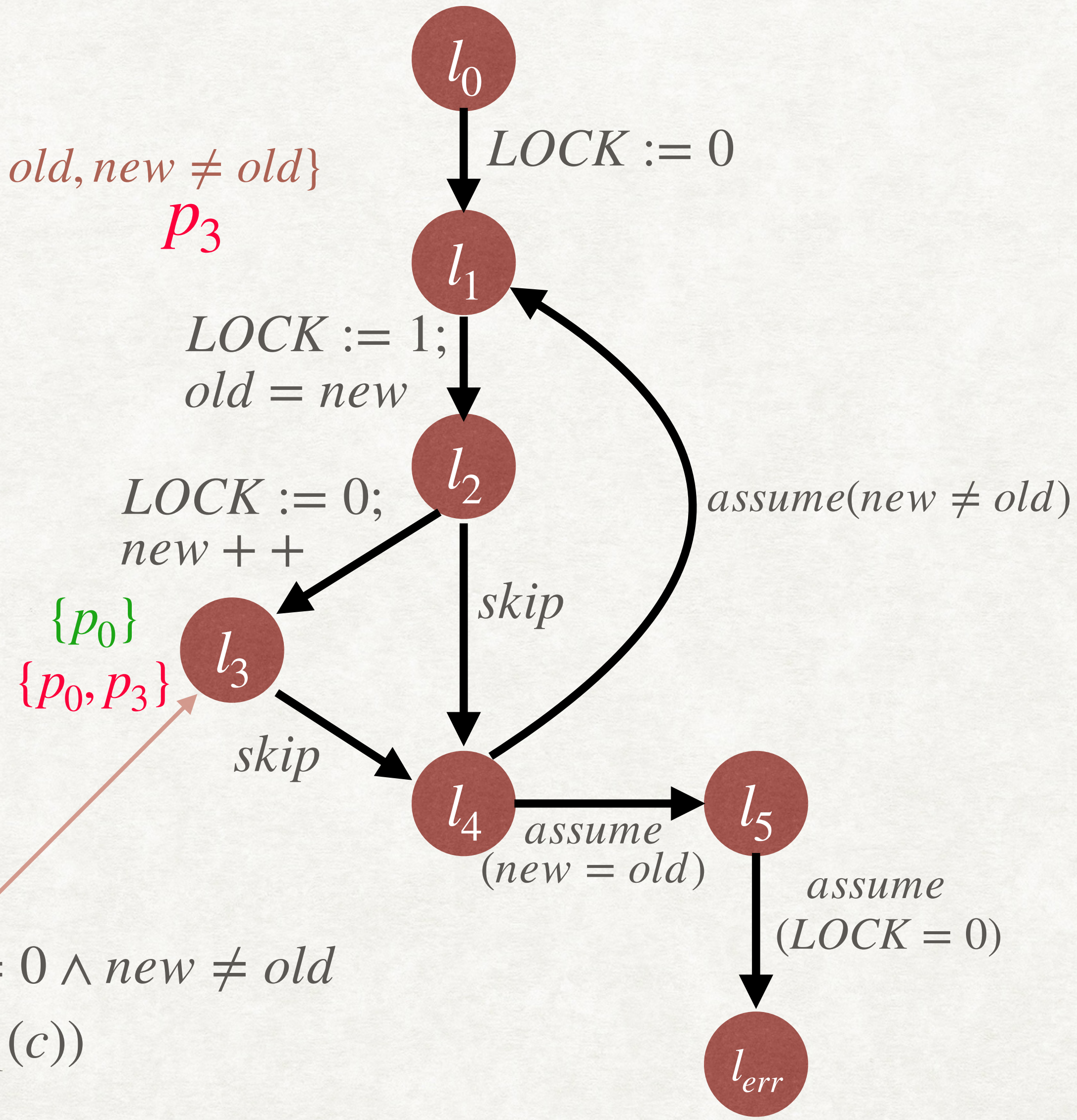
ABSTRACTION REFINEMENT: EXAMPLE

$$\overset{p_0}{}\quad\overset{p_1}{}$$

$P_1 = \{LOCK = 0, LOCK = 1\}$

$P_2 = \{LOCK = 0, LOCK = 1, new = old, new \neq old\}$

$p_0 \quad p_1 \quad p_2 \quad p_3$



Concrete state $c : LOCK = 0 \wedge new \neq old$

$\gamma_2(\alpha_2(c)) \subseteq \gamma_1(\alpha_1(c))$

ABSTRACTION REFINEMENT

- Given two abstract domains $(D_1, \leq_1, \alpha_1, \gamma_1)$ and $(D_2, \leq_2, \alpha_2, \gamma_2)$, we say that D_2 refines D_1 if $\forall c \in \mathbb{P}(State) \,.\, \gamma_2(\alpha_2(c)) \subseteq \gamma_1(\alpha_1(c))$.
- Intuitively, D_2 introduces lower over-approximation during abstraction, leading to more refined abstractions.
- Homework: Given sets of predicates P_1 and P_2 such that $P_1 \subseteq P_2$, prove that the abstract domain $\mathbb{P}(P_2) \cup \{ \bot \}$ refines $\mathbb{P}(P_1) \cup \{ \bot \}$

FINDING REFINEMENTS

- If verification fails with set of predicates P, then we can consider the counterexample, which is a path from the initial location to the error location.
- We can check if the counterexample is valid or spurious.
    - Can be checked by executing the path concretely or symbolically.
- If the counter example is spurious, then we can deduce new predicates which make the counter example infeasible.

