CSE4051: Program Verification

Applications of SAT

2025 Fall

Woosuk Lee

SAT solvers can be used in various applications

- Hardware and software verification
- Automated testing of circuits
- Package management
- Artificial intelligence (e.g., planning, scheduling)
- Cryptography
- Computational Biology
- …

Exercises

- In this lecture, we will try to solve various satisfiability problems using a SAT solver called Z3.
- Z3 is a high-performance theorem prover developed by Microsoft Research.
- We will use Z3Py, the Python interface for Z3, to write and solve logical formulas.

Using Z3Py

- Install Z3Py using pip: `pip install z3-solver`

- Import Z3Py in your Python script: `from z3 import *`

- Define Boolean variables:

```
a = Bool("a")
b = Bool("b")
```

- Create logical formulas using Z3Py:

```
f1 = And(Not(a), Not(b))
f2 = Or(a, b)
```

- Solve the satisfiability problem:

```
solve(Not(f1 == f2))
```

- The `solve` function will return whether the formula is satisfiable or not, and if it is, it will provide an interpretation that satisfies the formula.

Verifying Correctness of Optimizations

Optimization of if-then-else chains

original C code

```
if(!a && !b) h();
else if(!a) g();
else f();
```

⇓

```
if(!a) {
  if(!b) h();
  else g();
} else f();
```

⇒

```
if(a) f();
else {
  if(!b) h();
  else g(); }
```

⇑

optimized C code

```
if(a) f();
else if(b) g();
else h();
```

Verifying Correctness of Optimizations

- Represent procedures as Boolean variables

$$
\begin{array}{ll}
\textit{original} := & \textit{optimized} := \\
\quad \textbf{if } \neg a \wedge \neg b \textbf{ then } h & \quad \textbf{if } a \textbf{ then } f \\
\quad \textbf{else if } \neg a \textbf{ then } g & \quad \textbf{else if } b \textbf{ then } g \\
\quad \textbf{else } f & \quad \textbf{else } h
\end{array}
$$

- Compile if-then-else chains into Boolean formula

$$\text{compile}(\textbf{if } x \textbf{ then } y \textbf{ else } z) \quad \equiv \quad (x \wedge y) \ \vee \ (\neg x \wedge z)$$

- Check equivalence of Boolean formula

$$\boxed{\text{compile}(\textit{original}) \quad \Leftrightarrow \quad \text{compile}(\textit{optimized})}$$

Verifying Correctness of Optimizations

$$
\begin{aligned}
\textit{original} &\equiv \textbf{if } \neg a \wedge \neg b \textbf{ then } h \textbf{ else } \textbf{if } \neg a \textbf{ then } g \textbf{ else } h \\
&\equiv (\neg a \wedge \neg b) \wedge h \;\vee\; \neg(\neg a \wedge \neg b) \wedge \textbf{ if } \neg a \textbf{ then } g \textbf{ else } f \\
&\equiv (\neg a \wedge \neg b) \wedge h \;\vee\; \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \;\vee\; a \wedge f)
\end{aligned}
$$

$$
\begin{aligned}
\textit{optimized} &\equiv \textbf{if } a \textbf{ then } f \textbf{ else } \textbf{if } b \textbf{ then } g \textbf{ else } h \\
&\equiv a \wedge f \;\vee\; \neg a \wedge \textbf{ if } b \textbf{ then } g \textbf{ else } h \\
&\equiv a \wedge f \;\vee\; \neg a \wedge (b \wedge g \;\vee\; \neg b \wedge h)
\end{aligned}
$$

$$
(\neg a \wedge \neg b) \wedge h \vee \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \vee a \wedge f) \quad \Leftrightarrow \quad a \wedge f \vee \neg a \wedge (b \wedge g \vee \neg b \wedge h)
$$

Exercise

- Suppose now the optimized version is

```
if !a then h else if b then g else f
```

- Is it still equivalent to the original one?

Or (And(Not(a), h), And(a, Or(And(b, g), And(Not(b), f))))

Seat Assignment

- Consider three persons 1, 2, and 3 who need to be seated in a row. There are three constraints:
  - 1 does not want to sit next to 3
  - 1 does not want to sit in the leftmost chair
  - 2 does not want to sit to the right of 3
- We would like to check if there is a seat assignment for the three persons that satisfies the above constraints.

