COMP 410: SAT and Semantic Tableau

Kyle Dewey

SAT Background

SAT

- Short for the Boolean satisfiability problem
- Given a Boolean formula with variables, is there an assignment of true/false to the variables which makes the formula true?

SAT

- Short for the Boolean satisfiability problem
- Given a Boolean formula with variables, is there an assignment of true/false to the variables which makes the formula true?

$$(x \lor \neg y) \land (\neg x \lor z)$$

SAT

- Short for the Boolean satisfiability problem
- Given a Boolean formula with variables, is there an assignment of true/false to the variables which makes the formula true?

$(x \lor \neg y) \land (\neg x \lor z)$

Yes: x is true, z is true

SAT

- Short for the Boolean satisfiability problem
- Given a Boolean formula with variables, is there an assignment of true/false to the variables which makes the formula true?

$(x \lor \neg y) \land (\neg x \lor z)$

Yes: x is true, z is true

$(x \land \neg x)$

SAT

- Short for the Boolean satisfiability problem
- Given a Boolean formula with variables, is there an assignment of true/false to the variables which makes the formula true?

$(x \lor \neg y) \land (\neg x \lor z)$

Yes: x is true, z is true

$(x \land \neg x)$

No

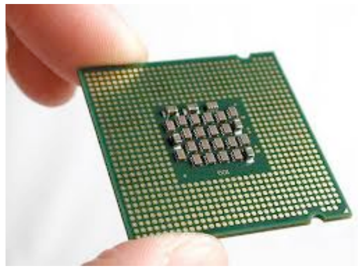
Relevance

Widespread usage in hardware and software verification

-Verification as in _proving_ the system does what we intend
-Much stronger guarantees than testing
-Testing can prove the existence of a bug (a failed test), whereas verification proves the absence of bugs (relative to the theorems proven)

Relevance

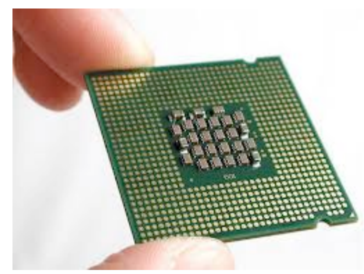
Widespread usage in hardware and software verification

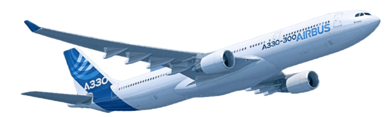


-Circuits can be represented as Boolean formulas
-Can basically phrase proofs as Circuit $\wedge$ BadThing. If unsatisfiable, then BadThing cannot occur. If satisfiable, then the solution gives the circumstance under which BadThing occurs.
-Many details omitted (entire careers are based on this stuff)

Relevance

Widespread usage in hardware and software verification

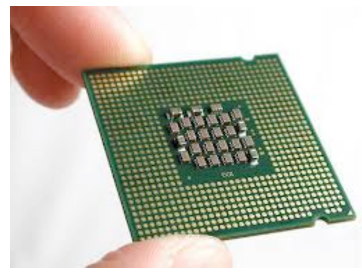




-(Likely) used by AirBus to verify that flight control software does the right thing
-Lots of proprietary details so it's not 100% clear how this verification works, but SAT is still relevant to the problem

Relevance

Widespread usage in hardware and software verification



-Nasa uses software verification for a variety of tasks; SAT is relevant, though other techniques are used, too

Relevance to Logic Programming

- Methods for solving SAT can be used to execute logic programs
- Logic programming can be phrased as SAT with some additional stuff

Semantic Tableau

- One method for solving SAT instances
- Basic idea: iterate over the formula
  - Maintain subformulas that must be true
  - Learn which variables must be true/false
  - Stop at conflicts (unsatisfiable), or when no subformulas remain (have solution)

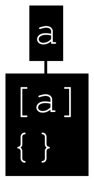
-There are many methods to this

Positive Literals

`a`

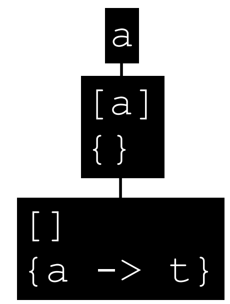
–As in, the input formula is simply “a”

Positive Literals



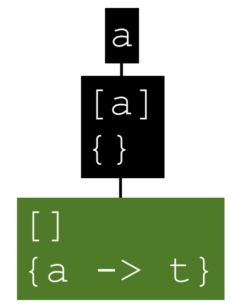
-One subformula must be true: a
-Initially, we don't know what any variables must map to

Positive Literals



-For formula “a” to be true, it must be the case that a is true

Positive Literals



-No subformulas remain, so we are done. The satisfying solution is that a must be true.

