Symbolic Model Checking: The IC3 Algorithm

Shoham Ben-David

IC3/PDR

- Aaron R. Bradley: **SAT-Based Model Checking without Unrolling.** VMCAI 2011
  - A paraphrase on the BMC paper
- "**I**ncremental **C**onstruction of **I**nductive **C**lauses for **I**ndubitable **C**orrectness" : **IC**3
  - Known also as **P**roperty **D**irected **R**eachability
- A very symbolic model checking algorithm
  - Uses SAT solving as a subroutine

From last week:

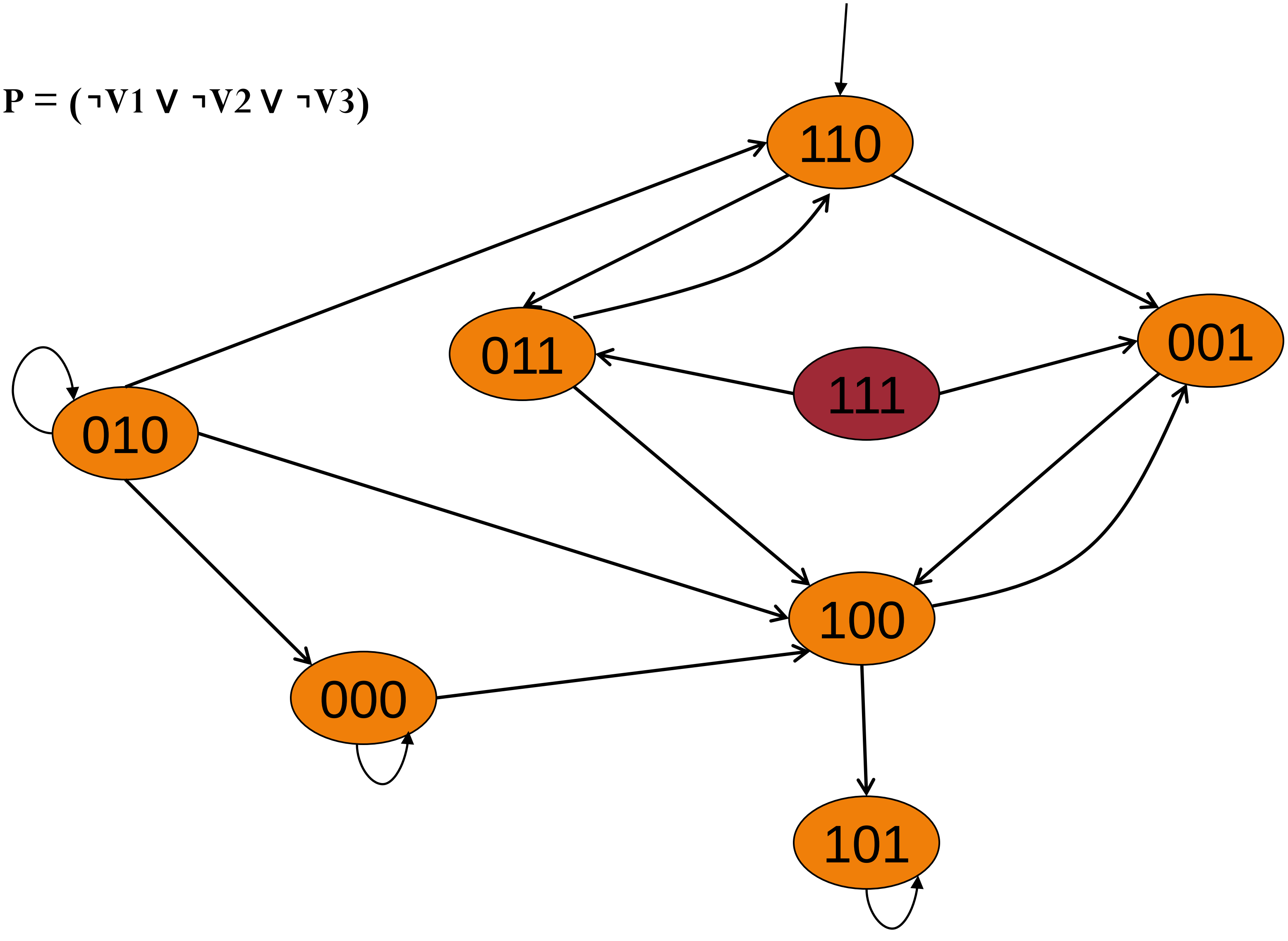
- A model **M** is described by
  - A set **V** of Boolean variables; the state space consists of $2^{|V|}$ states
  - The set **I** of initial states
  - The (total) transition relation **T**
    - Introducing a copy **V`** of **V**, the transition relation T can be represented as a Boolean expression over **V** and **V`**.
- The property **P** is a Boolean expression
- The reachable state space **R** is the set of all states that can be reached from **I** by taking any number of transitions through **T**.

