SAT for Software Model Checking
Introduction to SAT-problem for newbie

Moonzoo Kim
(original slides from Changbeom Choi)

Content

- Motivation
- Model Checking as a SAT problem
- SAT & SAT-solver?
- Discussion

Model Checking

Requirements

↓

Requirement Properties

$\Box(\Phi \rightarrow \Diamond \Omega)$

Target Program

↓

System Model

Model Checker

Satisfied

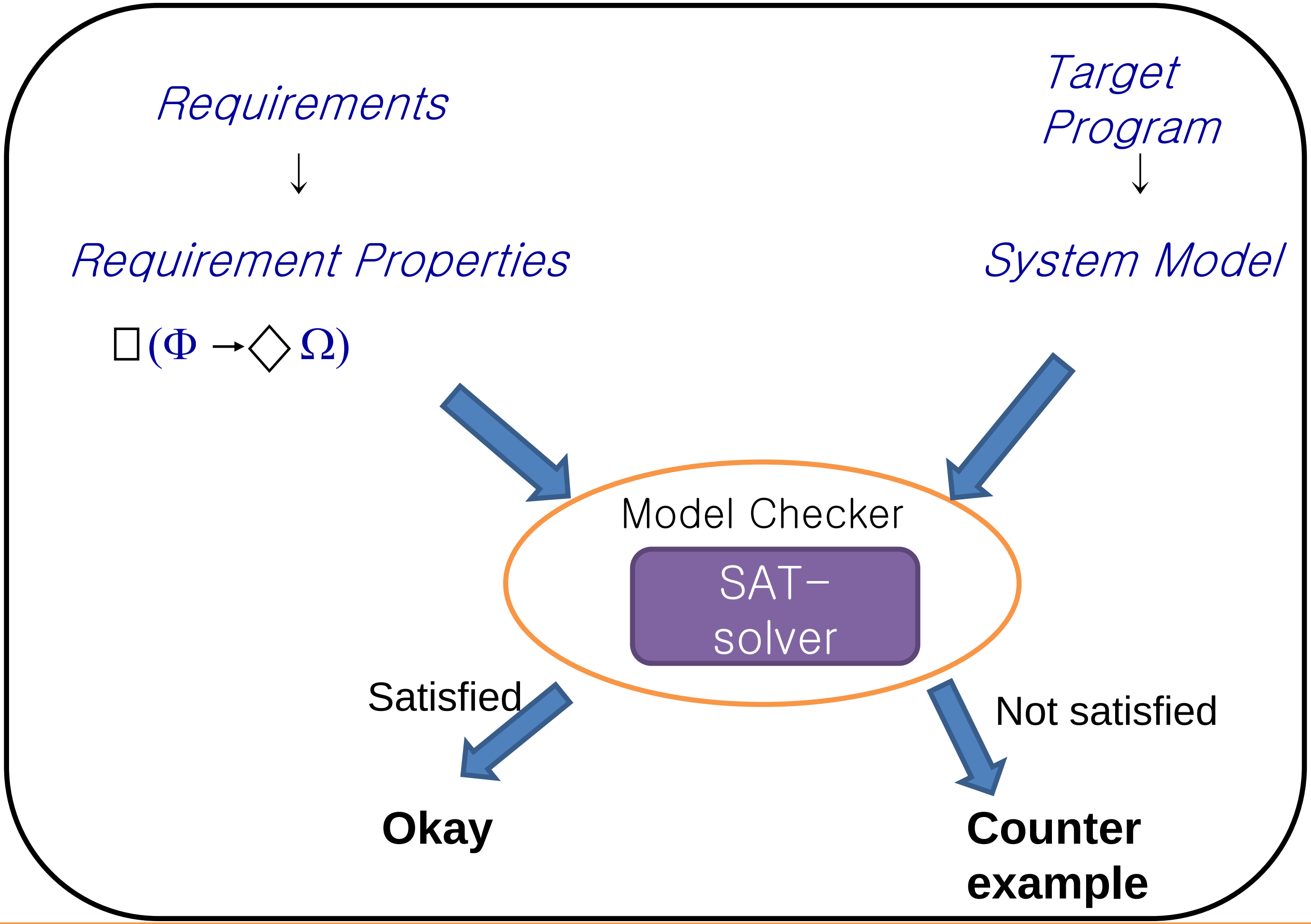
Okay

Not satisfied

Counter example

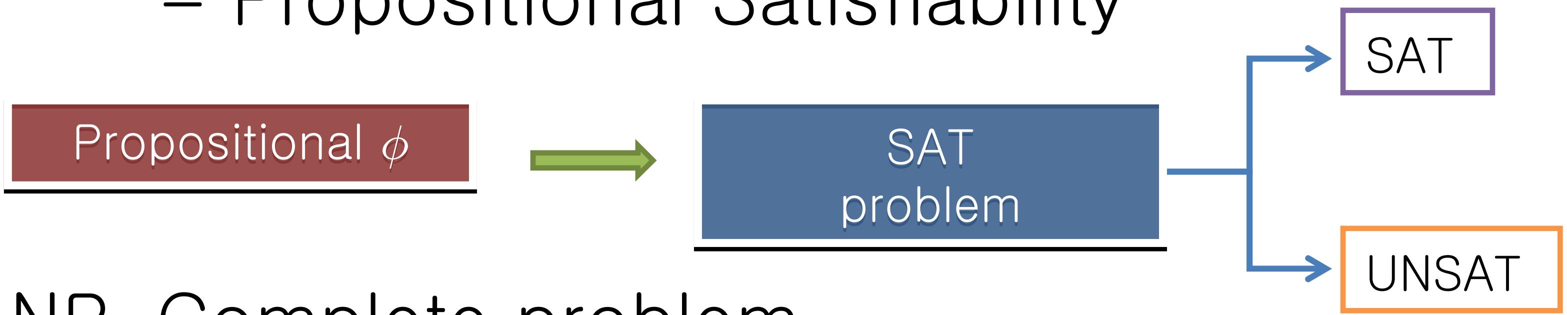
Model Checking as a SAT Problem

- What we are going to do



SAT?

- SAT = Satisfiability
  = Propositional Satisfiability



- NP-Complete problem
  - We can use SAT solver for many NP-complete problems
    - Hamiltonian path
    - 3 coloring problem
    - Traveling sales man's problem
- Recent interest as a verification engine