Negative Literals

¬a

–As in, the input formula is simply "¬a"

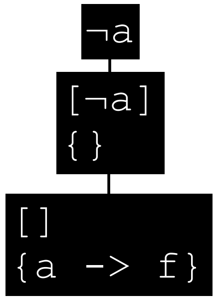
Negative Literals

¬a

[¬a]
{ }

–One subformula must be true: ¬a
–Initially, we don't know what any variables must map to

Negative Literals



-For subformula "¬a" to be true, it must be the case that a is false

Negative Literals

```
¬a

[¬a]
{}

[]
{a -> f}
```

-No subformulas remain, so we are done. The satisfying solution is that “a” must be false.

Logical And

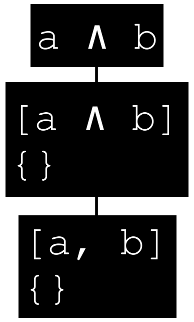
`a ∧ b`

Logical And

```
a ∧ b
  |
[a ∧ b]
{}
```

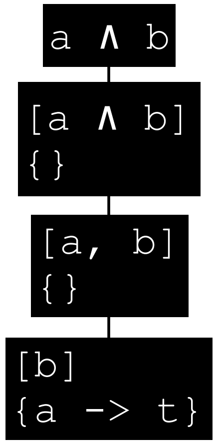
–Initially, one subformula must be true: a ∧ b
–Initially, we don't know what any variable must map to

Logical And



–For a ∧ b to be true, subformulas a and b must both be true

Logical And



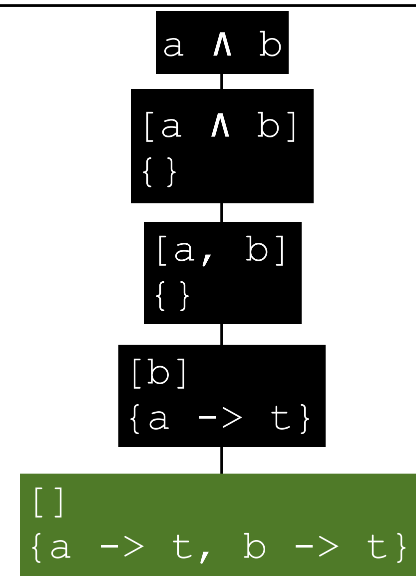
-From the positive literal case, for formula a to be true, variable a must be true

Logical And

```
a ∧ b

[a ∧ b]
{}

[a, b]
{}

[b]
{a -> t}

[]
{a -> t, b -> t}
```

-From the positive literal case, for formula b to be true, variable b must be true

Logical And



-No subformulas remain, so we are done with the solution that both a and b must be true

Logical And

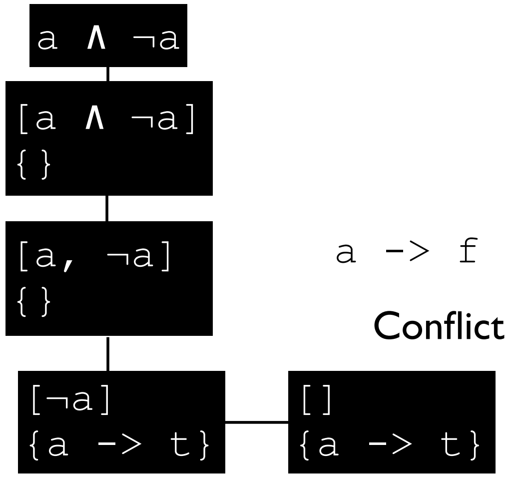
$a \land \neg a$

–Alternative example, showing a conflict

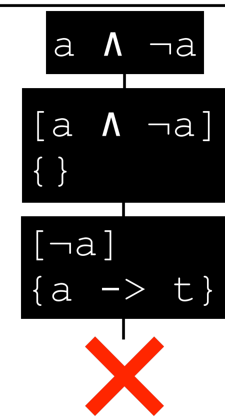
Logical And

```
a ∧ ¬a
[a ∧ ¬a]
{}
```

Logical And



Logical And



-Now we have a problem: for formula ¬a to be true, it must be the case that variable a is false
-We’ve already recorded that variable a must be true, which is the opposite of what we expect.
-As such, we have a conflict - this formula is unsatisfiable