TRACE FORMULA

- Given a counterexample $l_{i_0}, l_{i_1}, \ldots, l_{i_n}$ (where $i_0 = 0$ and $i_n = err$), assume that $\forall j \,.\, (l_{i_j}, c_{i_{j+1}}, l_{i_{j+1}}) \in T$. We can symbolically execute the path by constructing its trace formula:

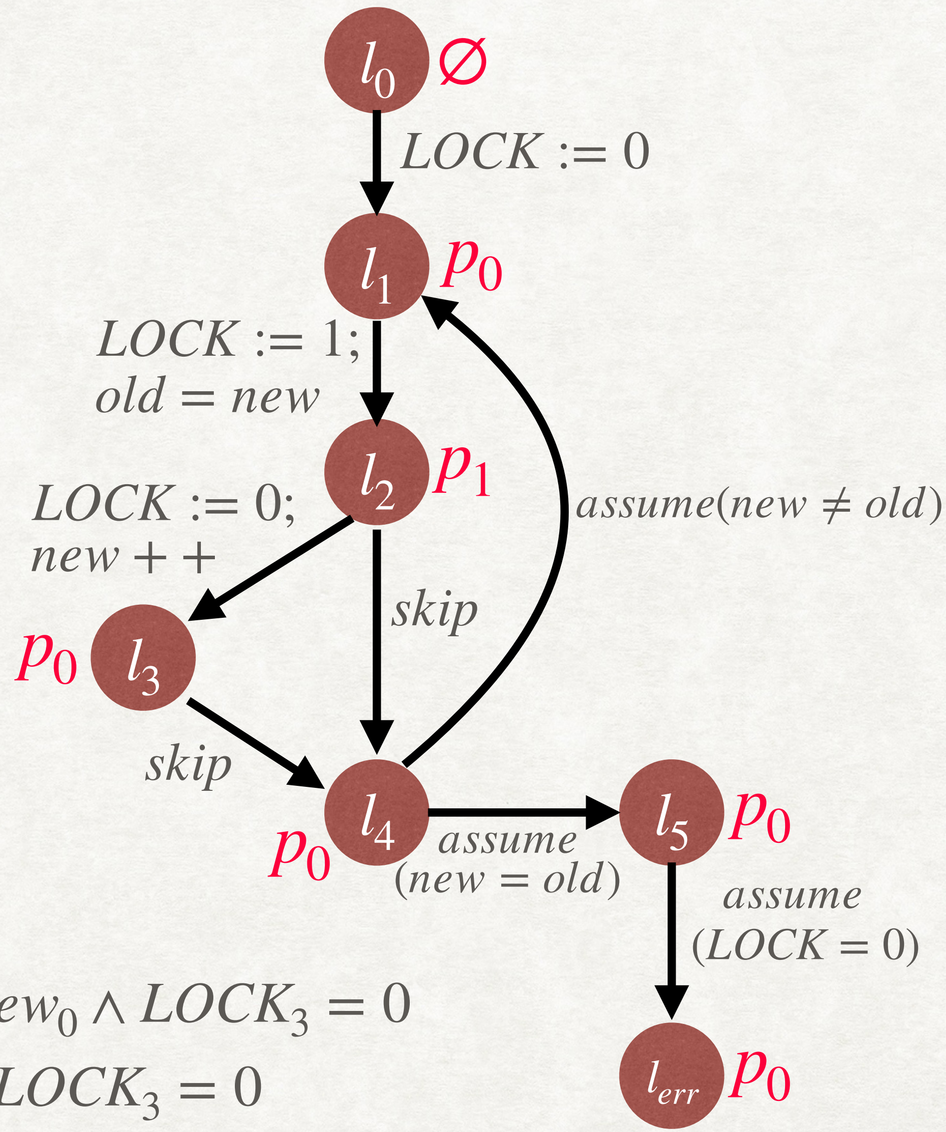
$$\bigwedge_{i=0}^{n-1} \rho(c_{i_{j+1}})[V_{i_j}/V, V_{i_{j+1}}/V']$$

- Here, $\rho(c_{i_j})$ is the encoding of the operational semantics of c_{i_j} in FOL.