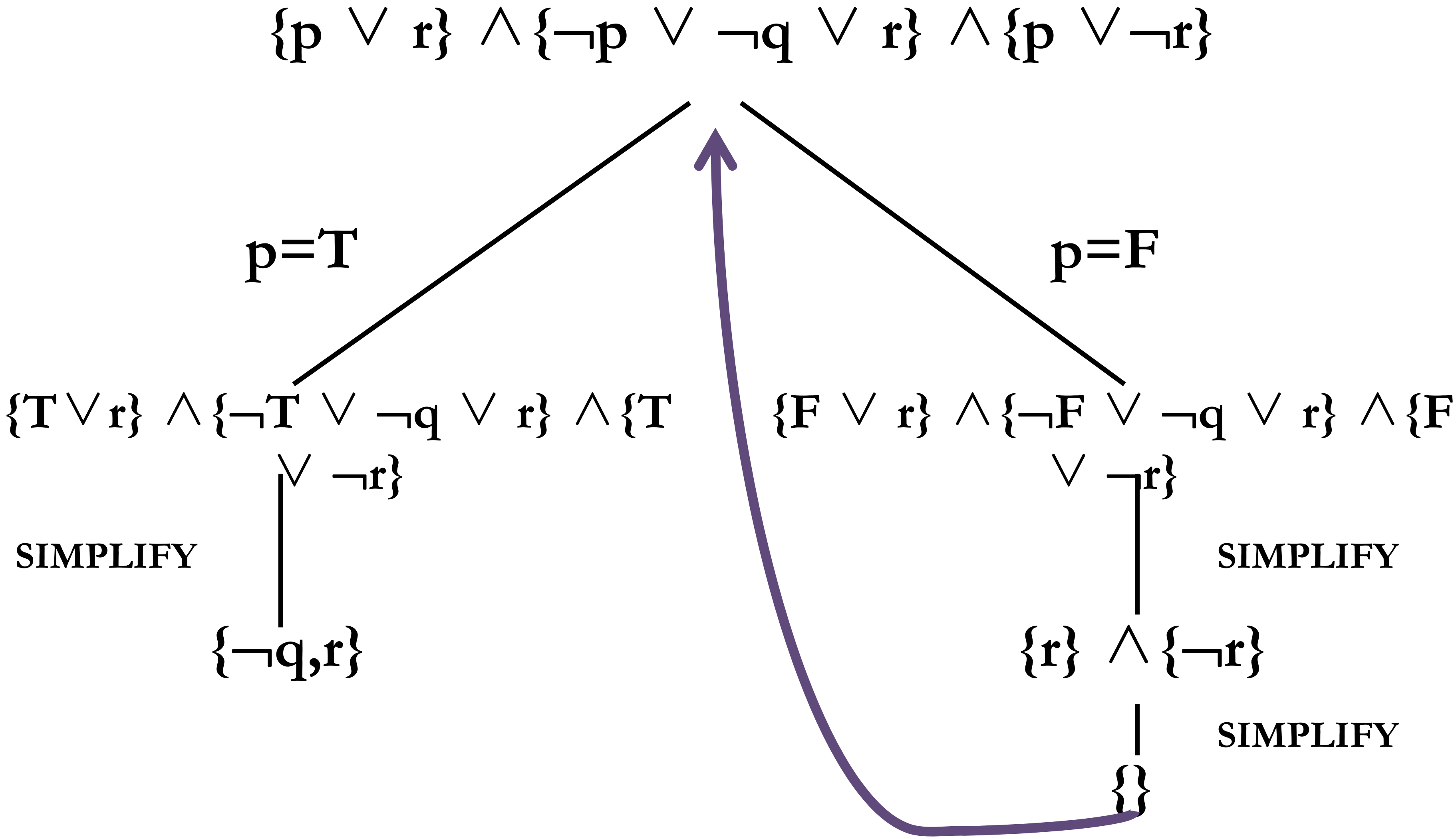
SAT Formula

- A set of propositional variables and clauses involving variables
  - $(x_1 \lor x_2' \lor x_3) \land (x_2 \lor x_1' \lor x_4)$
  - x_1, x_2, x_3 and x_4 are variables (true or false)
- Literals: Variable and its negation
  - x_1 and x_1'
- A clause is satisfied if one of the literals is true
  - x_1=true satisfies clause 1
  - x_1=false satisfies clause 2
- Solution: An assignment v that satisfies all clauses

DPLL(Davis-Putnam-Logemann-Loveland) Framework

```
/* The Quest for Efficient Boolean Satisfiability Solvers
 *  by L.Zhang and S.Malik, Computer Aided Verification 2002 */
DPLL(a formula φ, assignment) {
    necessary = deduction(φ, assignment);
    new_asgnment = union(necessary, assignment);
    if (is_satisfied(φ, new_asgnment))
        return SATISFIABLE;
    else if (is_conflicting(φ, new_asgnmnt))
        return UNSATISFIABLE;
    var = choose_free_variable(φ, new_asgnmnt);
    asgn1 = union(new_asgnmnt, assign(var, 1));
    if (DPLL(φ, asgn1) == SATISFIABLE)
        return SATISFIABLE;