From https://prl.korea.ac.kr/courses/aaa528/2025/slides/lec3.pdf

Encoding of Seat Assignment

- Let X_{ij} be boolean variables such that

  $$X_{ij} \iff \text{person } i \text{ seats in chair } j$$

- Constraints
  - Every person is seated: $\bigwedge_{i=1}^{3} \bigvee_{j=1}^{3} X_{ij}$
  - Every seat is occupied: $\bigwedge_{j=1}^{3} \bigvee_{i=1}^{3} X_{ij}$
  - One person per seat: $\bigwedge_{i,j \in \{1,2,3\}} (X_{i,j} \implies \bigwedge_{k,j \in \{1,2,3\}, k \neq j} \neg X_{i,k})$

Encoding of Seat Assignment

- Person 1 does not want to sit next to person 3:

$$(X_{00} \implies \neg X_{21}) \wedge (X_{01} \implies (\neg X_{20} \wedge \neg X_{22})) \wedge (X_{02} \implies \neg X_{21})$$

- Person 1 does not want to sit in the leftmost chair: $\neg X_{00}$

- Person 2 does not want to sit to the right of person 3:

$$(X_{20} \implies \neg X_{11}) \wedge (X_{21} \implies \neg X_{12})$$

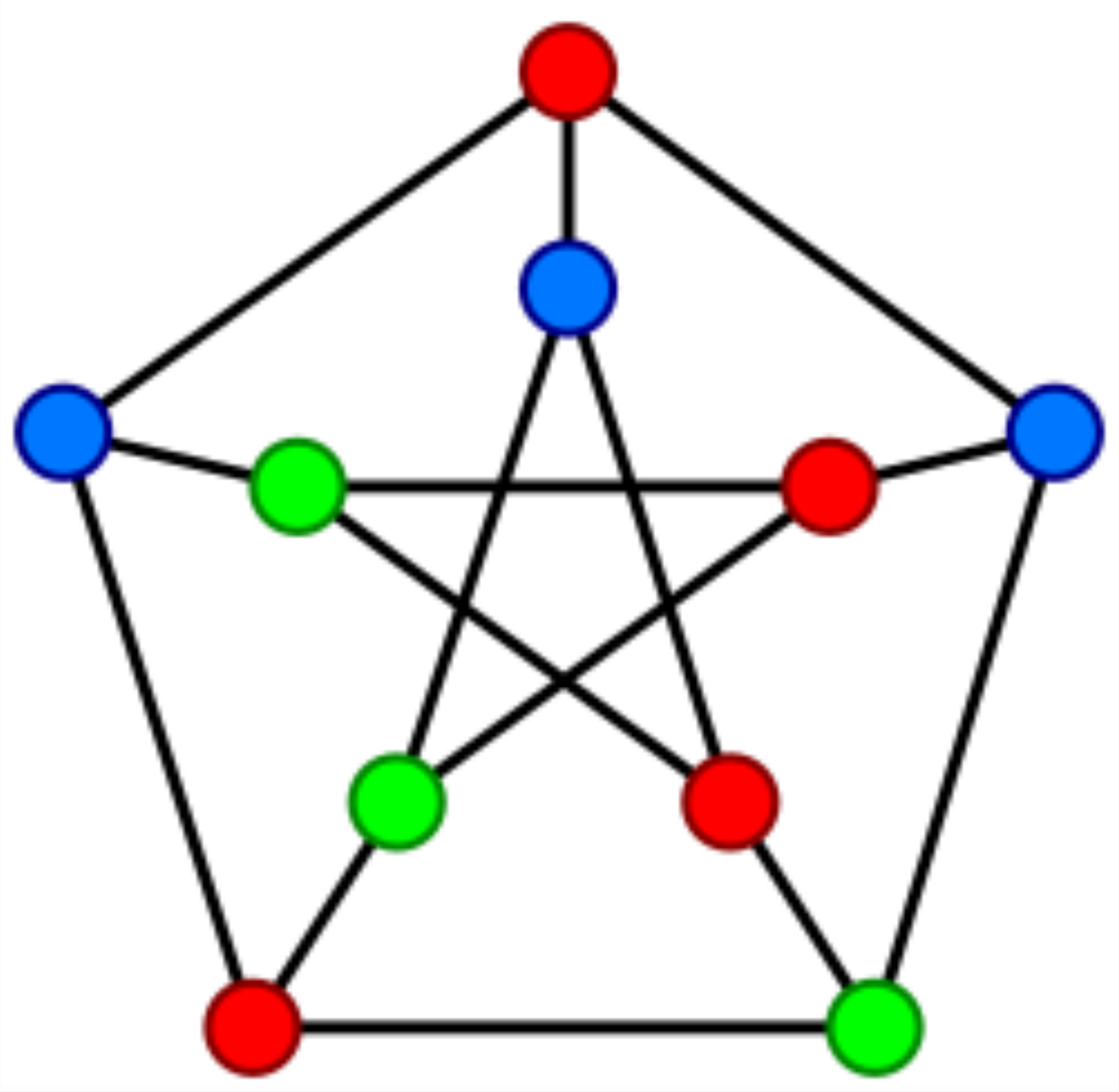
Exercise

- Remove the constraint "Person 1 does not want to sit in the leftmost chair" and get a seat assignment.