TRACE FORMULA : EXAMPLE

$P = \{LOCK = 0, LOCK = 1\}$

p_0 $\quad$ p_1



$LOCK_1 = 0 \wedge LOCK_2 = 1 \wedge old_1 = new_0 \wedge LOCK_3 = 0$

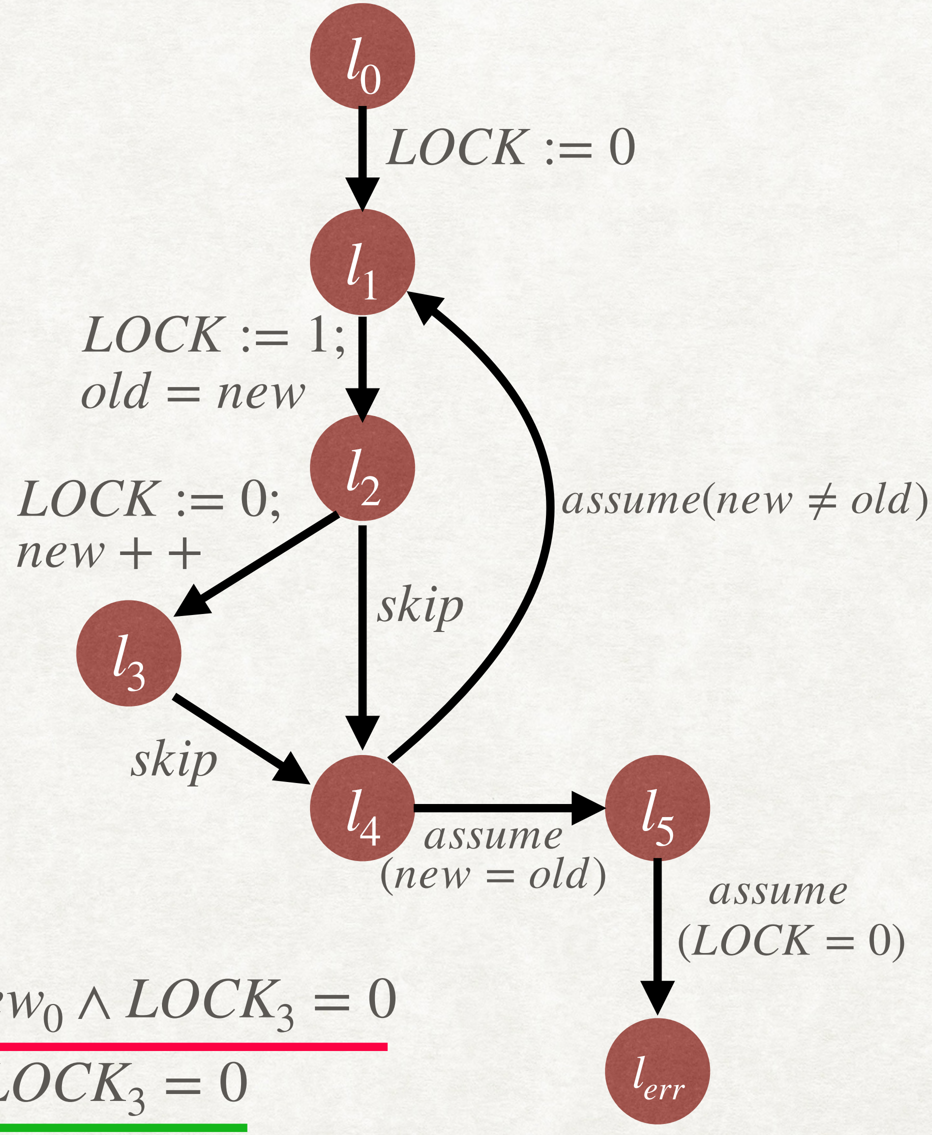
$\wedge\, new_1 = new_0 + 1 \wedge new_1 = old_1 \wedge LOCK_3 = 0$

INTERPOLATION

- Let A and B be formulae such that $A \wedge B$ is unsatisfiable. An interpolant I between A and B is a formula such that
  - $A \Rightarrow I$
  - $I \wedge B$ is unsatisfiable
  - $vars(I) \subseteq vars(A) \cap vars(B)$
- Example
  - $A : x > 0 \wedge x = y \wedge y' = y + 1$
  - $B : y' < 0$
  - $I : y' > 0$
- Craig Interpolation Lemma: An interpolant always exists.
  - Interpolant can be automatically constructed from the proof of unsatisfiability of $A \wedge B$.

