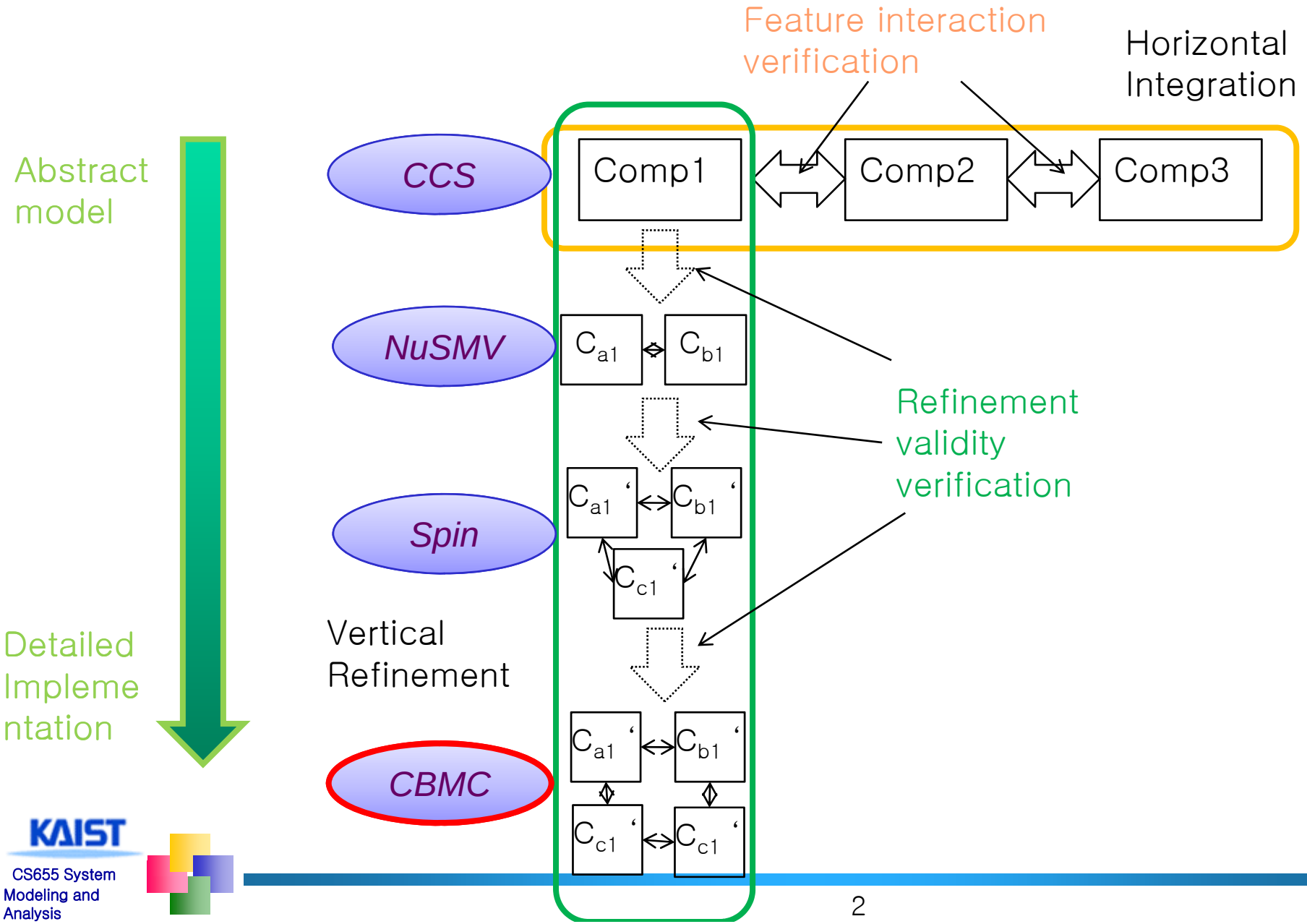
SAT-based Analysis for C Programs

Moonzoo Kim

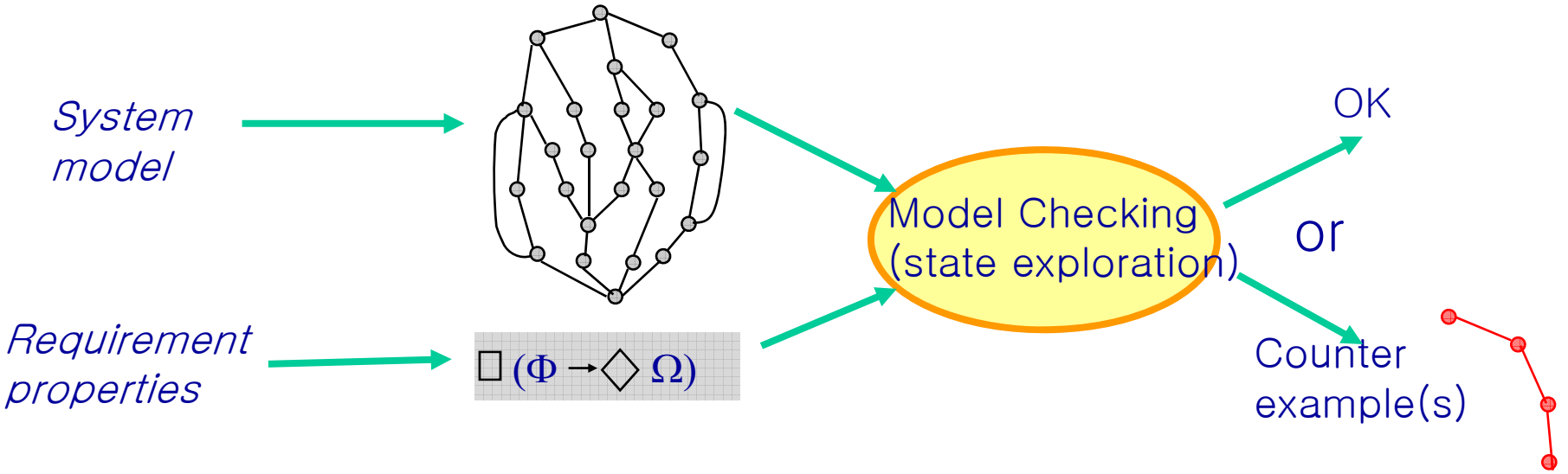
KAIST Korea Advanced Institute of Science and Technology

Verification Frameworks for Various Abstraction Level



Model Checking Basics