Symbolic Representation

- We view **P** as a set of states
  - All the states that satisfy **P**.
- Suppose that **P** holds in the model ($\mathbf{M} \vDash \mathbf{P}$).
  - It means that $\mathbf{R} \subseteq \mathbf{P}$.
- If we had a Boolean expression representing **R**, we could simply check $\mathbf{R} \Rightarrow \mathbf{P}$
  - by checking satisfiability of $\mathbf{R} \land \neg \mathbf{P}$

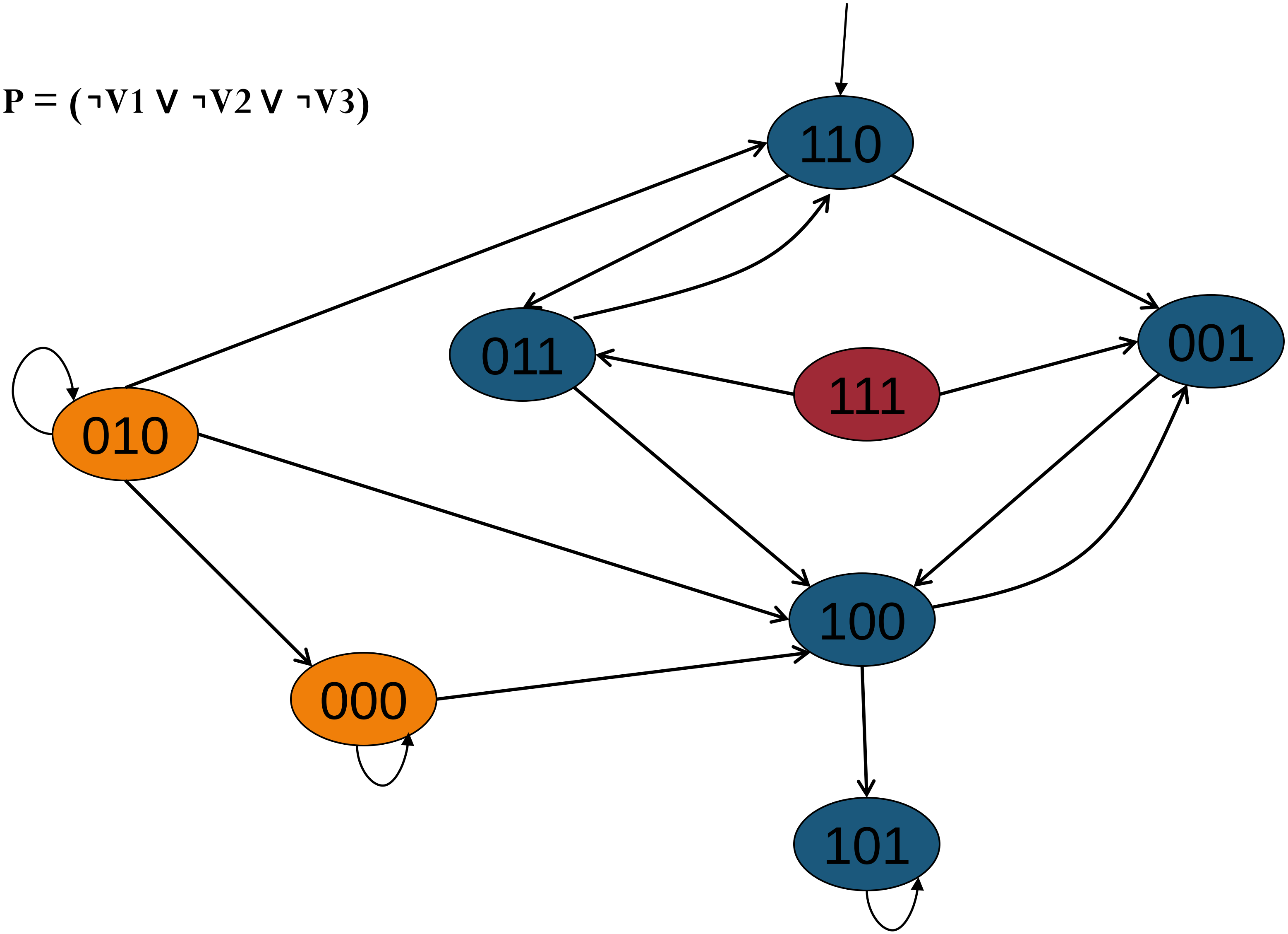
States Satisfying P

P = (¬V1 V ¬V2 V ¬V3)