INTERPOLANT OF TRACE FORMULA

$I : old_1 \neq new_1$



A

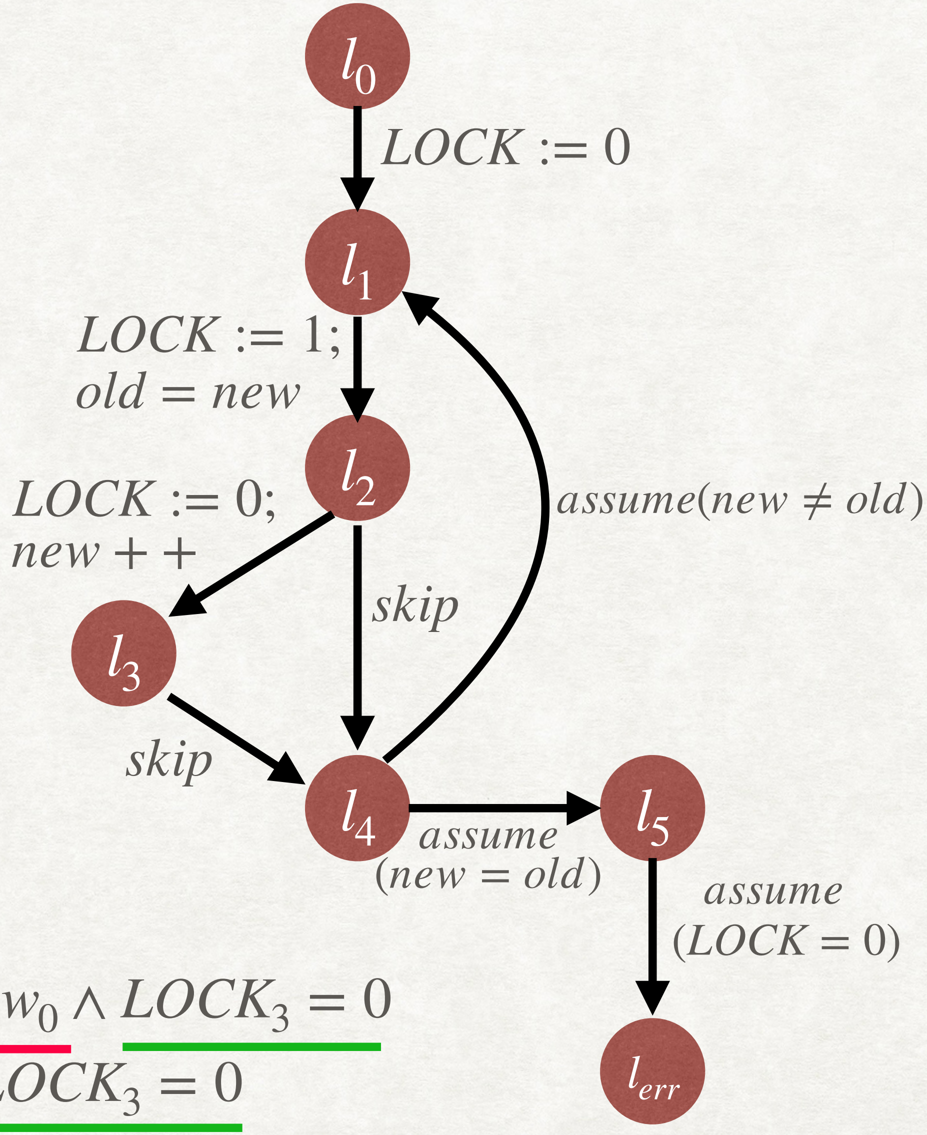
$LOCK_1 = 0 \wedge LOCK_2 = 1 \wedge old_1 = new_0 \wedge LOCK_3 = 0$

$\wedge\, new_1 = new_0 + 1 \wedge new_1 = old_1 \wedge LOCK_3 = 0$

B

INTERPOLANT OF TRACE FORMULA

$$I : old_1 = new_0$$



A

$LOCK_1 = 0 \wedge LOCK_2 = 1 \wedge old_1 = new_0 \wedge LOCK_3 = 0$

$\wedge\, new_1 = new_0 + 1 \wedge new_1 = old_1 \wedge LOCK_3 = 0$

B

INTERPOLANT CHAIN

$$\underbrace{\rho(c_{i_1}) \wedge \ldots \wedge \rho(c_{i_j})}\wedge \underbrace{\rho(c_{i_{j+1}}) \wedge \ldots \wedge \rho(c_{i_n})}$$

Interpolant I_{i_j}:

- Contains states which are reachable after j steps
- Cannot complete the remaining steps
- Variables are in $V_{i_{j+1}}$

We compute chain of interpolants $I_{i_1}, \ldots, I_{i_{n-1}}$

The interpolants would also satisfy the inductiveness condition:
$I_{i_j} \wedge \rho(c_{i_{j+1}}) \Rightarrow I_{i_{j+1}}$

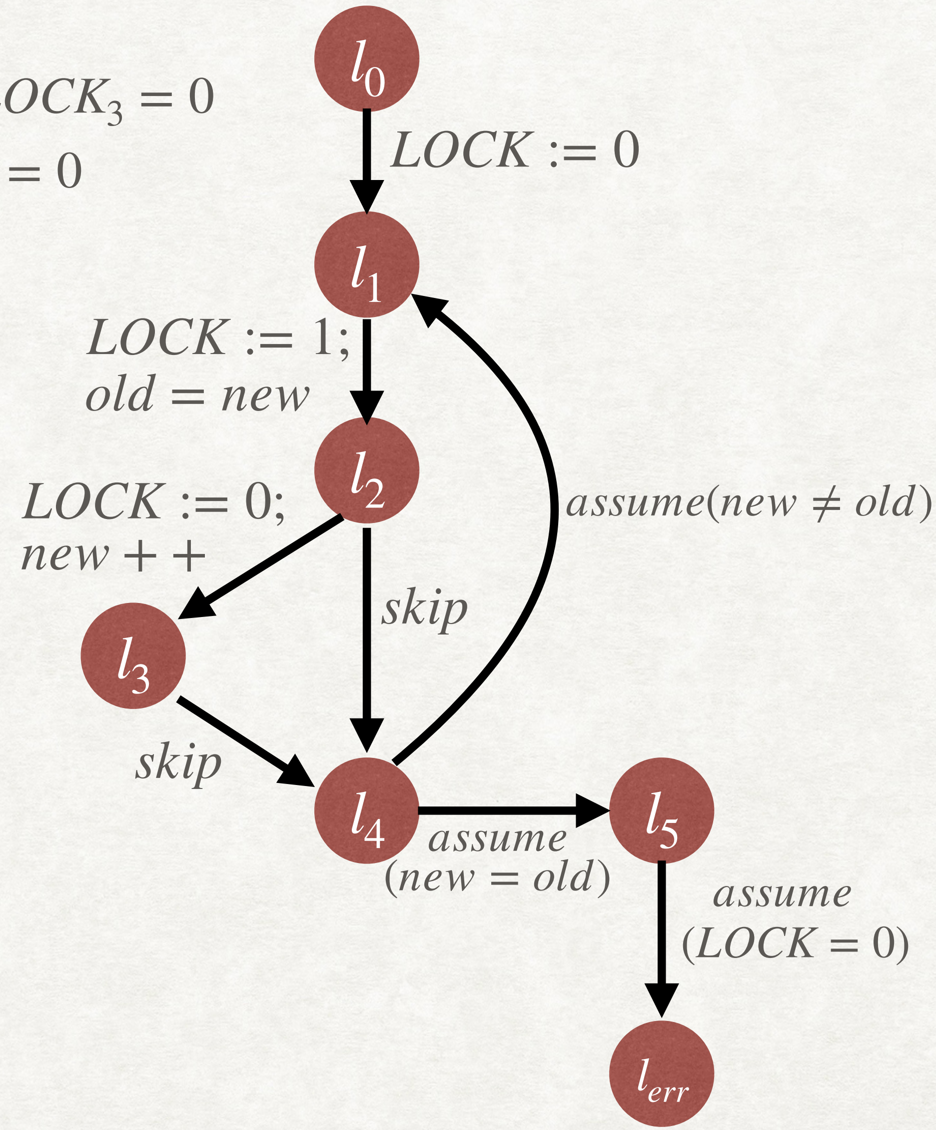
From the interpolant chain, we can now obtain new predicates by removing the subscripts from variable names.