- Specify requirement properties and build a system model
- Generate possible states from the model and then check whether given requirement properties are satisfied within the state space

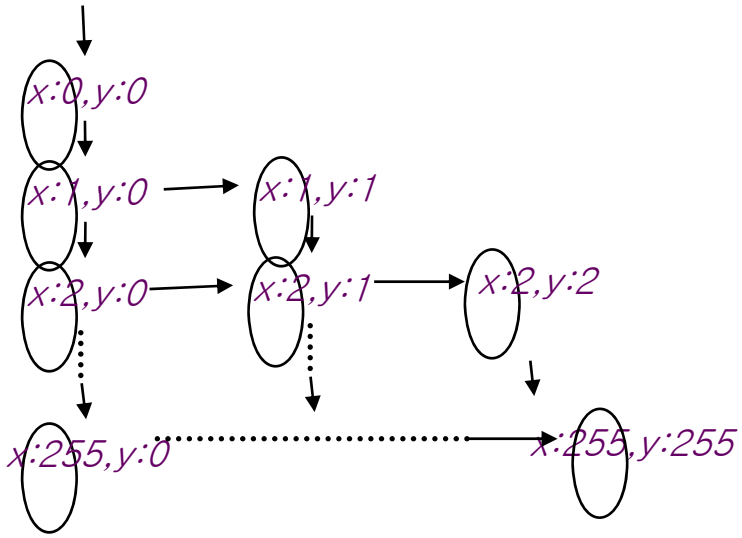


An Example of Model Checking ½

(checking every possible execution path)