Reachable States

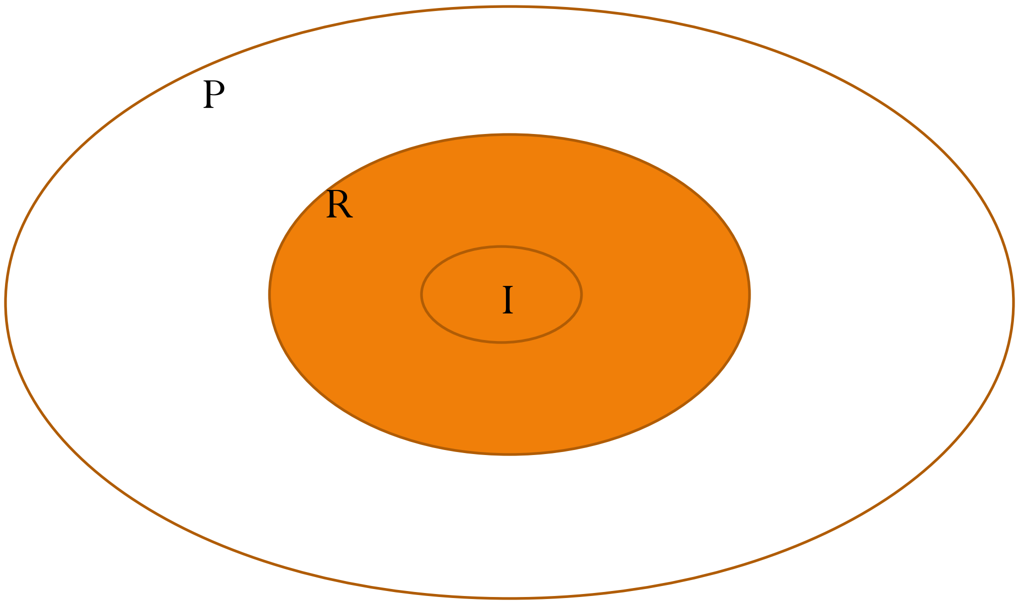
P = (¬V1 ∨ ¬V2 ∨ ¬V3)



Some Observations

- **R** has a special property: $\mathbf{R \wedge T \Rightarrow R'}$
  - If we take a transition from any state in **R**, we shall reach a state in **R**
  - **R** is a 'fix point' for the transition relation
- For a set of states **S**, with $\mathbf{I \subseteq S}$, if S is a fix point, it must include **R**.
  - If $\mathbf{I \subseteq S}$ and $\mathbf{S \wedge T \Rightarrow S'}$ then $\mathbf{R \subseteq S}$
  - **S** is an over-approximation of **R**
- If we find such a set **S**, and show in addition that $\mathbf{S \subseteq P}$, we are done!

An Over Approximation



Searching for an over approximation of R gives us flexibility.
May be easier to find.

The IC3 Algorithm: Main Idea

- Let I be the set of initial states, P the invariant formula
- Build a series of sets

$$I,\ F_1,\ F_2,\ \ldots,\ F_k$$

- Such that
  - For all j, F_j is an over-approximation of the set of states reachable from I in j steps or less.
  - Each F_j satisfies P
- If there exists a j such that $F_j = F_{j+1}$ then a fix point is found, P holds in the model.