Exercise: First Side of SAT Sheet

Logical Or

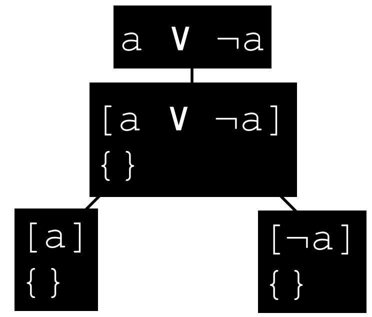
$a \lor \neg a$

Logical Or

a ∨ ¬a

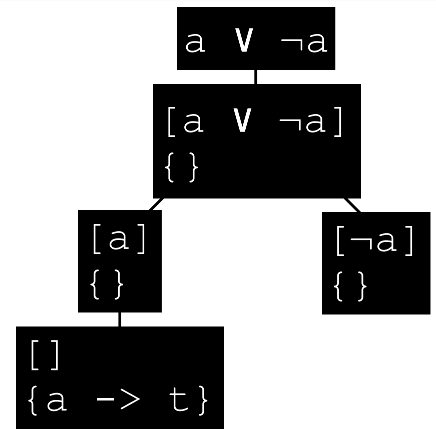
[a ∨ ¬a]
{ }

Logical Or



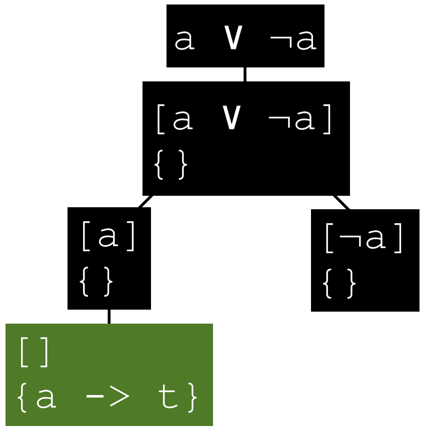
–World splits on or: in one world we pick the left subformula, and in another we pick the right

Logical Or



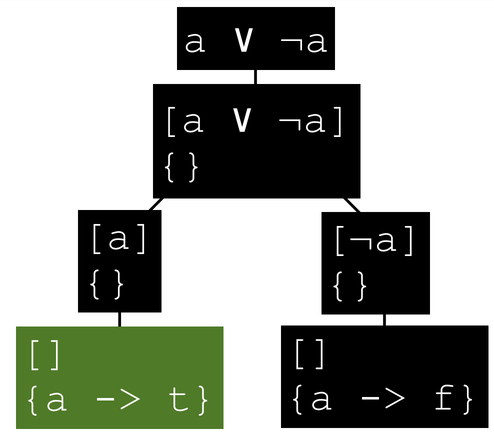
–World splits on or: in one world we pick the left subformula, and in another we pick the right

Logical Or



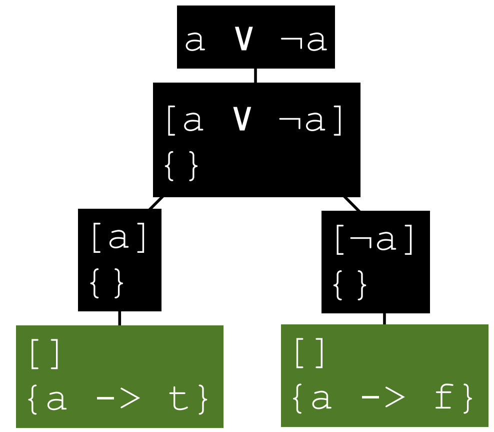
–World splits on or: in one world we pick the left subformula, and in another we pick the right

Logical Or



-World splits on or: in one world we pick the left subformula, and in another we pick the right

Logical Or



-World splits on or: in one world we pick the left subformula, and in another we pick the right

Examples

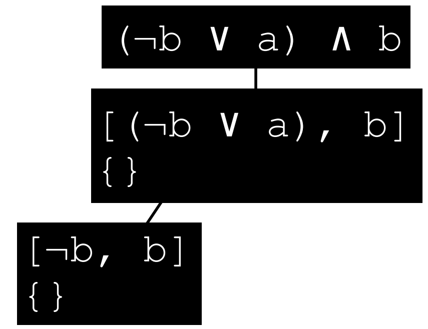
Example 1:

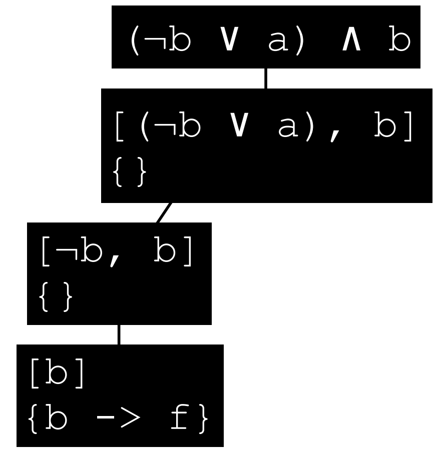
$(\neg b \lor a) \land b$

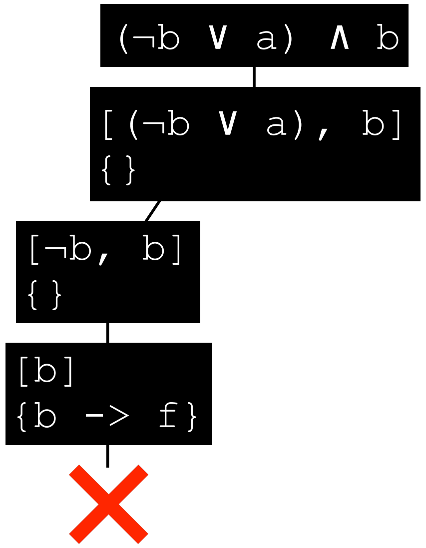
$(\neg b \lor a) \land b$

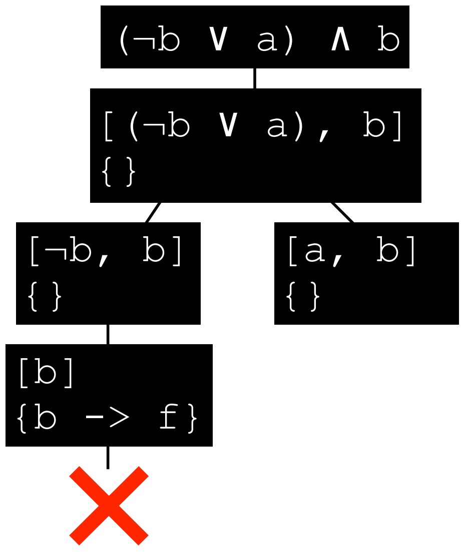
(¬b ∨ a) ∧ b

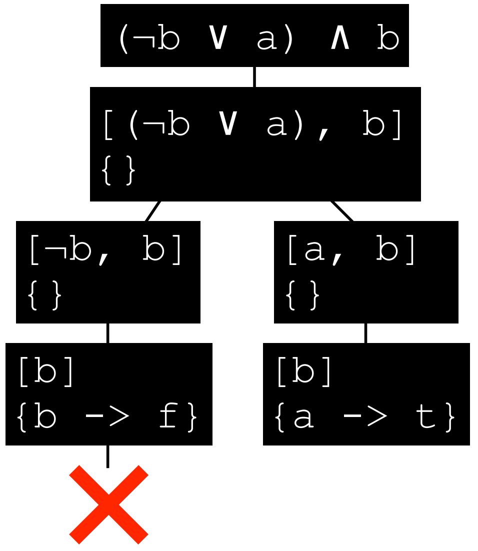
[(¬b ∨ a), b]
{}


(¬b ∨ a) ∧ b
[(¬b ∨ a), b]
{}
[¬b, b]
{}

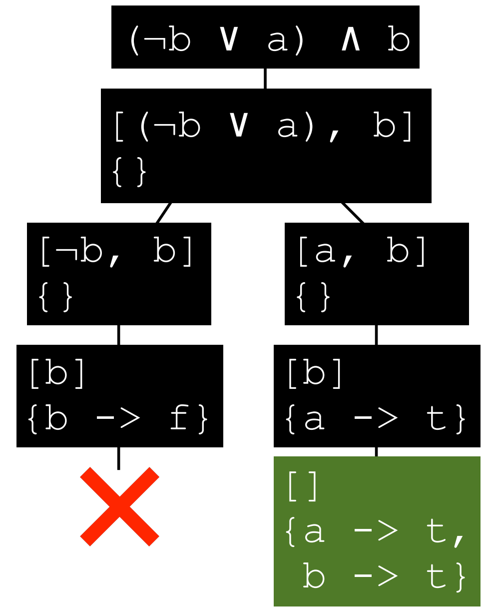

(¬b ∨ a) ∧ b
[(¬b ∨ a), b]
{}
[¬b, b]
{}
[b]
{b -> f}


(¬b ∨ a) ∧ b
[(¬b ∨ a), b]
{}
[¬b, b]
{}
[b]
{b -> f}


(¬b ∨ a) ∧ b
[(¬b ∨ a), b]
{}
[¬b, b]
{}
[a, b]
{}
[b]
{b -> f}



(¬b ∨ a) ∧ b
[(¬b ∨ a), b]
{}
[¬b, b]
{}
[a, b]
{}
[b]
{b -> f}
[b]
{a -> t}



(¬b ∨ a) ∧ b
[(¬b ∨ a), b]
{}
[¬b, b]
{}
[a, b]
{}
[b]
{b -> f}
[b]
{a -> t}
[]
{a -> t,
b -> t}

Example 2:

$(\mathtt{x} \lor \neg\mathtt{y}) \land (\neg\mathtt{x} \lor \mathtt{z})$

$(x \lor \neg y) \land (\neg x \lor z)$

$(x \lor \neg y) \land (\neg x \lor z)$

$[(x \lor \neg y), (\neg x \lor z)]$
$\{\}$

(x ∨ ¬y) ∧ (¬x ∨ z)