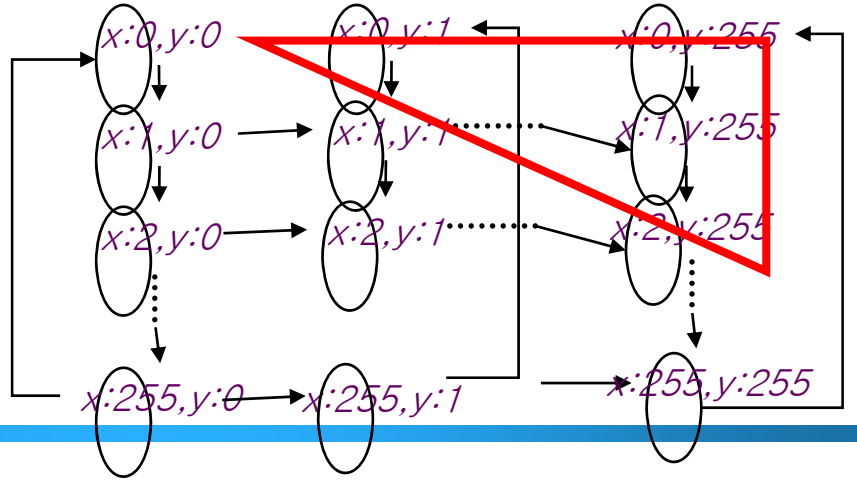
System Spec.

```
unsigned char x=0;
unsigned char y=0;

void proc_A()
  while(1)
    x++;
}
⋮
void proc_B(){
  while(1)
    if (x>y)
      y++;
}
```



Req. Spec

□ (x >= y)



An Example of Model Checking 2/2

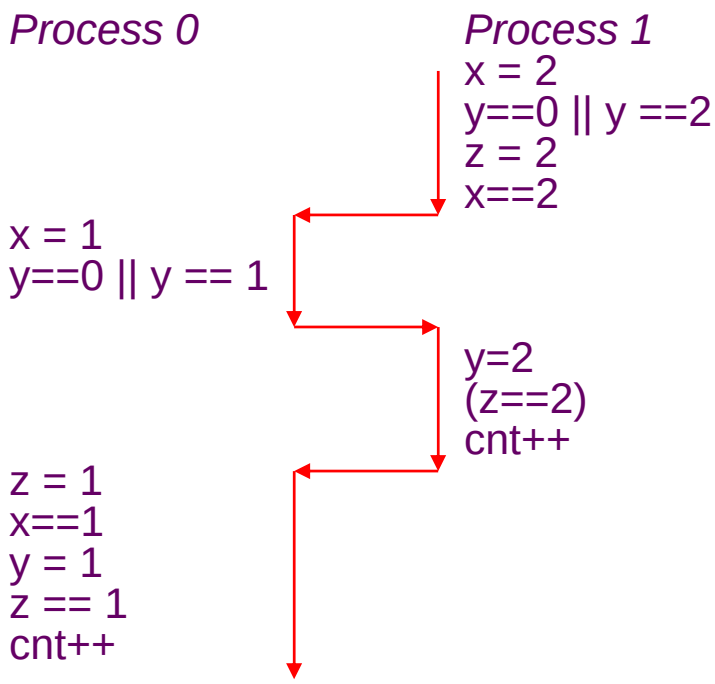
(*checking every possible thread scheduling*)

```
char cnt=0,x=0,y=0,z=0;

void process() {
        char me = _pid +1; /* me is 1 or 2*/
again:
        x = me;
        If (y ==0 || y== me) ;
        else goto again;

        z =me;
        If (x == me) ;
        else goto again;

        y=me;
        If(z==me);
        else goto again;

        /* enter critical section */
        cnt++;
        assert( cnt ==1);
        cnt --;
        goto again;
}
```

Software locks

Critical section

Mutual Exclusion Algorithm



Counter Example

Model Checking v.s. Static Analysis

- Data flow analysis: fastest & least precision
  - “May” analysis,
- Abstract interpretation: fast & medium precision
  - Over-approximation & under-approximation
- Model checking: slow & complete
  - Complet value analysis
  - No approximation

- Static analyzer & MC as a C debugger
  - Handling complex C structures such as pointer and array
    - DFA: might-be
    - AI: may-be
    - MC: can-be
    - SAT-based MC: (almost)complete

Example. Sort (1/2)