    else {
        asgn2 = union (new_asgnmnt, assign(var,0));
        return DPLL (φ, asgn2);
    }
}
```

DPLL Example



Simple Translation From Code to SAT Formula

- CBMC (C Bounded Model Checker, In CMU)
  - Handles function calls using inlining
  - Unwinds the loops a fixed number of times
  - Allows user input to be modeled using non-determinism
    - So that a program can be checked for a set of inputs rather than a single input
  - Allows specification of assertions which are checked using the bounded model checking

Unwinding Loop

Original code

```
x=0;
while (x < 2) {
  y=y+x;
  x++;
}
```

Unwinding the loop 3 times

```
x=0;
if (x < 2) {
  y=y+x;
  x++;
}
if (x < 2) {
  y=y+x;
  x++;
}
if (x < 2) {
  y=y+x;
  x++;
}
```

Unwinding assertion: → `assert (! (x < 2))`

From C Code to SAT Formula

Original code

```
x=x+y;
if (x!=1)
  x=2;
else
  x++;
assert(x<=3);
```

Convert to static single v
(static single assignment (SSA))

x_1=x_0+y_0;
if (x_1!=1)
  x_2=2;
else
  x_3=x_1+1;
x_4=(x_1!=1)?x_2:x_3;
assert(x_4<=3);