[(x ∨ ¬y), (¬x ∨ z)]
{}

[x, (¬x ∨ z)]
{}

```
(x ∨ ¬y) ∧ (¬x ∨ z)

[(x ∨ ¬y), (¬x ∨ z)]
{}

[x, (¬x ∨ z)]
{}

[(¬x ∨ z)]
{x -> t}
```

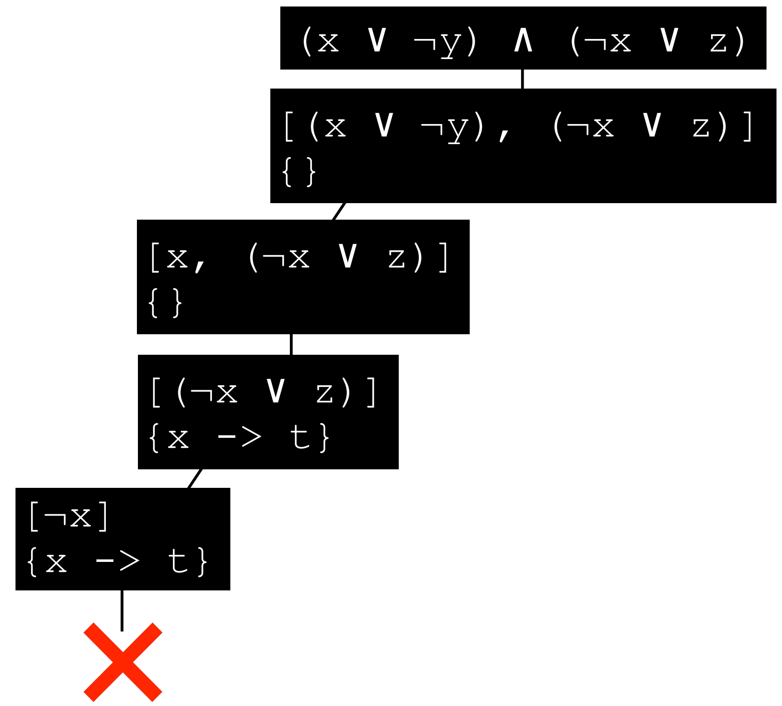
(x ∨ ¬y) ∧ (¬x ∨ z)