- Suppose that we have an array of 4 elements each of which is 1 byte long

  - unsigned char a[4];
- We wants to verify sort.c works correctly
  - main() { sort(); assert(a[0]<= a[1]<= a[2]<=a[3]);}
- Explicit model checker (ex. Spin) requires at least 2^{32} bytes of memory
    - 4 bytes = 32 bits, No way…
- Symbolic model checker (ex. NuSMV) takes 200 megabytes in 400 sec

Example. Sort (2/2)

```
1. #include <stdio.h>
2. #define N 5
3. int main(){
4.     int data[N], i, j, tmp;
5.     /* Assign random values to the array*/
6.     for (i=0; i<N; i++){
7.         data[i] = nondet_int();
8.     }
9.     /* It misses the last element, i.e., data[N-1]*/
10.    for (i=0; i<N-1; i++)
11.        for (j=i+1; j<N-1; j++)
12.            if (data[i] > data[j]){
13.                tmp = data[i];
14.                data[i] = data[j];
15.                data[j] = tmp;
16.            }
17.  /* Check the array is sorted */
18.    for (i=0; i<N-1; i++){
19.        assert(data[i] <= data[i+1]);
20.    }
21. }
```

- Total 6224 CNF clause with 19099 boolean propositional variables

- Theoretically, 2^{19099} (2.35×10^{5749}) choices should be evaluated!!!

SAT	VSIDS	Modified
Conflicts	73	72
Decisions	2435	2367
Time(sec)	0.015	0.013

UNSAT	VSIDS	Modified
Conflicts	35067	30910
Decisions	161406	159978
Time(sec)	1.89	1.60

