SAT-Based Model Checking Without Unrolling

Aaron R. Bradley

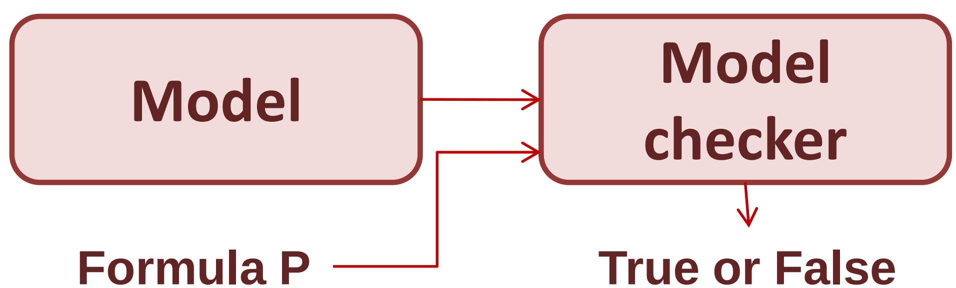
Outline

- Introduction to model checking
- Definitions
- Algorithm
- Conclusion
- References

Introduction (1/2)

Important part of formal verification

- What is model checking?
  - Objective : verify finite state systems formally.
  - Model ***M*** : finite state transition graph
  - Formula ***P*** : specification (temporal logic)
  - Check whether all reachable states *s* of *M* satisfy *P*, denoted ***M,s*** $\models$ ***P***.



Introduction (2/2)

- Types of model checking
  - BDD based
    - Too large for a great amount of variables
    - Variable ordering is time consuming
  - SAT based
    - Bounded Model Checking
    - Unbounded Model Checking
      - Unbounded symbolic model checking
      - Interpolation-based model checking (ITP)
    - SAT-based model checking without unrolling

Outline

- Introduction to model checking
- Definitions
- Algorithm
- Conclusion
- References

Definitions

transition relation

Initial condition

Next state

A set of Boolean variables

- **Finite-state transition system** $S : (\bar{x}, I(\bar{x}), T(\bar{x}, \bar{x}'))$
- **State** s : an assignment of $\bar{x}$.

  $s = x_1 x_2 \neg x_3$

  F-state : S $\models$ **F**
- **Clause** : a disjunction of literals.

  $c = (x_1 + \neg x_3)$

  $d = x_1 \subseteq c$

  Subclause $d \subseteq c$: The set of d's literals is a subset of *c*'s.
- **Trace** $s_0, s_1, s_2, \ldots$: $s_0 \models I$, $s_i, s'_{i+1} \models T$.
- **Safety property** $P(\bar{x})$: for the system S,

  A state that appears in some traces is **reachable.**

  P is S-invariant iff only *P-states* are reachable.

  not invariant iff $\exists$ a trace $s_0, \ldots, s_k$ such that $s_k \not\models P$

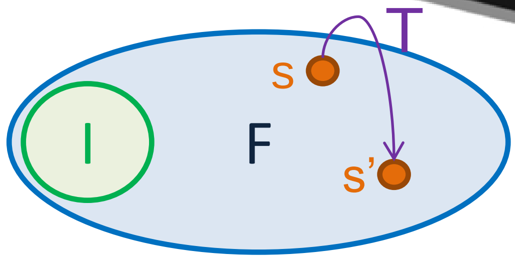
$F(\bar{x})$: formula (can be seen as a set of states with variable $\in \bar{x}$)

Definitions

- $F(\bar{x})$ is ***inductive*** :

  (1) initiation $I \Rightarrow F$

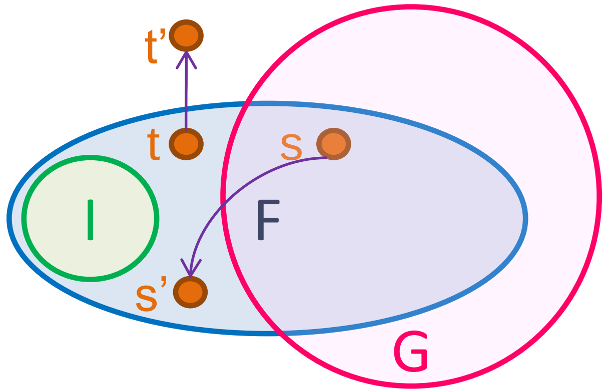
  (2) consecution $F \wedge T \Rightarrow F'$.