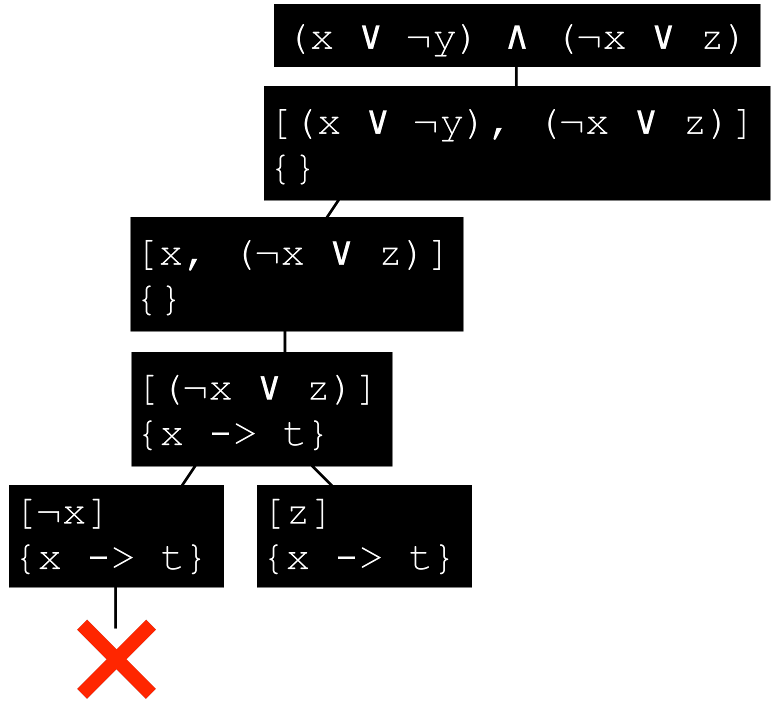
[(x ∨ ¬y), (¬x ∨ z)]
{}

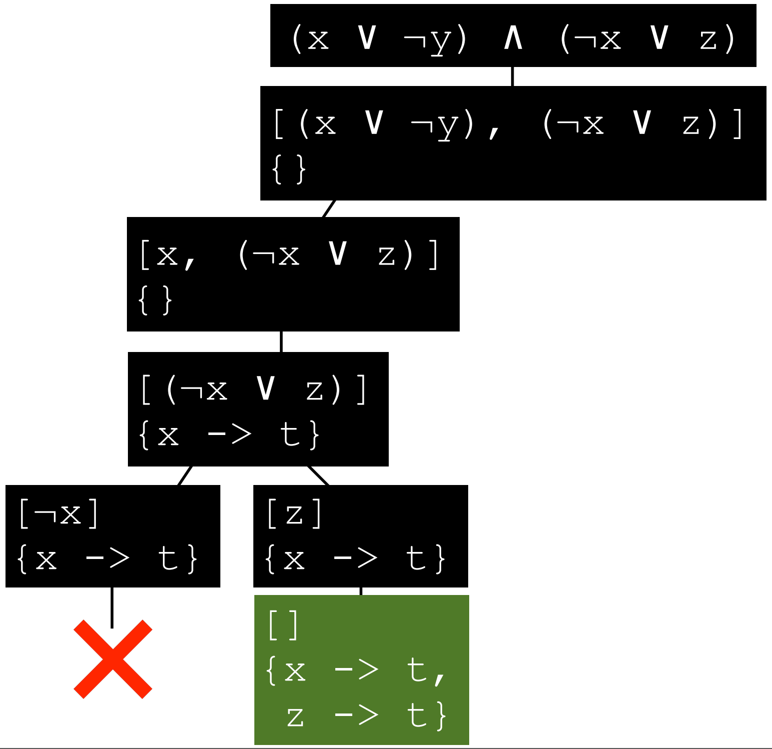
[x, (¬x ∨ z)]
{}

[(¬x ∨ z)]
{x -> t}

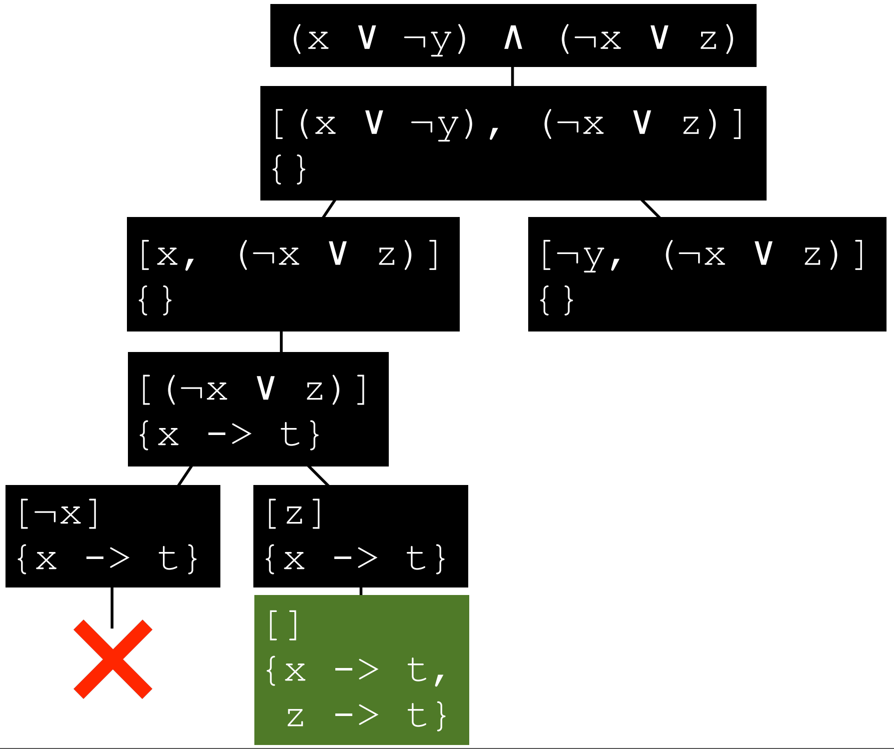
[¬x]
{x -> t}

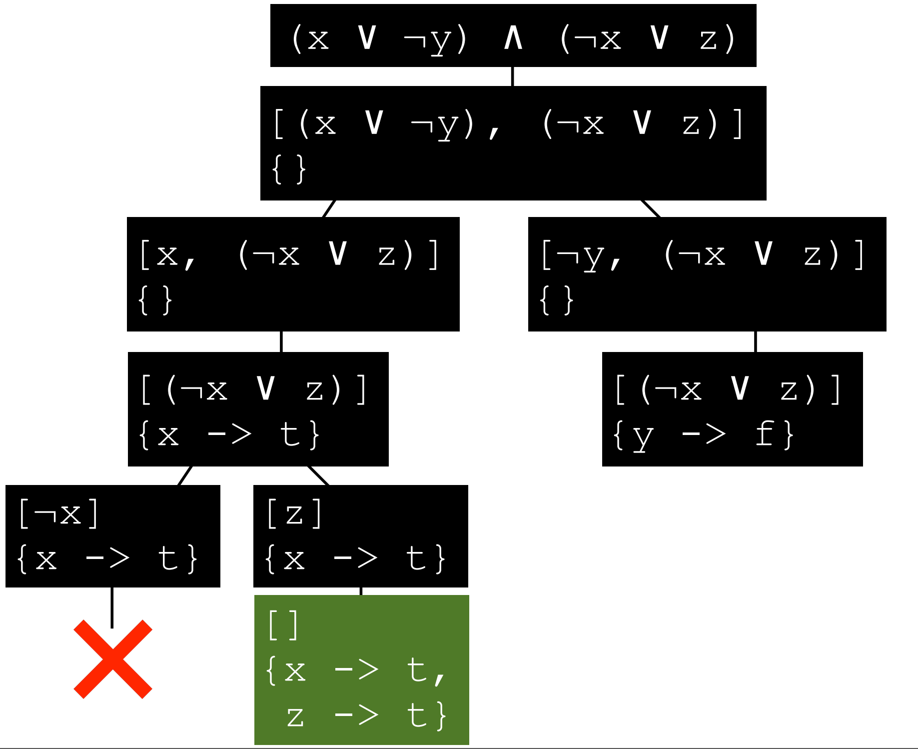