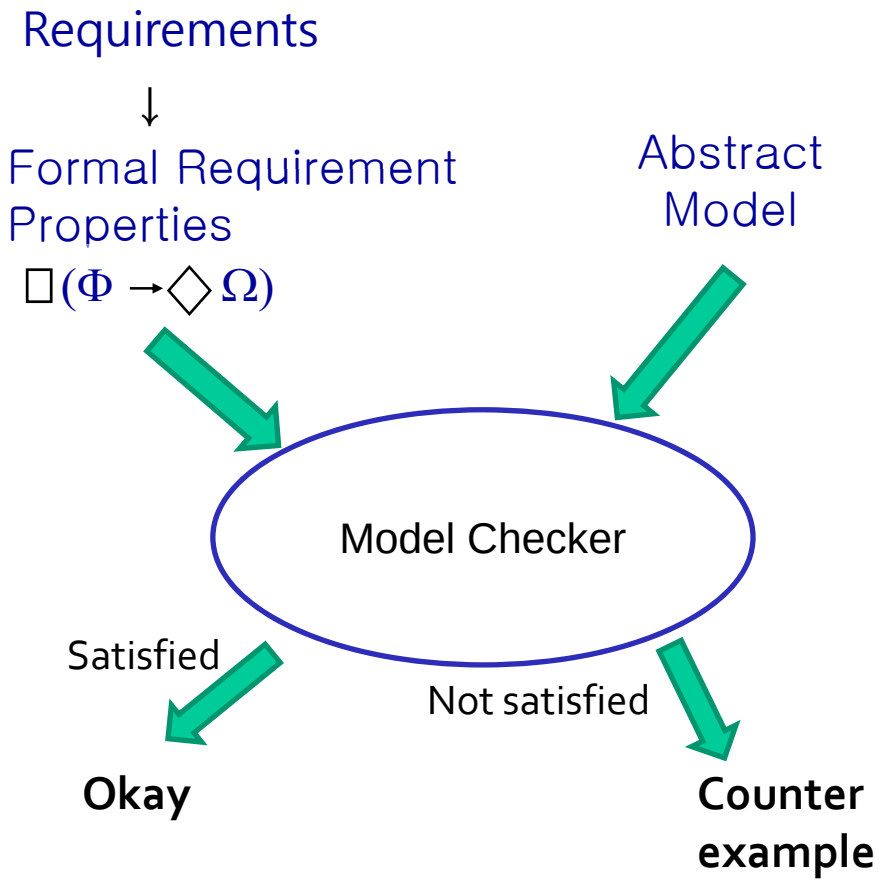
SAT-based Bounded Model Checking

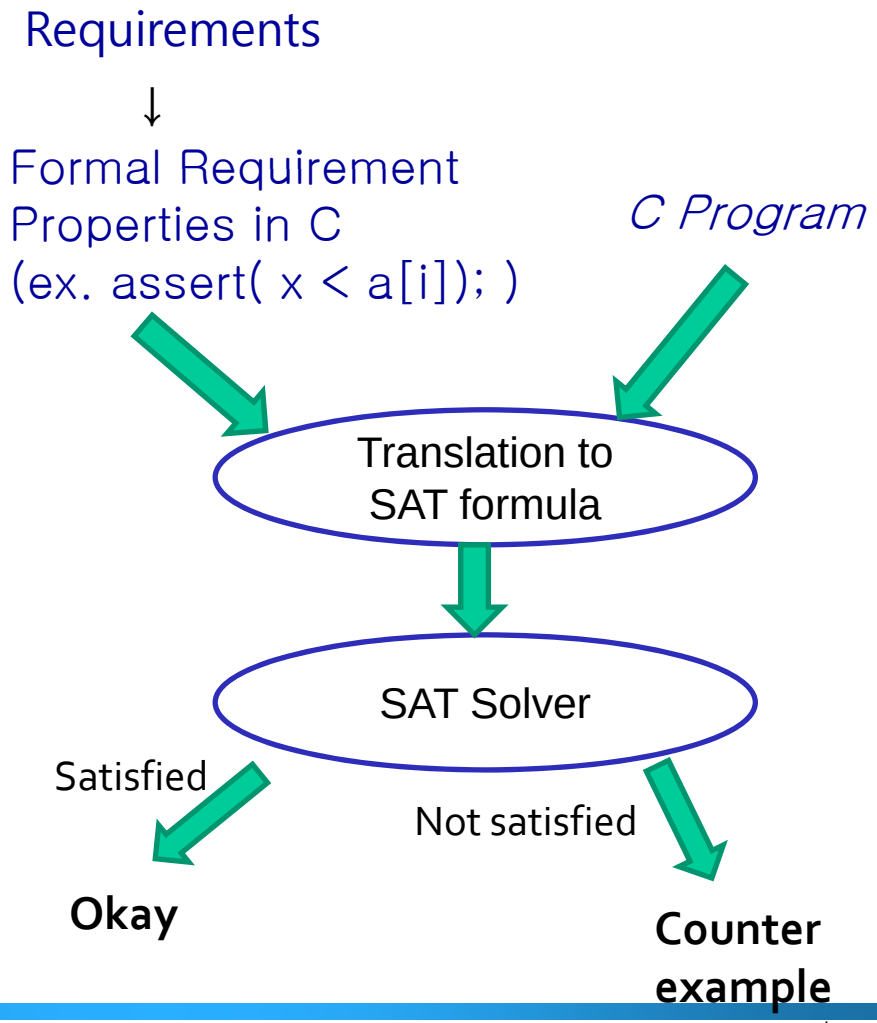
- Model Checking History
- SAT Basics
- Model Checking as a SAT problem

Model Checking History

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	

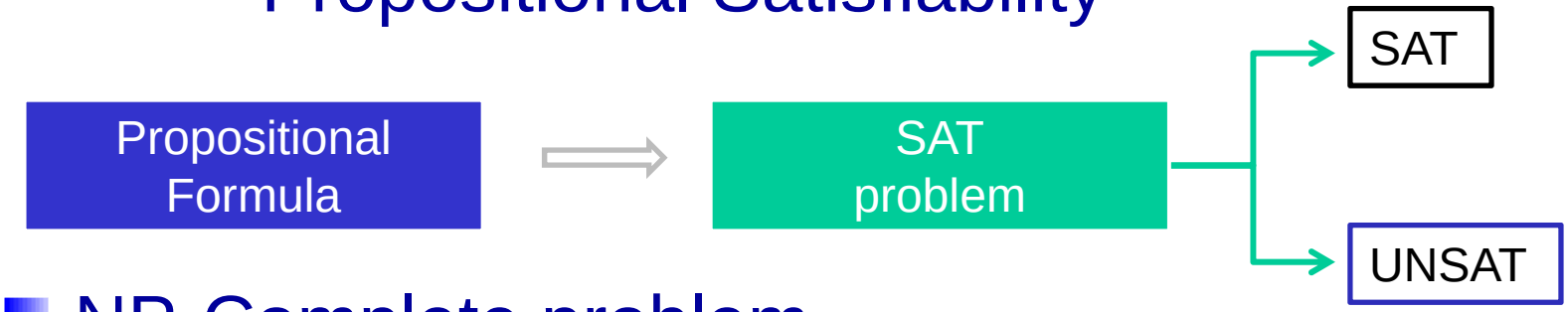
Overview of SAT-based Bounded Model Checking