Or (And(Not(a), h), And(a, Or(And(b, g), And(Not(b), f))))

Graph Coloring

- A graph is k-colorable if there is an assignment of k colors to its vertices such that no two adjacent vertices have the same color.
- Deciding if such a coloring exists is a classic NP-complete problem with many practical applications, such as register allocation in compilers.
- For example, a coloring with 3 colors of a graph:

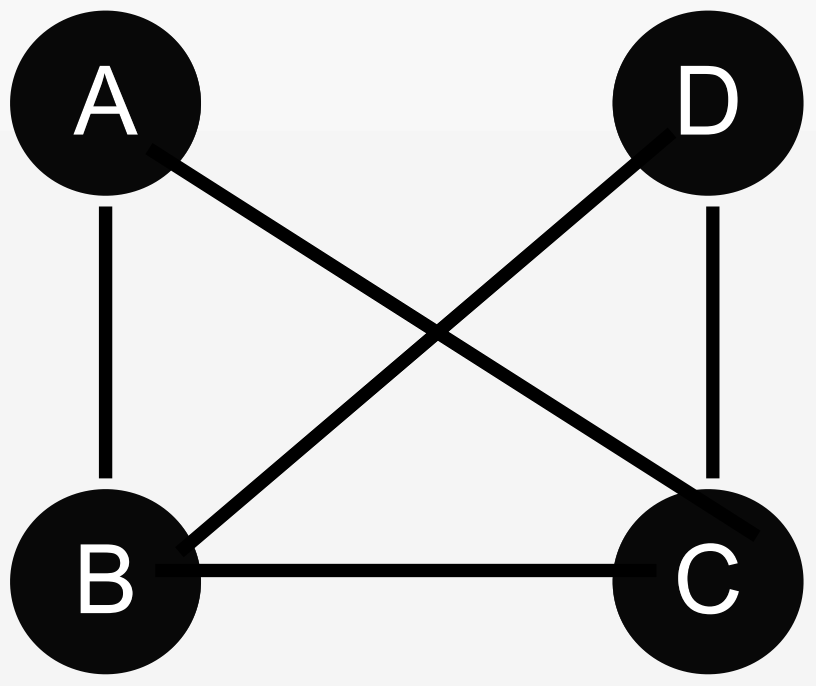


Graph Coloring

- A finite graph $G = < V, E >$ where $V = \{v_1, \ldots, v_n\}$ is a set of vertices and $E = \{(v_{i1}, w_{i1}), \ldots, (v_{im}, w_{im})\}$ is a set of edges. Given a set of k colors in $C = \{c_1, \ldots, c_k\}$, the k-coloring problem for G is to assign a color $c \in C$ to each vertex $v \in V$ s.t. for every edge $< v, w > \in E$, color$(v) \neq$ color(w).
- Introduce Boolean variables x_{ij} such that $x_{ij} \iff v_i$ is assigned color c_j
- Conditions
  - Every vertex is assigned at least one color.
  - Every vertex is assigned not more than one color.
  - Neighbors are not assigned the same color

Graph Coloring

```python
from z3 import *

# Define the graph
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'C', 'D'],
    'C': ['A', 'B', 'D'],
    'D': ['B', 'C']
}

nodes = list(graph.keys())
k = 3  # number of colors

# Step 1: Create Boolean variables: color_vars[node][color]
color_vars = {
    node: [Bool(f"{node}_{c}") for c in range(k)]
    for node in nodes
}

solver = Solver()
```

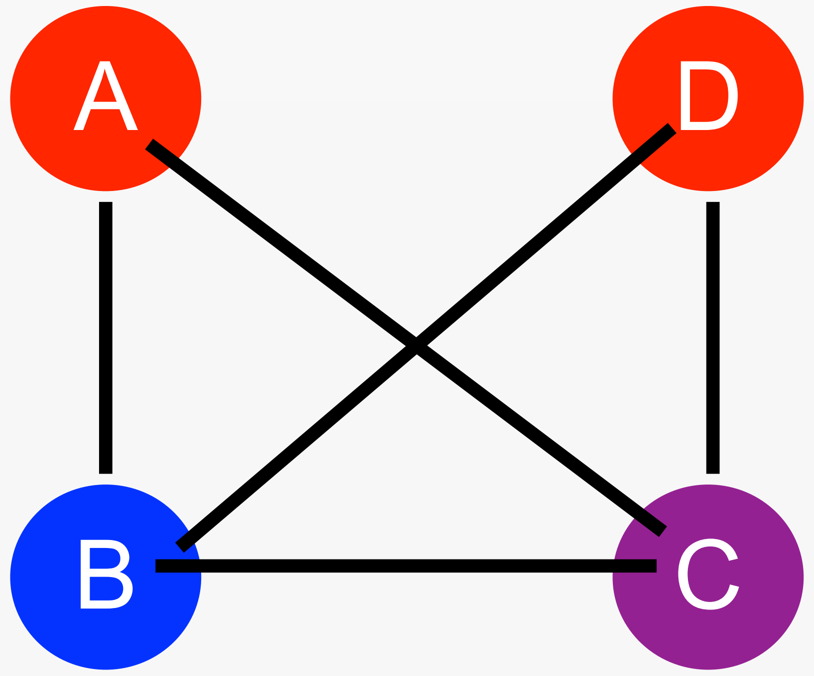


Graph Coloring

```python

# Step 2: Each node must have exactly one color
for node in nodes:
    # At least one color
    solver.add(Or(color_vars[node]))

    # At most one color
    for c1 in range(k):
        for c2 in range(c1 + 1, k):
            solver.add(Not(And(color_vars[node][c1], color_vars[node][c2])))

# Step 3: Adjacent nodes must not share the same color
for node in graph:
    for neighbor in graph[node]:
        if node < neighbor:  # avoid duplicate constraints
            for c in range(k):
                solver.add(Or(Not(color_vars[node][c]), Not(color_vars[neighbor][c])))