(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[¬x]
{x -> t}


(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[¬x]
{x -> t}
[z]
{x -> t}

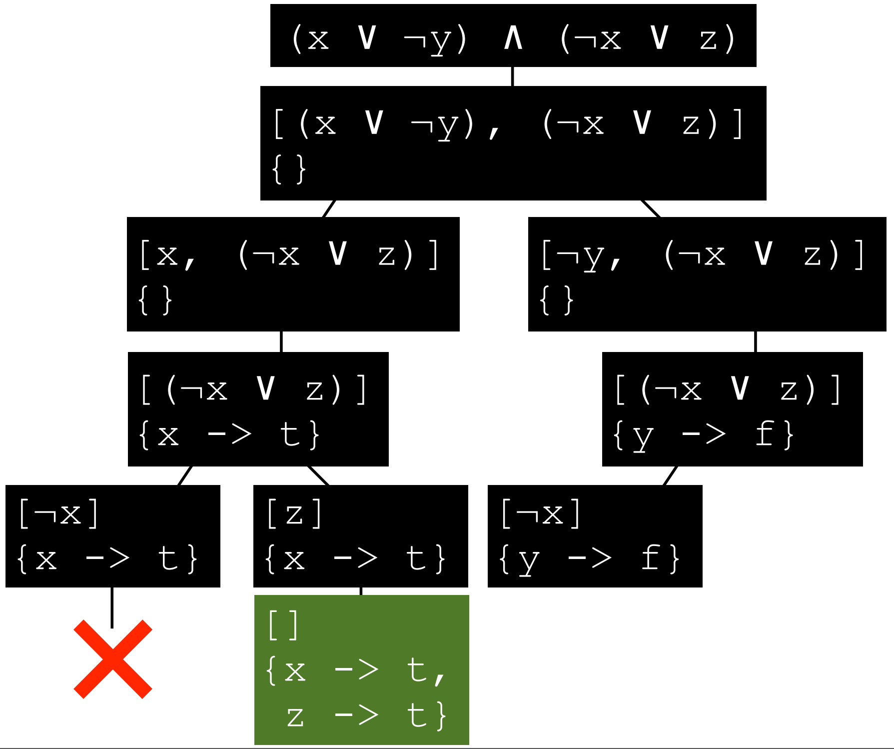


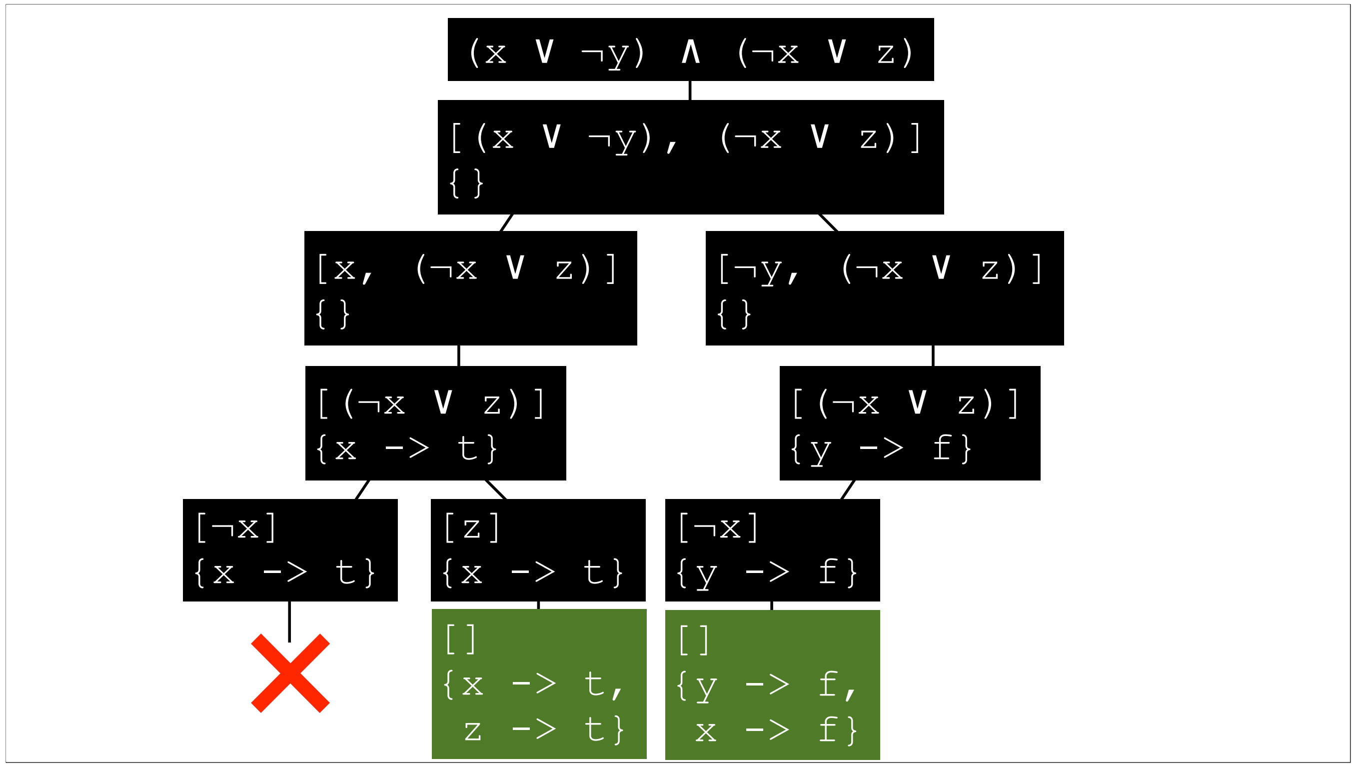
(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[¬x]