SAT Basics (1/2)

- 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

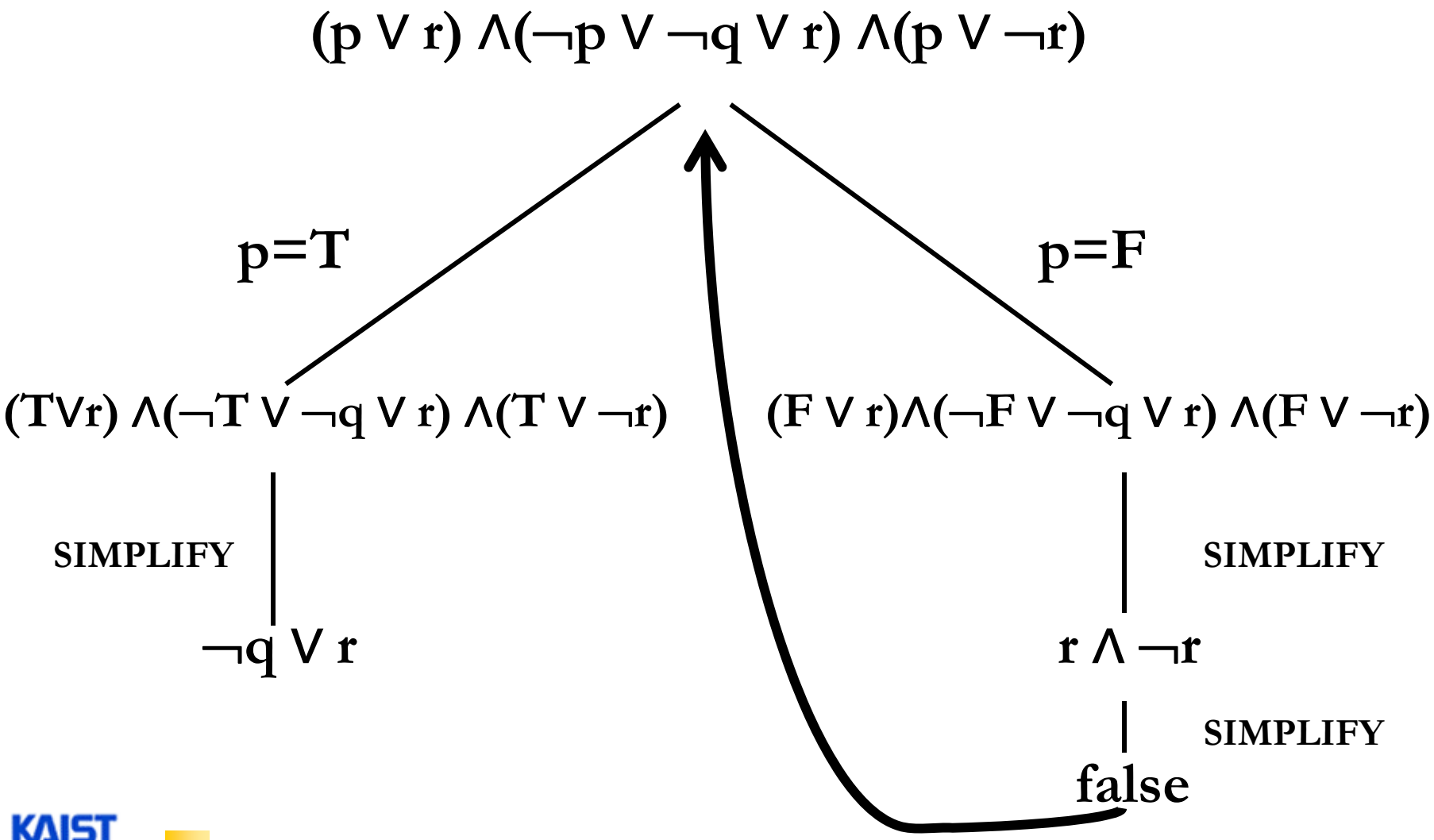
SAT Basics (2/2)