Adding all interpolants from the chain is guaranteed to remove the spurious counterexample.

INTERPOLANT CHAIN:EXAMPLE

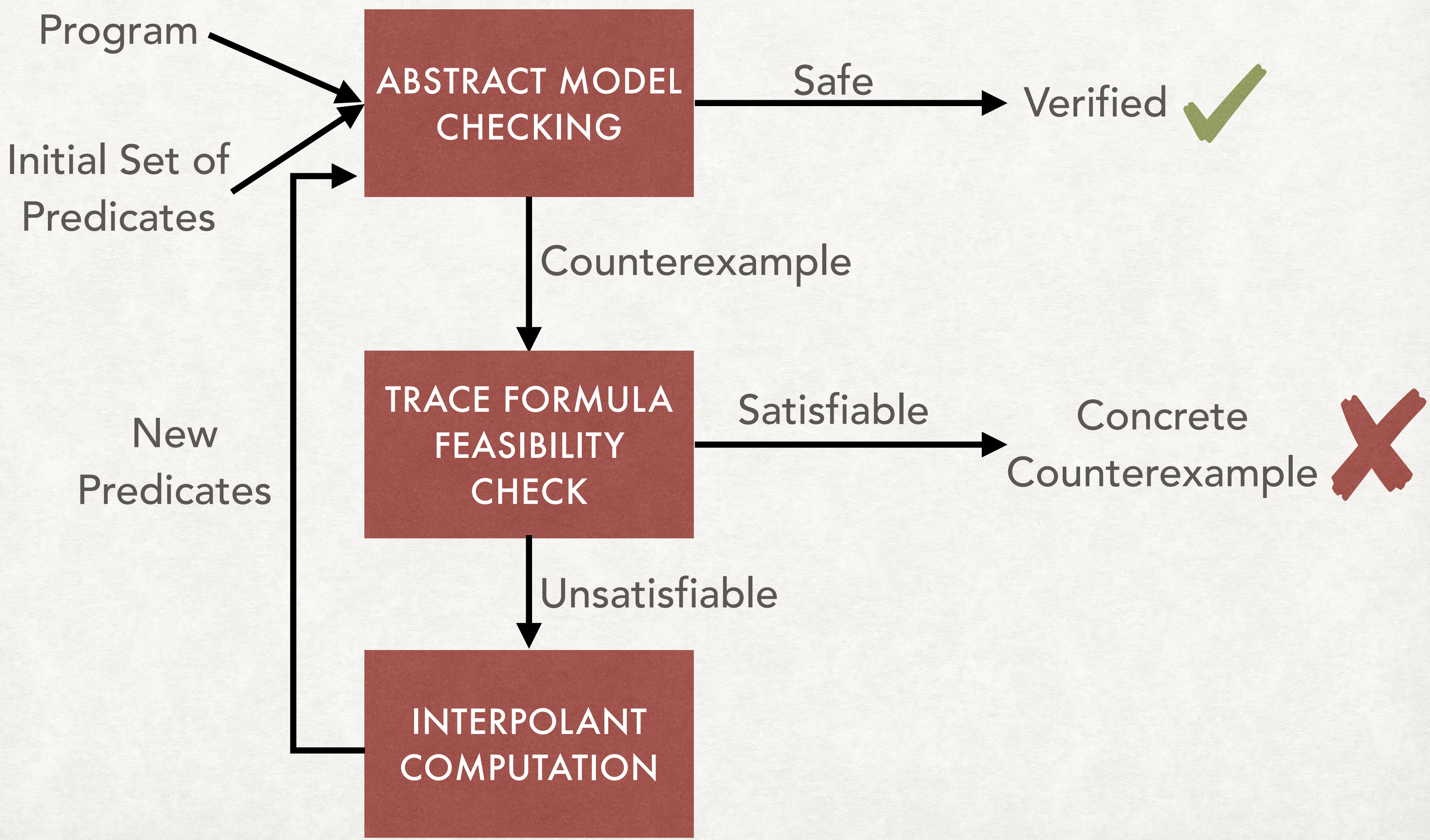
$LOCK_1 = 0 \wedge LOCK_2 = 1 \wedge old_1 = new_0 \wedge LOCK_3 = 0$
$\wedge new_1 = new_0 + 1 \wedge new_1 = old_1 \wedge LOCK_3 = 0$