```

Graph Coloring

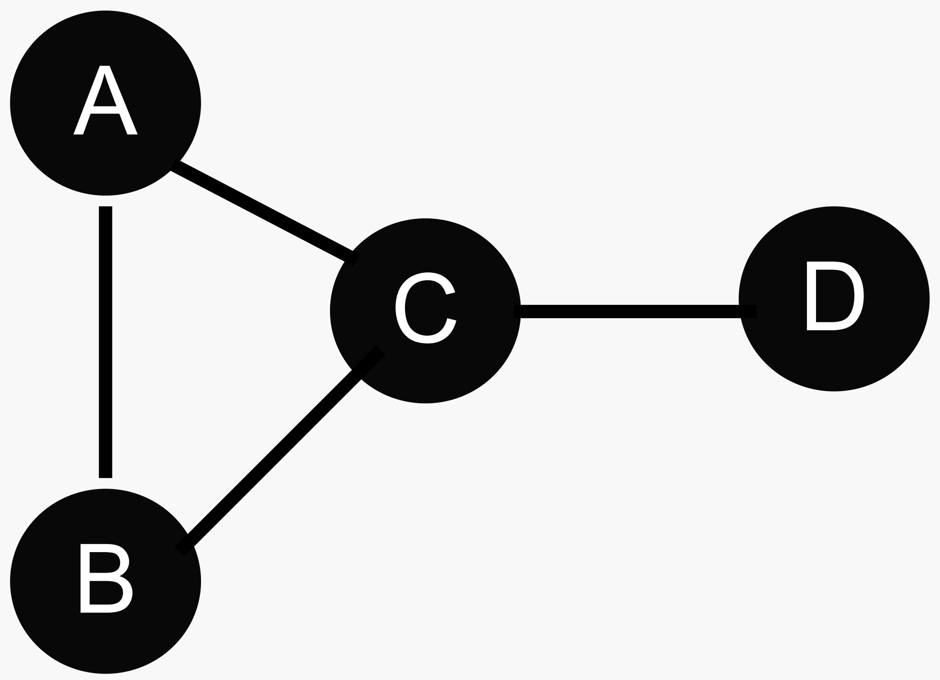
```python
# Step 4: Solve and display
if solver.check() == sat:
    model = solver.model()
    print("Coloring found:")
    for node in nodes:
        for c in range(k):
            if model.evaluate(color_vars[node][c]):
                print(f"  {node}: Color {c}")
else:
    print("No valid coloring found.")
```




```
Coloring found:
  A: Color 1
  B: Color 2
  C: Color 0
  D: Color 1
```

Exercise

- Consider the graph on the right.



Get all possible 3-colorings of the graph.

```
for node in nodes:
    for c in range(k):
        lit = color_vars[node][c]
        if m.evaluate(lit):
            block.append(Not(lit))
        else:
            block.append(lit)
```

Package Management

- Install problem: determining whether a new set of packages can be installed in a system
- Many packages depend on other packages to provide some functionality.
- Each distribution contains a meta-data file containing the name, version, etc.
- More importantly, it contains **depends** and **conflicts** clauses that stipulate which other packages should be on the system.

Package Management

$x_a \iff$ package a is installed

```
Package: apache
Architecture: i386
Version: 1.3.34-2
Provides: httpd-cgi, httpd
Depends: libc6(>=2.3.5-1),
 libdb4.3(>=4.3.28-1),
 debconf(>=0.5) | debconf-2.0,
 apache-common(>=1.3.34-2),
 perl(>=5.8.4-2)
Conflicts: apache-modules,
 jserv(<=1.1-3)
 libapache-mod-perl
Description: HTTP server.
```