- A set of propositional variables and clauses involving variables
  - $(x_1 \vee x_2' \vee x_3) \wedge (x_2 \vee x_1' \vee x_4)$
  - 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 that satisfies all clauses

Basic SAT Solving Mechanism (1/2)

```
/* 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);
```

Basic SAT Solving Mechanism (2/2)



Model Checking as a SAT problem (1/4)

- 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

Model Checking as a SAT problem (2/4)

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))
```

Model Checking as a SAT problem (3/4)

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 assignment

```
x1=x0+y0;
if (x1!=1)
  x2=2;
else
  x3=x1+1;
x4=(x1!=1)?x2:x3;
assert(x4<=3);
```

Generate constraints

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

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

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

Model Checking as a SAT problem (4/4)

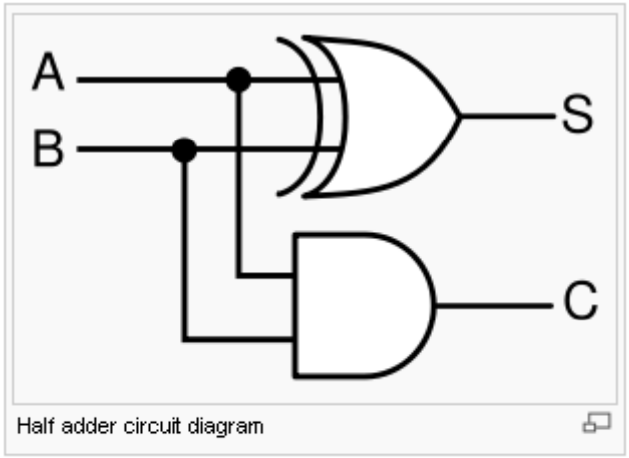
- Example of arithmetic encoding into pure propositional formula

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

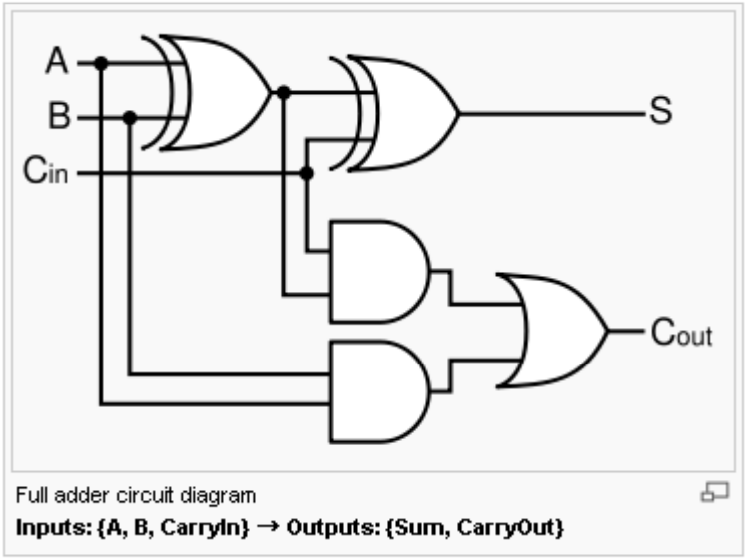
  $C \equiv z=x+y \equiv \quad (z_0 \leftrightarrow (x_0 \oplus y_0) \oplus ((x_1 \wedge y_1) \vee (((x_1 \oplus y_1) \wedge (x_2 \wedge y_2)))$

  $\wedge \ (z_1 \leftrightarrow (x_1 \oplus y_1) \oplus (x_2 \wedge y_2))$

  $\wedge \ (z_2 \leftrightarrow (x_2 \oplus y_2))$



Half adder circuit diagram



Full adder circuit diagram

Inputs: {A, B, CarryIn} → Outputs: {Sum, CarryOut}