- $F(\bar{x})$ is ***inductive relative*** to $G(\bar{x})$:

  (1) $I \Rightarrow F$

  (2) $G \wedge F \wedge T \Rightarrow F'$



- $F(\bar{x})$ is ***inductive strengthening*** of P :

  (1) $I \Rightarrow F \wedge P$

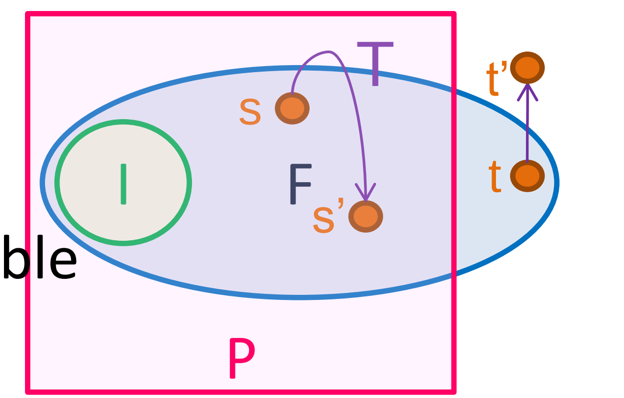
  (2) $F \wedge P \wedge T \Rightarrow F' \wedge P'$

  $I \Rightarrow P$

  $P \wedge T \Rightarrow P'$

  If F is a system ➔ only *P-states* are reachable

  ➔ *P* is F-invariant



Definitions

- ***Inductive generalization*** of a cube *s* : The process to find a **minimal inductive subclause** *d* of $\neg s$ (inductive relative to G)

  (1) $d \subseteq \neg s$

  (2) *d* is *inductive*. $I \Rightarrow d, d \wedge T \Rightarrow d'$

  (3) *d* doesn't contain any *inductive* subclauses.

subclause

inductive

minimal

- How to find d :

  1. *down*: Let d=c_0 = c, check if ($I \Rightarrow d, G \wedge d \wedge T \Rightarrow d'$), d is inductive.

     else let d=c_1 = $c_0 \cap \neg s$ (s : counterexample), iterate the process…

  2. *MIC*: $d \subseteq c$ is inductive but not necessarily minimal.

     Let d_1 = d (drop some literal), and call *down* to reduce its size.

     If it fails, try a different literal until no literal can be dropped.

Example (1/3)

- **Find minimal inductive subclause** $d \subseteq \neg c$
- Assume I = {100}, c = {110}

  Clauses(c) = (xy¬z)

  Clause(¬c) = (¬ x+ ¬y+z)

- *down*

  Let d = ¬c = { 000, 001, 010, 011, 100, 101, 111} = (¬ x+ ¬y+z)

  Check $(I \Rightarrow d) = \neg I + d = (\neg x + y + z) + (\neg x + \neg y + z) = T$

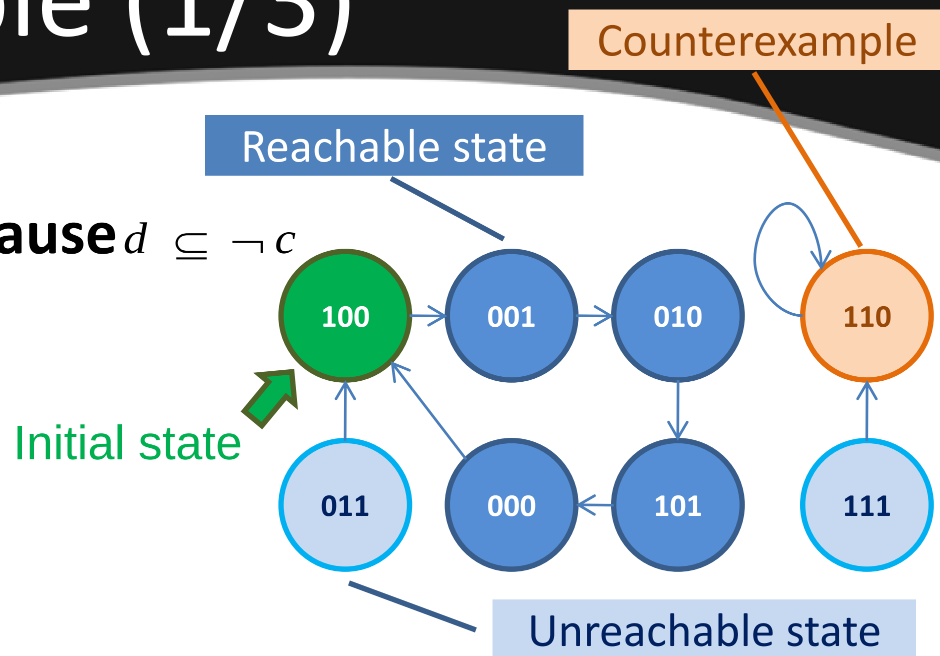
  $(d \wedge T \Rightarrow d') = F \leftarrow$ there is a counterexample s = {111} would exit d