Figure 1: Metadata for `apache`

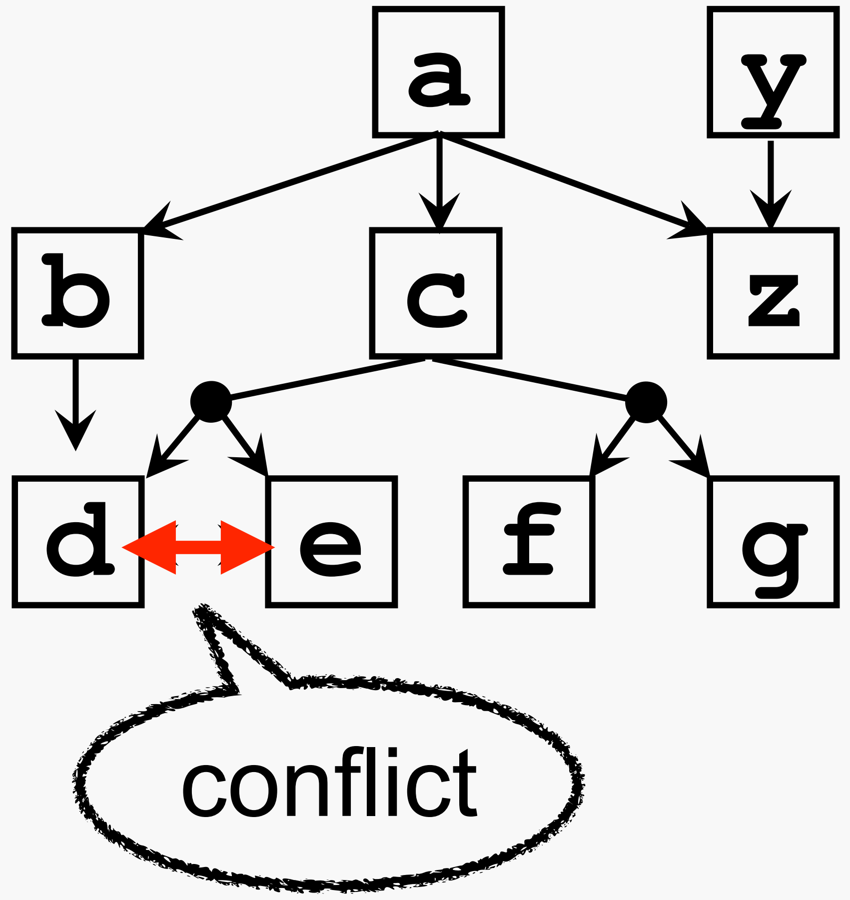


Figure 2: Distribution Graph

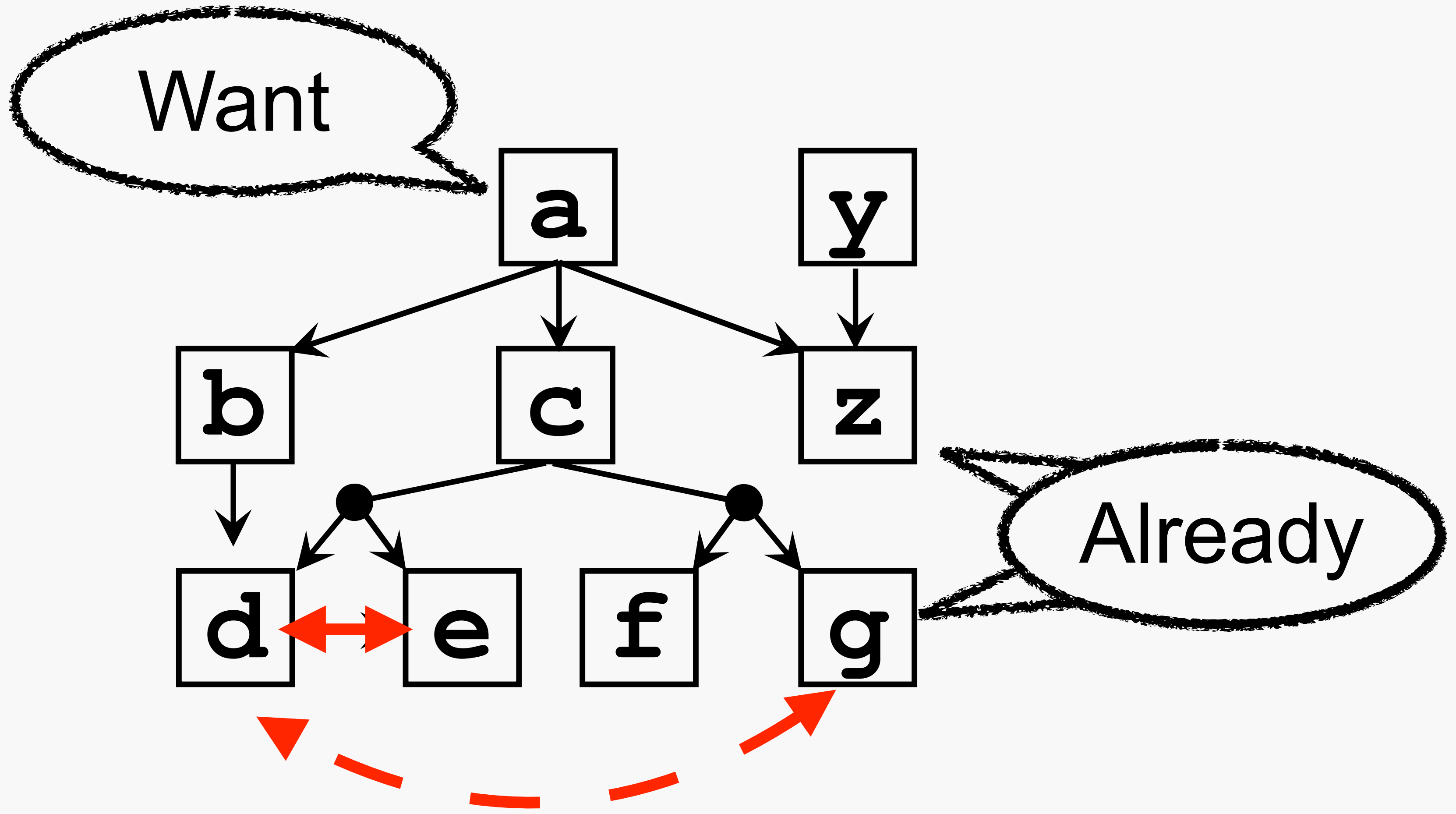
Distribution Rules	Constraints
`Package: a`	
`Depends: b,`	$(\neg x_a \lor x_b)$
`c,`	$(\neg x_a \lor x_c)$
`z`	$(\neg x_a \lor x_z)$
`Package: b`	
`Depends: d`	$(\neg x_b \lor x_d)$
`Package: c`	
`Depends: d | e,`	$(\neg x_c \lor x_d \lor x_e)$
`f | g`	$(\neg x_c \lor x_f \lor x_g)$
`Package: d`	
`Conflicts: e`	$(\neg x_d \lor \neg x_e)$