Generate constraints

$$C \equiv x_1=x_0+y_0 \land x_2=2 \land x_3=x_1+1 \land (x_1!=1 \land x_4=x_2 \lor x_1=1 \land x_4=x_3)$$
$$P \equiv x_4 <= 3$$

Check if $C \land \neg P$ is satisfiable, if it is then the assertion is violated

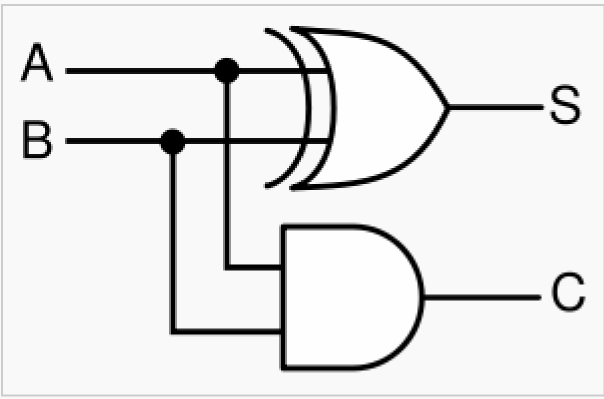
$C \land \neg P$ is converted to Boolean logic using a bit vector representation
for the integer variables $y_0, x_0, x_1, x_2, x_3, x_4$ and their arithmetic operations

From C Code to SAT Formula

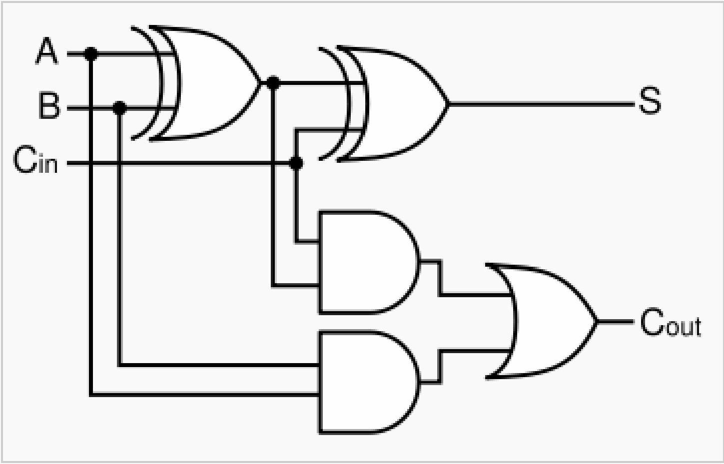
- Example of arithmetic encoding into pure propositional formula

Assume that x,y,z are three bits positive integers represented by propositions $x_0x_1x_2$, $y_0y_1y_2$, $z_0z_1z_2$

$$C \equiv z=x+y \equiv (z_0 \leftrightarrow (x_0 \oplus y_0) \oplus ((x_1 \wedge y_1) \vee (((x_1 \oplus y_1) \wedge (x_2 \wedge y_2)))$$
$$\wedge (z_1 \leftrightarrow (x_1 \oplus y_1) \oplus (x_2 \wedge y_2))$$
$$\wedge (z_2 \leftrightarrow (x_2 \oplus y_2))$$



Half adder circuit diagram



Full adder circuit diagram
Inputs: {A, B, CarryIn} → Outputs: {Sum, CarryOut}

SAT-Solvers?

- Started with DPLL (1962)
  - Able to solve 10-15 variable problems
- Satz (Chu Min Li, 1995)
  - Able to solve some 1000 variable problems
- Chaff (Malik et al., 2001)
  - Intelligently hacked DPLL , Won the 2004 competition
  - Able to solve some 10000 variable problems
- Current state-of-the-art
  - Minisat and SATELITEGTI (Chalmer's university, 2004-2006)
  - Jerusat and Haifasat (Intel Haifa, 2002)
  - Ace (UCLA, 2004-2006)

Countermeasure of State Explosion

1981	Clarke / Emerson: CTL Model Checking Sifakis / Quielle	10^5
1982	EMC: Explicit Model Checker Clarke, Emerson, Sistla	
1990	Symbolic Model Checking Burch, Clarke, Dill, McMillan	10^{100}
1992	SMV: Symbolic Model Verifier McMillan	
1998	Bounded Model Checking using SAT Biere, Clarke, Zhu	10^{1000}
2000	Counterexample-guided Abstraction Refinement Clarke, Grumberg, Jha, Lu, Veith	