1	$true$
2	$true$
3	$old_1 = new_0$
4	$old_1 = new_0$
5	$old_1 \neq new_1$
6	$false$



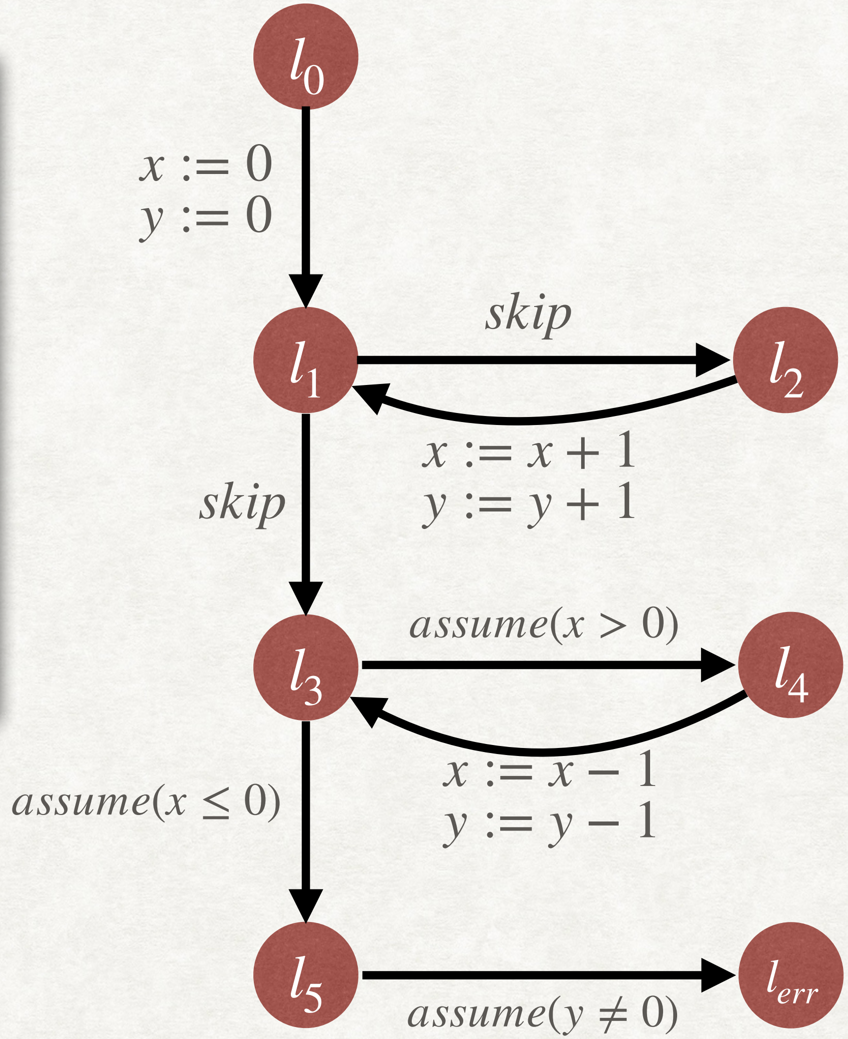
COUNTEREXAMPLE GUIDED ABSTRACTION REFINEMENT

CEGAR



CEGAR: EXAMPLE

```
1:   x = 0;
     y = 0;
2:   while (*){
3:     x++;
       y++;
     }
4:   while (x>0){
5:     x--;
       y--;
     }
6:   if (y != 0) error()
```



CEGAR: EXAMPLE

$P = \{x = 0, y = 0\}$

Counterexample:

$l_0\ l_1\ l_2\ l_1\ l_3\ l_5\ l_{err}$

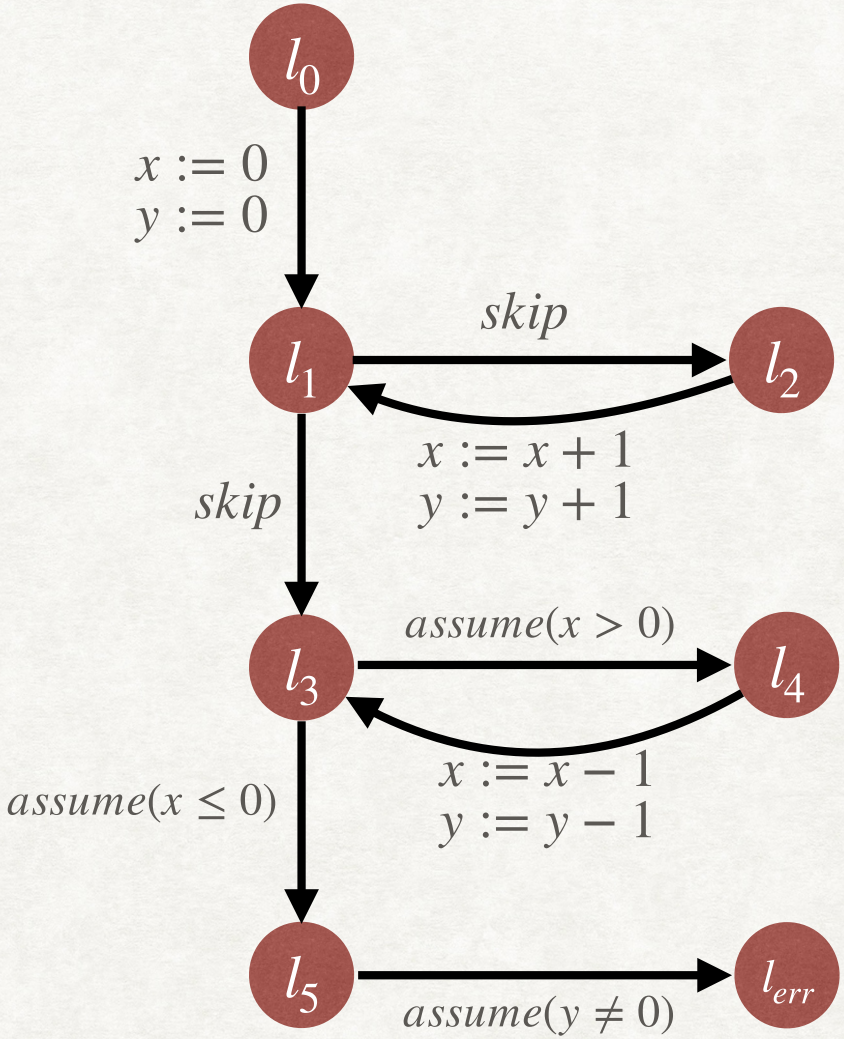
Trace Formula:

$x_1 = 0 \wedge y_1 = 0 \wedge x_2 = x_1 + 1 \wedge$

$y_2 = y_1 + 1 \wedge x_2 \leq 0 \wedge y_2 \neq 0$

Interpolant:

$x_2 > 0$



CEGAR: EXAMPLE

$P = \{y = 0, x \geq 0\}$

Counterexample:

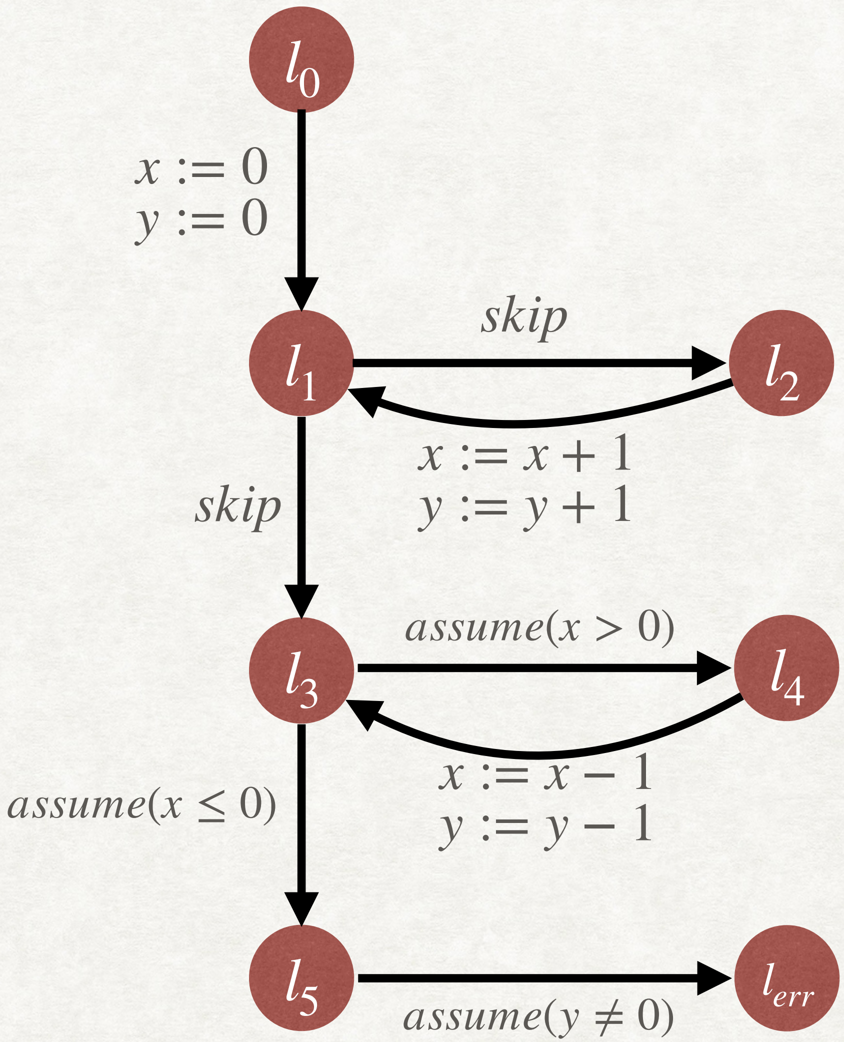
$l_0 \; l_1 \; l_2 \; l_1 \; l_3 \; l_4 \; l_3 \; l_5 \; l_{err}$