Figure 3: Fragment of Distribution Metadata and Corresponding Constraints

The formula will be the constraints in Figure 3 along with packages to be installed and already installed

Package Management

- Installation in the presence of conflicts: to install “a” while minimizing the number of removed components, what can we do?



MaxSAT

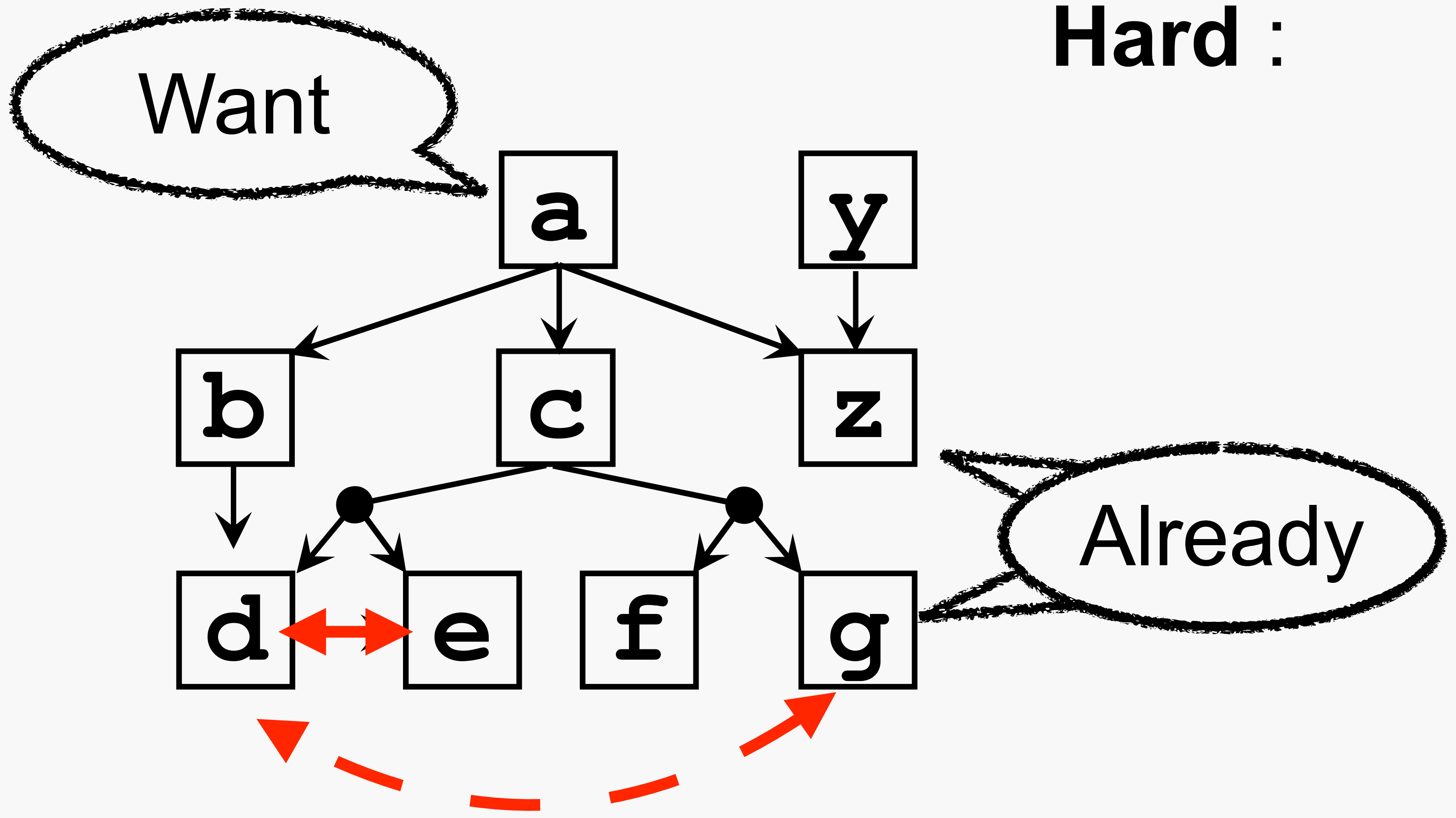
- Given a formula F in CNF, find assignment maximizing the number of satisfied clauses of F
  - If F is satisfiable, the solution is simply the number of clauses in F
  - If F is unsatisfiable, we want to find a maximum subset of F's clauses whose conjunction is satisfiable
  - For $(a \vee b) \wedge \neg a \wedge \neg b$, a solution is $\{a \mapsto \bot, b \mapsto \bot\}$