Main Idea, More Specificaly

$$I,\ F_1,\ F_2, \ldots,\ F_k$$

- $\forall i,\ F_i \Rightarrow F_{i+1}$
- $\forall i,\ F_i \Rightarrow P$
- $\forall i,\ F_i \wedge T \Rightarrow F`_{i+1}$

Algorithm

- Check $I \Rightarrow P$?
- Check $I \wedge T \Rightarrow P`$?
- Set: $F_1 := P$
- For every clause $c \in$ clauses(I), if $c \notin$ clauses(F_1), check:
  - $I \wedge T \Rightarrow c`$?
  - If it does, set $F_1 := F_1 \wedge c$

A step forward

- Suppose that $I, F_1, F_2, \ldots, F_k$ exist, with the conditions mentioned above.
- Check:

  Is it the case that $F_k \wedge T \Rightarrow P'$?
- If it is, then
  - set $F_{k+1} := P$
  - for every clause $c \in F_k$, check
    - $F_k \wedge T \Rightarrow c'$?
    - If it is, set $F_{k+1} := F_{k+1} \wedge c$
  - If $F_k = F_{k+1}$: done
    - Improvement: compare clauses (syntactic check)

Example 1

- P= (-1,-2,-3)
- I = (1)(2)(-3)
- T=(-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1`,1)(-3,1`,2)

- Step 1: I ⇒ P?
  - I ∧ ¬P = (1)(2)(-3) (1)(2)(3) -- unsatisfiable √
- Step 2: I ∧T ⇒ P`?
  - I ∧T ∧ ¬P` =

(1)(2)(-3) (-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1`,1)(-3,1`,2) (1`)(2`)(3`)

-- unsatisfiable √

Example 1 – Cont.

- P= (-1,-2,-3)
- I = (1)(2)(-3)
- T=(-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1`,1)(-3,1`,2)

- Step 3:
  - Set F_1 := P
  - For every clause c ∈ I, check: I ∧T ⇒ c`?

I ∧T ∧ ¬c` = (1)(2)(-3) (-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1`,1)(-3,1`,2) (-1`)

-- satisfiable for all c ∈ I. Nothing can be added to F_1

Example 1 – Cont.

- P= (-1,-2,-3)
- I = (1)(2)(-3)
- T=(-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1`,1)(-3,1`,2)

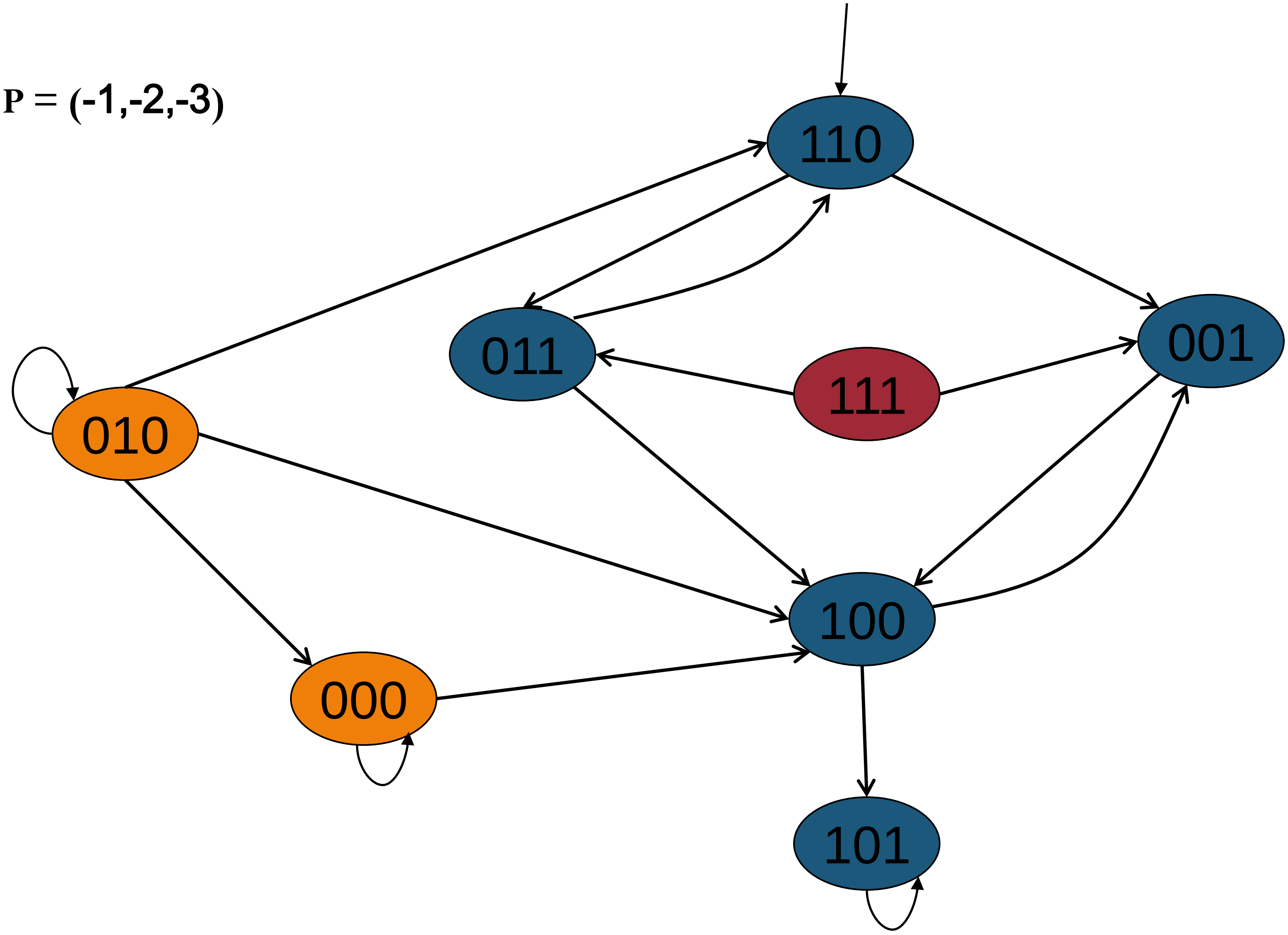
- Step 4: : $F_1 \wedge T \Rightarrow P`$?
  - $F_1 \wedge T \wedge \neg P` =$