Trace Formula:

$x_1 = 0 \land y_1 = 0 \land x_2 = x_1 + 1 \land$
$y_2 = y_1 + 1 \land x_3 = x_2 - 1 \land$
$y_3 = y_2 - 1 \land x_3 \leq 0 \land y_3 \neq 0$

Interpolant:

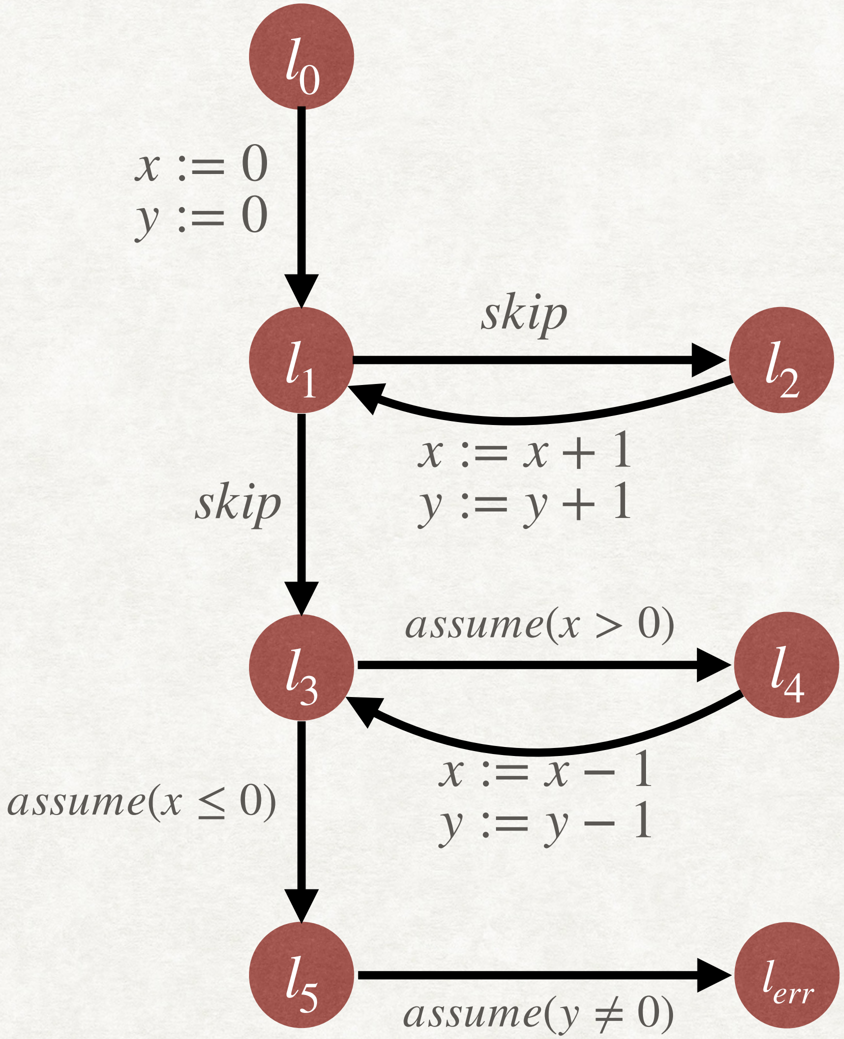
$x_2 > 0 \land x_2 = y_2$



CEGAR: EXAMPLE

$P = \{y = 0, x \geq 0, x = y\}$

Verified ✓



CEGAR: TERMINATION

- The CEGAR procedure may not necessarily terminate, if new predicates keep on getting added by interpolant computation.
- To guarantee termination, we can restrict the language from which predicates are drawn.
- If the safety proof can be expressed in the language of predicates, CEGAR is guaranteed to find it.

BOOLEAN PREDICATE ABSTRACTION

- The domain D is the set of all boolean formulae over the predicates P.
  - The partial order relation is implication $\Rightarrow$.
- The abstraction function maps a set of states c to the smallest boolean formula ϕ (smallest in terms of implication) over P such that each state in c is a model of ϕ.
  - Computing the abstraction of a set of states is exponential in the size of the predicate domain.

COURSE CONCLUSION

CONSTRAINT SOLVERS

- Propositional Logic, SAT solving, DPLL
- First-Order Logic, SMT
- First-Order Theories

↓

DEDUCTIVE VERIFICATION

- Operational Semantics
- Strongest Post-condition, Weakest Pre-condition
- Hoare Logic

↓

MODEL CHECKING AND OTHER VERIFICATION TECHNIQUES

- Predicate Abstraction, CEGAR
- Abstract Interpretation
- Property-directed Reachability
- …

ANNOUNCEMENTS

- Final Exam Dates: ~~Dec 18-20~~ (Yet to be decided)
- Project Presentations: Dec 4
  - 9:30 AM - 12 PM, 1 PM - 2:30 PM
  - 25 Minute slots
- Project Report: Dec 6

THANK YOU

PLEASE GIVE COURSE FEEDBACK