Partial MaxSAT

- The goal is the same as MaxSAT except that we have
  - Hard constraints: clauses that must be satisfied
  - Soft constraints: clauses that do not have to be satisfied but we want to satisfy as many as possible
- Goal: Given a formula in CNF marked as hard or soft, find an assignment that satisfies all hard constraints and maximizes the number of satisfied soft constraints

Exercise

- Installation in the presence of conflicts: to install "a" while minimizing the number of removed components, what can we do?
  - => we can encode the problem as a partial MaxSAT problem and solve it.

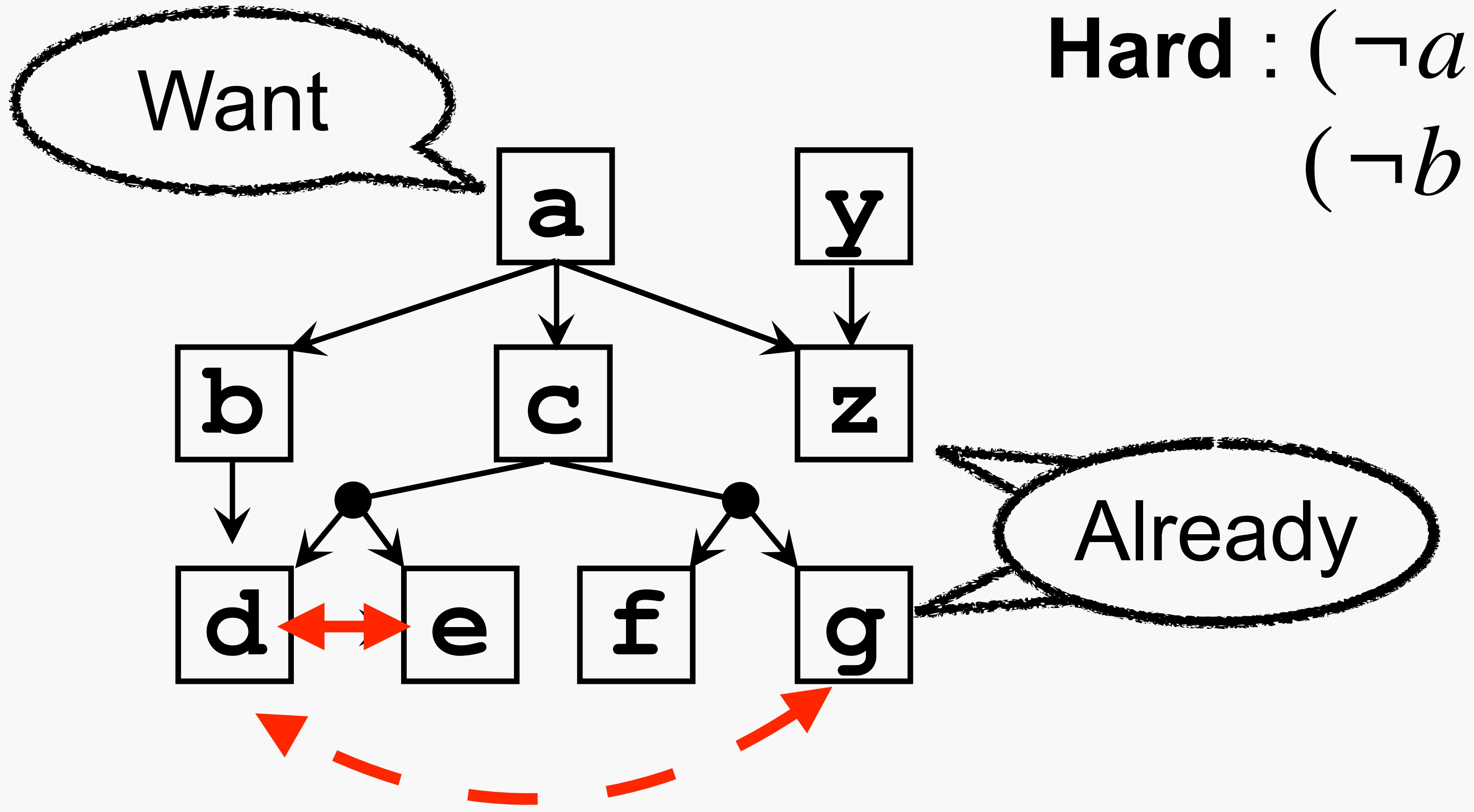


Hard :

Soft:

Exercise

- Installation in the presence of conflicts: to install "a" while minimizing the number of removed components, what can we do?
    - => we can encode the problem as a partial MaxSAT problem and solve it.