  Over-approximate

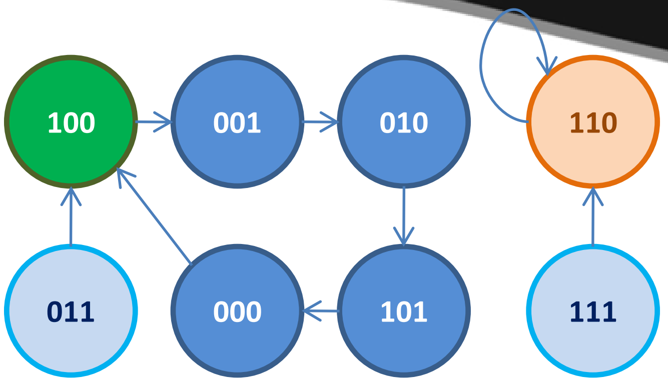
  Let d = d ∩ ¬s = { 000, 001, 010, 011, 100, 101} = (¬ x+ ¬y)

  Check $(I \Rightarrow d) = \neg I + d = (\neg x + y + z) + (\neg x + \neg y) = T$

  $(d \wedge T \Rightarrow d') = T$ *return* d



Example (2/3)



- *MIC*

  Let d_1 = d (drop $\neg$ y) = $\neg$x

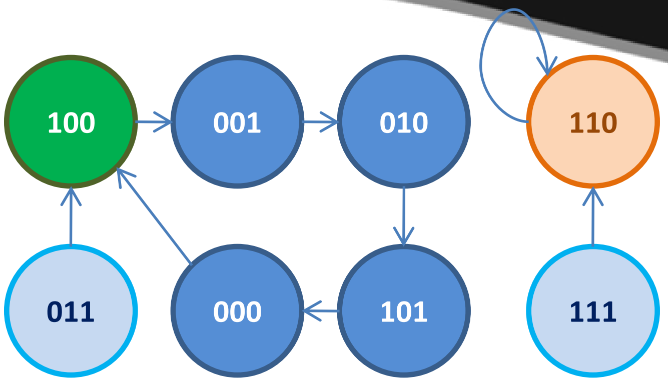
  call *down with Clause($\neg$c) = ($\neg$ x+ $\neg$y+z)*

  Let d = $\neg$x = { 000, 001, 010, 011}

  Check $(I \Rightarrow d) = \neg I + d = (\neg x + y + z) + (\neg x) = F$ $\quad$ *return* False

  $(d \wedge T \Rightarrow d') = F$

Example (3/3)



- *MIC*
- Let d_1 = d (drop $\neg$x) = $\neg$y

  call *down with Clause($\neg$c) = ($\neg$ x+ $\neg$y+z)*

  Let d = $\neg$y = { 000, 001, 100, 101}

  Check $(I \Rightarrow d) = \neg I + d = (\neg x + y + z) + (\neg y) = T$

  $(d \wedge T \Rightarrow d') = F$ $\leftarrow$ there is a counterexa mple s = {001} would exit d

  Let d = d ∩ $\neg$s = { 000, 100, 101} = ($\neg$y)(x+y+$\neg$z) = ($\neg$y)(x+$\neg$z)

  Check $(I \Rightarrow d) = \neg I + d = (\neg x + y + z) + (\neg y)(x + \neg z) = T$

  $(d \wedge T \Rightarrow d') = F$ $\leftarrow$ there is a counterexa mple s = {100} would exit d

  Let d = d ∩ $\neg$s = { 000, 101} ➔ $(I \Rightarrow d) = F$ *return* False
- d = ($\neg$x+ $\neg$y) is the minimal inductive subclause of c.

