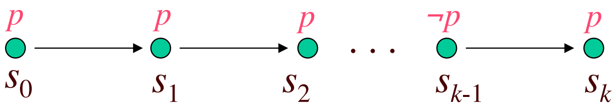
Formulation of famous problems as SAT: *Bounded Model Checking* (1/4)

Given a property p: (e.g. "*always* signal_a = signal_b")

Is there a state reachable within k cycles, which satisfies $\neg p$?



Formulation of famous problems as SAT: *Bounded Model Checking* (2/4)

The reachable states in k steps are captured by:

$$I(s_0) \wedge \rho(s_0, s_1) \wedge \rho(s_1, s_2) \wedge \ldots \wedge \rho(s_{k-1}, s_k)$$

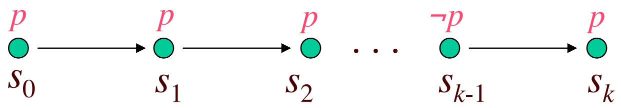
The property p fails in one of the cycles $1..k$:

$$\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_k$$

Formulation of famous problems as SAT: *Bounded Model Checking* (3/4)

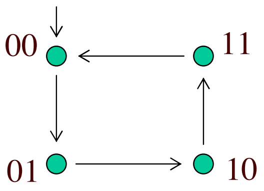
The safety property p is valid up to cycle k iff $\Omega(k)$ is unsatisfiable*:*

$$\Omega(k): \quad I_0 \wedge \bigwedge_{i=0}^{k-1} \rho(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p_i$$



Formulation of famous problems as SAT: *Bounded Model Checking* (4/4)

Example: a two bit counter



Initial state: $I_0: \neg l \wedge \neg r$

Transition: $\rho: \begin{array}{l} l' = (l \neq r) \\ r' = \neg r \end{array}$

Property: always $(\neg l \vee \neg r)$.

For $k = 2$:

$$\varphi: \; (\neg l_0 \wedge \neg r_0) \wedge \left(\begin{array}{l} l_1 = (l_0 \neq r_0) \wedge r_1 = \neg r_0 \wedge \\ l_2 = (l_1 \neq r_1) \wedge r_2 = \neg r_1 \end{array} \right) \wedge \left(\begin{array}{l} (l_0 \wedge r_0) \vee \\ (l_1 \wedge r_1) \vee \\ (l_2 \wedge r_2) \end{array} \right)$$

For $k = 2$, $\Omega(k)$ is unsatisfiable. For $k = 4$ $\Omega(k)$ is satisfiable

What is SAT?

Given a propositional formula in CNF, find an assignment to Boolean variables that makes the formula true:

$$\omega_1 = (x_2 \vee x_3)$$

$$\omega_2 = (\neg x_1 \vee \neg x_4)$$

$$\omega_3 = (\neg x_2 \vee x_4)$$

$$A = \{x_1=0, x_2=1, x_3=0, x_4=1\}$$

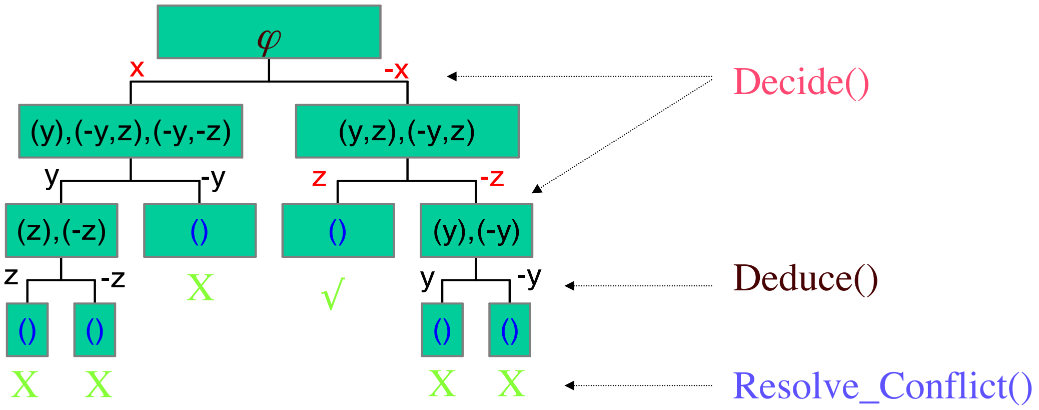
SATisfying assignment!

Why SAT?

- Fundamental problem from theoretical point of view
- Numerous applications:
  - CAD, VLSI
  - Optimization
  - Bounded Model Checking and other type of formal verification
  - AI, planning, automated deduction

A Basic SAT algorithm

Given φ in CNF: (x,y,z),(-x,y),(-y,z),(-x,-y,-z)



Decide()

Deduce()

Resolve_Conflict()

A Basic SAT algorithm

```
While (true)
{
        if (!Decide()) return (SAT);
        while (!Deduce())
                if (!Resolve_Conflict()) return (UNSAT);
}
```

Choose the next variable and value. Return False if all variables are assigned

Apply unit clause rule. Return False if reached a conflict

Backtrack until no conflict. Return False if impossible