Hard : $(\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg a \vee z) \wedge (\neg y \vee z) \wedge$
$(\neg b \vee d) \wedge (\neg c \vee d \vee e) \wedge (\neg c \vee f \vee g) \wedge (\neg d \vee \neg e) \wedge a$

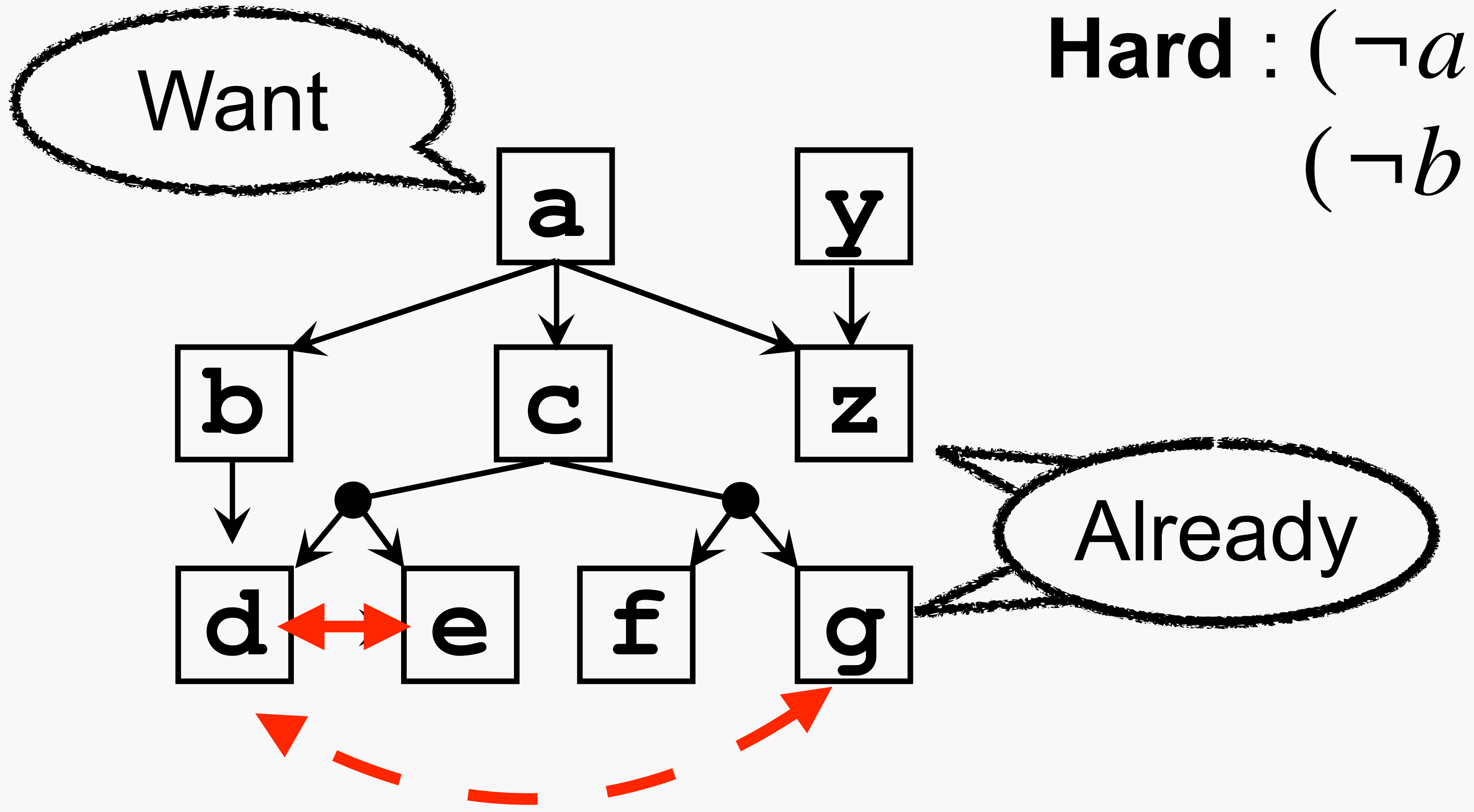
Soft: $z \wedge g$

Partial Weighted MaxSAT

- Soft clauses have weights indicating their importance.
- Goal: Find assignment maximizing the sum of weights of satisfied soft clauses
- Partial MaxSAT is an instance of partial weighted MaxSAT where all clauses have equal weight.

Exercise

- To install "a" minimizing the total size of removed components, assuming z and g are 5MB and 2MB each



Hard : $(\neg a \lor b) \land (\neg a \lor c) \land (\neg a \lor z) \land (\neg y \lor z) \land$
$(\neg b \lor d) \land (\neg c \lor d \lor e) \land (\neg c \lor f \lor g) \land (\neg d \lor \neg e) \land a$

Soft: $z : 5 \land g : 2$