Outline

- Introduction to model checking
- Definitions
- Algorithm
- Conclusion
- References

Algorithm

- Input: a transition system ***S*** and a safety property ***P***.
- Output: an **inductive strengthening** (if P is S-invariant)

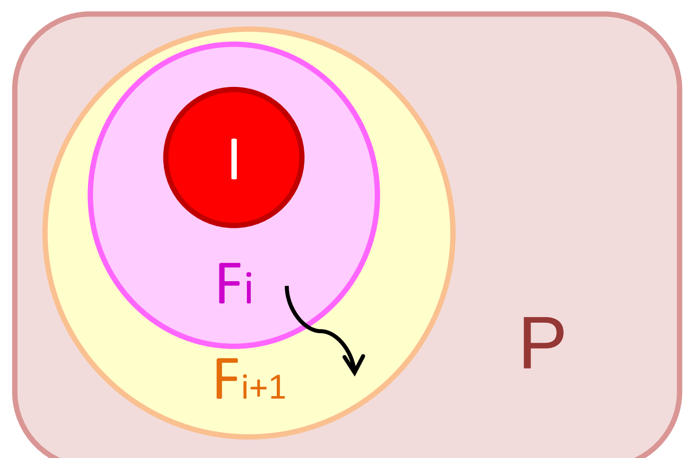
  or a **counterexample trace** (if P is not S-invariant)
- Core logical data structure

  The algorithm incrementally refines and extends a sequence of formulas $F_0=I$, F_1, F_2, . . . , F_k that are over-approximations of the sets of states reachable in at most 0, 1, 2, . . . , k steps. They obeys the following properties:

  - (1) $I \Rightarrow F_0$
  - (2) $F_i \Rightarrow F_{i+1}$

    $\text{clauses}(F_{i+1}) \subseteq \text{clauses}(F_i)$
  - (3) $F_i \Rightarrow P$
  - (4) $F_i \wedge T \Rightarrow F'_{i+1}$

  for $0 \leq i \leq k$,

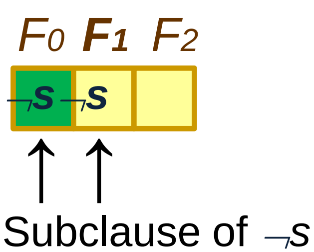


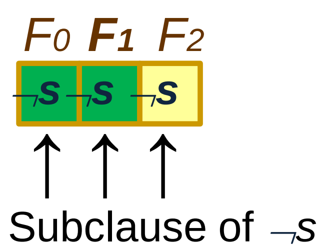
Computational Flow

- $F_0 = I$: the initial state set, check 0 or 1 step violation.
- $F_1 = P$: find 2 step violation from I $\quad F_1 \wedge s \wedge T \Rightarrow \neg P'$

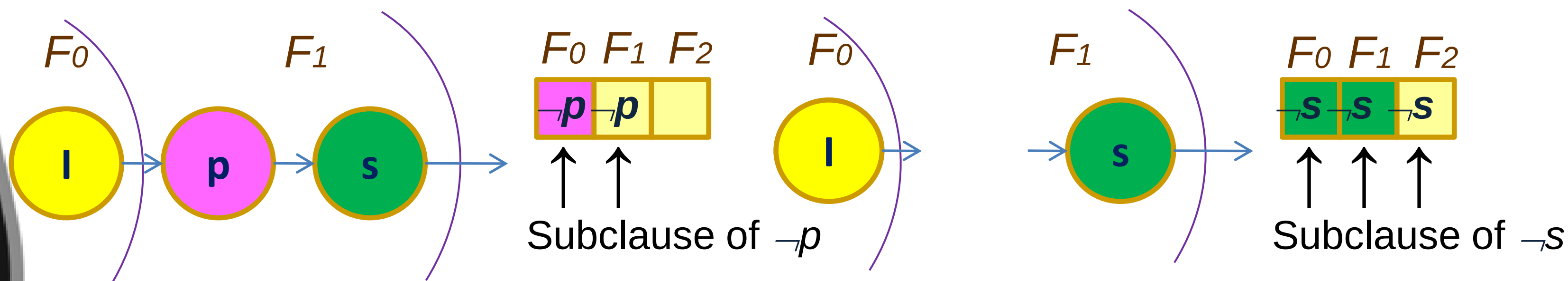
  if $\neg s$ *is* *inductive relative to* F_i ($F_1 \wedge \neg s \wedge T \Rightarrow \neg s'$), eliminate *s* by adding *clauses*($\neg$s) *to record this information.*