    (-1,-2,-3) (-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1,1`)(-3,2,1`) (1`)(2`)(3`)

    -- unsatisfiable √
  - Since F_1 =P we are done !

Example 1

P = (-1,-2,-3)



A step forward – Cont.

- Suppose that $I, F_1, F_2, \ldots, F_k$ exist, with the above conditions.
- Check:

  Is it the case that $F_k \wedge T \Rightarrow P'$?
- If **not** then
  - The SAT solver produces a counterexample, which includes a state $\mathbf{s} \in \mathbf{F_k}$ that is one step away from violating **P**
  - Consider the clause $\neg \mathbf{s}$ (why clause?)
    - Find the maximal **j** such that $\mathbf{F_j \wedge \neg s \wedge T \Rightarrow \neg s'}$
    - (If none exist then P does not hold in M!)
    - Update: $\mathbf{F_i := F_i \wedge \neg s}$ for $\mathbf{0 < i <= j+1}$

IC3: Cont.

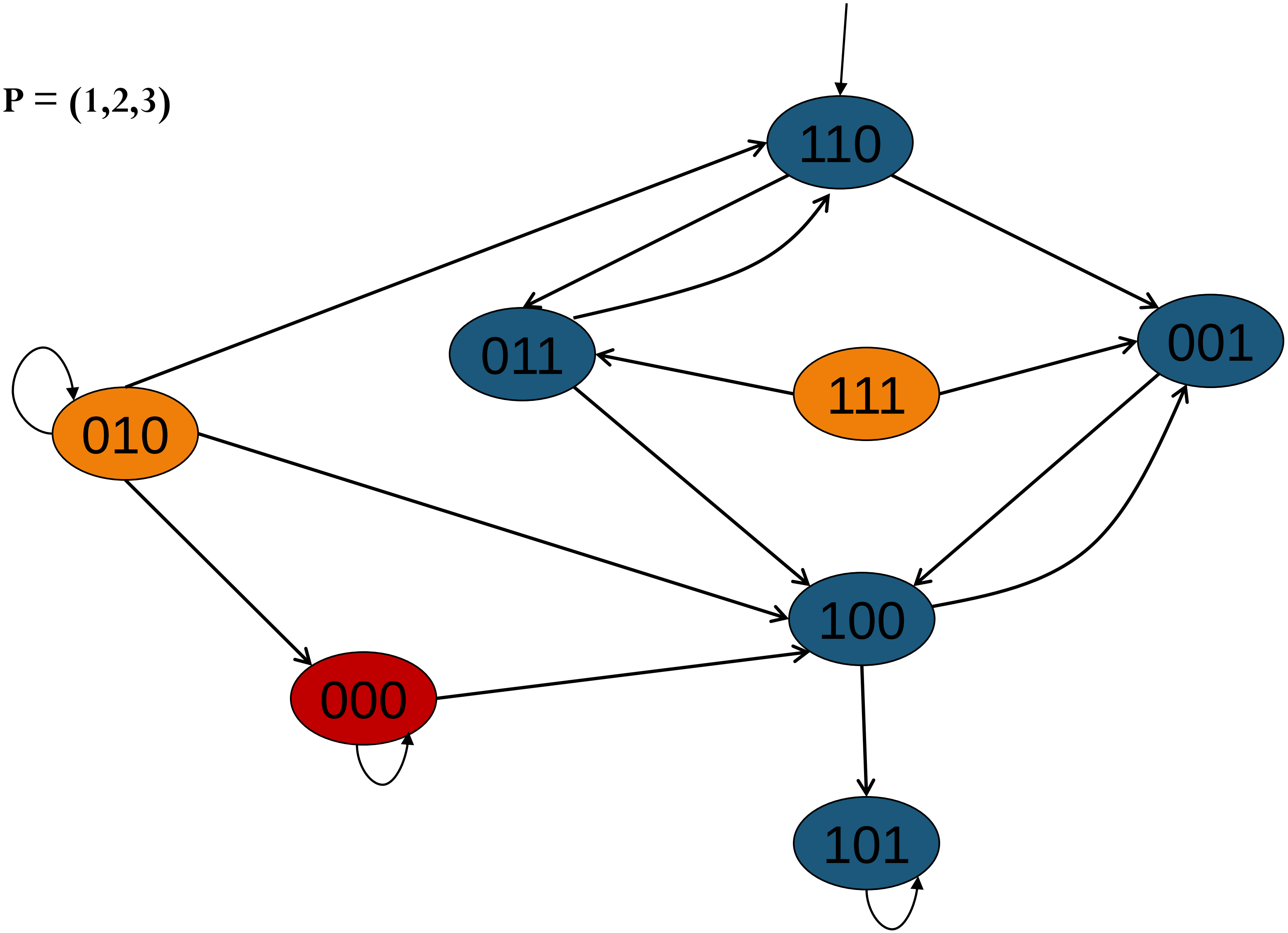
- Check:

  Is it the case that $\mathbf{F_k} \wedge \mathbf{T} \Rightarrow \mathbf{P'}$?

- If **not** then
  - Find a problematic state s and propagate ¬s as far as possible
  - If **¬s** was added to $\mathbf{F_k}$, try again:
    - $\mathbf{F_k} \wedge \mathbf{T} \Rightarrow \mathbf{P'}$?
  - Otherwise
    - find a state **t** that is a predecessor of **s**
    - recur on **t**

Example 2

P = (1,2,3)



Example 2.

- P= (1,2,3)
- I = (1)(2)(-3)
- T=(-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1`,1)(-3,1`,2)

- Step 4: : $F_1 \wedge T \Rightarrow P`$?
  - $F_1 \wedge T \wedge \neg P` =$

 (1,2,3) (-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1,1`)(-3,2,1`) (-1`)(-2`)(-3`)

-- Satisfiable: -1,2,-3,-1`,-2`,-3` is a satisfying assignment

  - ¬P can be reached from s=(-1)(2)(-3)
  - Check: $I \wedge \neg s \Rightarrow \neg s`$?

(1,-2,3)(1),(2),(3) (-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1,1`)(-3,2,1`) (-1`),(2`),(-3`)

  - Unsatisfiable!
  - Set : $F_1 := F_1 \wedge \neg s$ = (1,2,3) (1,-2,3)

Example 2: Cont.

- Recheck: $F_1 \wedge T \Rightarrow P`$?
  - $F_1 \wedge T \wedge \neg P` =$

    (1,2,3) (1,-2,3)(-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1,1`)(-3,2,1`) (-1`)(-2`)(-3`)

    -- Unsatisfiable
  - Set $F_2 := P$
  - Check: $F_1 \wedge \neg s \Rightarrow \neg s`$?

    (1,-2,3)(1,2,3) (-1,3`)(1,-3`)(2,-2`)(-1,-2,-1`)(-3,1,1`)(-3,2,1`) (-1`),(2`),(-3`)
  - Unsatisfiable
  - Set : $F_2 := F_2 \wedge \neg s$
  - But $F_1 = F_2$!
  - A fix point is found. P holds in the model.

IC3 summary

- A combination of induction, over-approximation and SAT solving
- Instead of a "Black Box" use of SAT: make SAT solving an integral part of the procedure
  - Many small SAT problems to solve (10,000 and more)
- As of today:
  - State-of-the-art symbolic model checking algorithm
  - Many (improved) implementations exist