Ex. i=0 *i=1*





- Case[i= 0]: $\neg s$ is not inductive relative to F_k, need to push.



Computational Flow

- $F_2 = P$: find 3 step violation from I $F_2 \wedge s \wedge T \Rightarrow \neg P'$ …

- If $F_{k+1} = F_k$, terminate the program and return true.
  (P is inductive strengthening = P is invariant in this system)

Properties of Fi

$$I \Rightarrow F_0$$
$$F_i \Rightarrow F_{i+1}$$
$$F_i \Rightarrow P$$
$$F_i \wedge T \Rightarrow F'_{i+1}$$
$$F_k = F_{k+1}$$

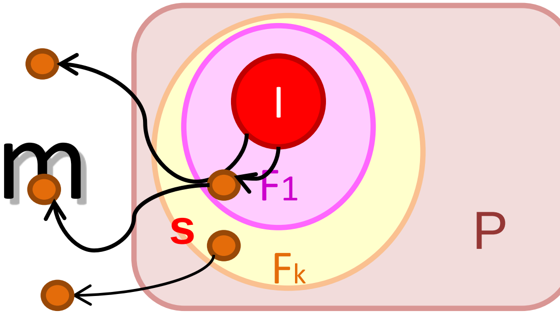
for $0 \leq i \leq k$,

$$I \Rightarrow F_0 \Rightarrow F_1 \Rightarrow \ldots \Rightarrow F_k$$
$$F_k \Rightarrow P$$
$$F_k \wedge T \Rightarrow F_k'$$

Inductive strengthening

$$I \Rightarrow F \wedge P$$
$$F \wedge P \wedge T \Rightarrow F' \wedge P'$$

Block Diagram



Make s inductive relative to F_k, so

Main function:
If P is S-invariant,
return true

prove () {
  obvious violation
  for k = 1 to …
    check(k)
  propagate(k)
}

Ensure $F_{k+1} \Rightarrow P$

check (k) {
  while $F_k \wedge T \Rightarrow \neg P'$
    s = the state lead to violation
    i = inductive(s, k)
    push({i+1, s}, k)
}

F_0~F_i cannot reach s.

inductive (s, k) {
Find $\neg s$ is inductive relative to which F_i (smallest i), then add minimal subclause of $\neg s$ to F_0~F_{i+1}, return i
}

F_{i+1}~F_k cannot reach s.

push ({n, s}, k) {
While(not all n of *states* > k)
choose min n, if $F_n \wedge T \Rightarrow s'$
  p = predecessor of s
  m = inductive(p, k) $p \wedge T \Rightarrow s$
  add {m+1, p} to *states*
else
  m = inductive(s, k)
  states \ {n, s} U {m+1, s} }

propagate (k) {
  for i = 1 to k
    $\forall c \in clauses\ (F_i)$
      if c not in F_{i+1}
      & $F_i \wedge T \not\Rightarrow \neg c'$
      add c to F_{i+1}
  If F_{i+1} = F_i return true }

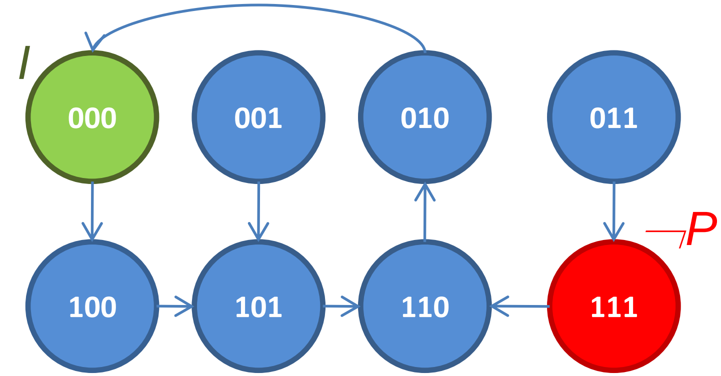
F_0 F_1 … F_i … F_k F_{k+1}

$\neg s$	$\neg s$	$\neg s$	$\neg s$	$\neg s$	s

Subclause of $\neg s$

If i < k-1, F_k still contains s

Example (1/4)



- Example I = {000}, $P = \neg x + \neg y + \neg z$

prove() {

(0 or 1 step violation) // false

initialize // F_0 = {000}, $F_1 = F_2 = F_3 = \ldots =$ P

for // k = 1

check (k=1) {

while (F_1 will go to $\neg P$) // true

s = the state lead to violation // s={111}

i = **inductive (*s*, *k*=1)** {

find $\neg s$ inductive relative to F_0

eliminate s in F_0, F_1 // F_0 = {000}, F_1=P\{111}

} // i=0

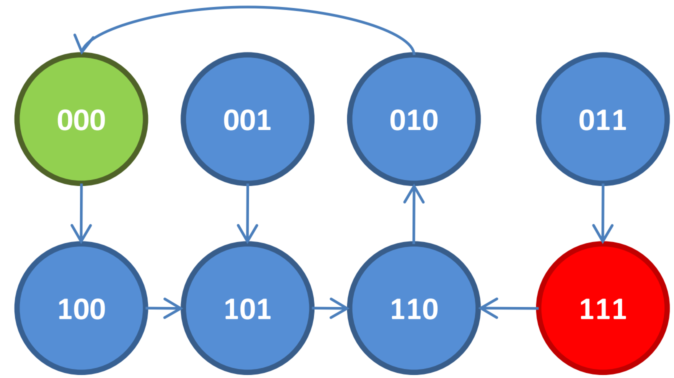
push ({n=1, *s=111*}, k=1)

prove () {
obvious violation
for k = 1 to …
check(k)
propagate(k)
}

check (k) {
while $F_k \wedge T \Rightarrow \neg P'$
s = the state lead to violation
i = inductive(s, k)
push({i+1, s}, k)
}

inductive (s, k) {
Find $\neg s$ is inductive relative to which F_i (smallest i), then add minimal subclause of $\neg s$ to F_0~F_{i+1}, return i
}

Example (2/4)

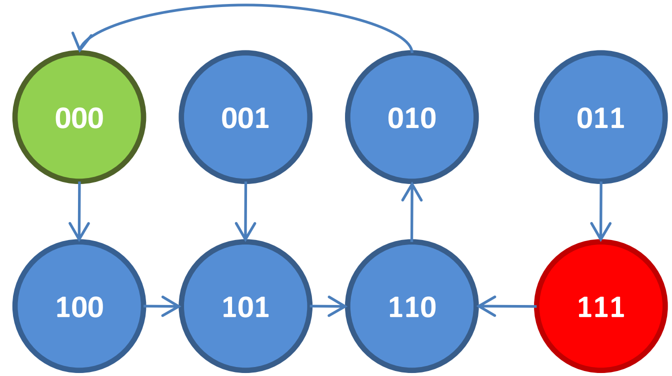


- Example I = {000}, $P = \neg x + \neg y + \neg z$

push ({*n*, *s*}, *k*) {
While(not all n of *states* > *k*)
choose min n, if $F_n \wedge T \Rightarrow s'$
p = predecessor of *s*
m = inductive(*p*, *k*) $p \wedge T \Rightarrow s$
add {*m*+1, *p*} to *states*
else
m = inductive(*s*, *k*)
states \ {n, s} U {*m*+1, *s*} }

push ({*n*=1, *s*=111}, *k*=1) {
while (not all *n* > *k*) { // true, *states* = {1,111}
choose {1, *s*}, if($F_n \wedge T \Rightarrow s'$) // true
p = predecessor of *s* // p = {011}
m = **inductive (*p*, *k*=1)** {
find $\neg p$ inductive relative to F_1
eliminate *s* in F_0, F_1, F_2
// F_0 = {000}, $F_1 = P \backslash \{111,011\}$, $F_2 = P \backslash \{011\}$
} // *m* = 1
add {2, *p*} to *states* // *states* = {1,111}, {2,011}
}

Example (3/4)

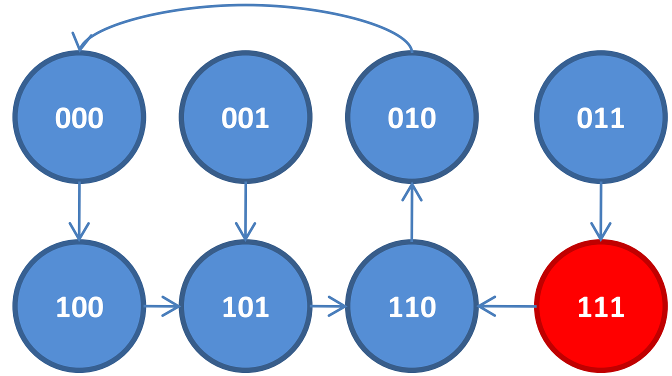


- Example I = {000}, $P = \neg x + \neg y + \neg z$

push ({*n*, *s*}, *k*) {
While(not all n of *states* > *k*)
choose min n, if $F_n \wedge T \Rightarrow s'$
  p = predecessor of *s*
  m = inductive(*p*, *k*) $p \wedge T \Rightarrow s$
  add {*m*+1, *p*} to *states*
else
  m = inductive(*s*, *k*)
  states \ {n, s} U {*m*+1, *s*} }

while (not all *n* > *k*) // true, *states* = {1,111}, {2,011}
  choose {1, *s*}, if($F_n \wedge T \Rightarrow s'$) // false, because F_1=*P*\{111,011}
  m = **inductive (*s*, *k*=1)** {
    find $\neg s$ inductive relative to F_1
    eliminate *s* in F_0, F_1, F_2
    // F_0 = {000}, F_1=*P*\{111,011}, F_2=*P*\{111, 011}
  } // *m* = 1
  states \ {1,*s*} U {2,*s*} // *states* = {2,111}, {2,011}

Example (4/4)



- Example I = {000}, $P = \neg x + \neg y + \neg z$

push ({n, s}, k) {
While(not all n of *states* > k)
choose min n, if $F_n \wedge T \Rightarrow s'$
  p = predecessor of s
  m = inductive(p, k) $p \wedge T \Rightarrow s$
  add {m+1, p} to *states*
else
  m = inductive(s, k)
  states \ {n, s} U {m+1, s} }

while (not all $n > k$) // false, *states* = {2,111}, {2,011}

// exit push

// F_0 = {000}, F_1=P\{111,011}, F_2=P\{111, 011}

// $F_0 = \neg x \neg y \neg z$, $F_1 = \neg y + \neg z$, $F_2 = \neg y + \neg z$

propagate (k) {
  for i = 1 to k
    $\forall c \in clauses(F_i)$
      if c not in F_{i+1}
        & $F_i \wedge T \not\Rightarrow \neg c'$
        add c to F_{i+1}
    If $F_{i+1} = F_i$ return true}

propagate(1) {
  for i = 1
    for c = ($\neg y + \neg z$) if(…) // false
    if $F_{i+1} = F_i$ return *true* //true
}

prove () {
  obvious violation
  for k = 1 to …
    check(k)
  propagate(k)
}

Prove return true ➔ P is invariant ! (It cannot reach $\neg P$ state)

Outline

- Introduction to model checking
- Definitions
- Algorithm
- Conclusion
- References

Conclusion (1/2)

- Performance of ic3 in HWMCC'10
  - Incremental generation of stepwise-relative inductive clauses is a promising new approach to symbolic model checking.
  - It is amenable to simple yet effective parallelization.
- Future work :
  - How inductive clause generation can be used to accelerate finding counterexamples
  - Apply the ideas of stepwise-relative inductive generalization to an infinite-state setting.

Outline

- Introduction to model checking
- Definitions
- Algorithm
- Conclusion
- References

Reference

- Orna Grumberg and Helmut Veith , ***25 years of model checking: history, achievements, perspectives***

Thanks for your attention.