{x -> t}
[z]
{x -> t}
[]
{x -> t,
z -> t}


(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[¬y, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[¬x]
{x -> t}
[z]
{x -> t}
[]
{x -> t,
z -> t}

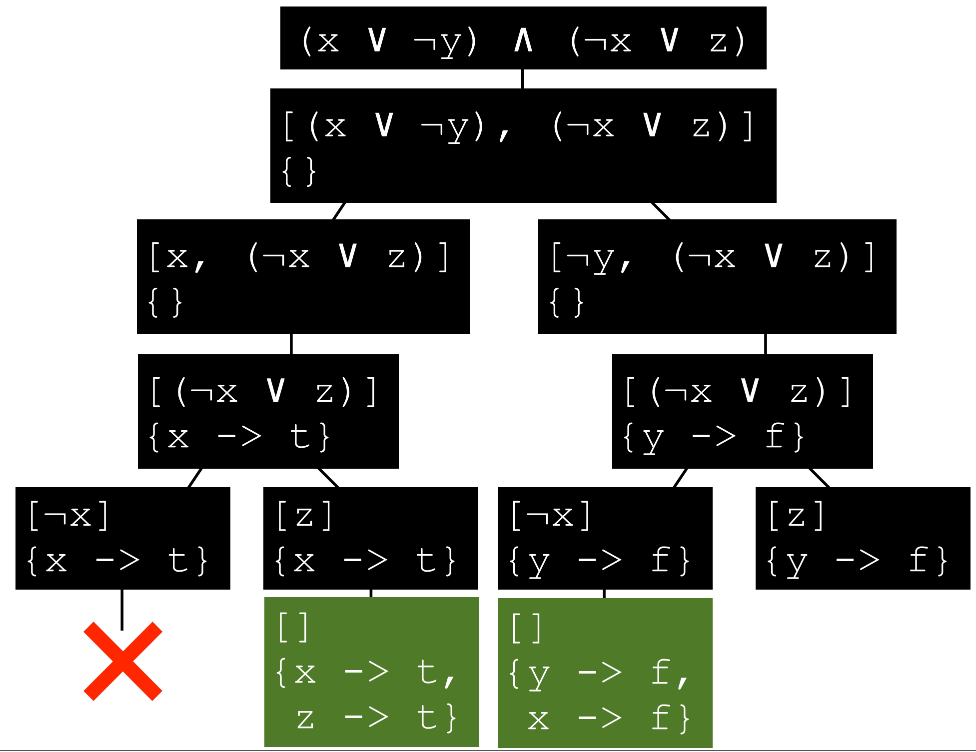


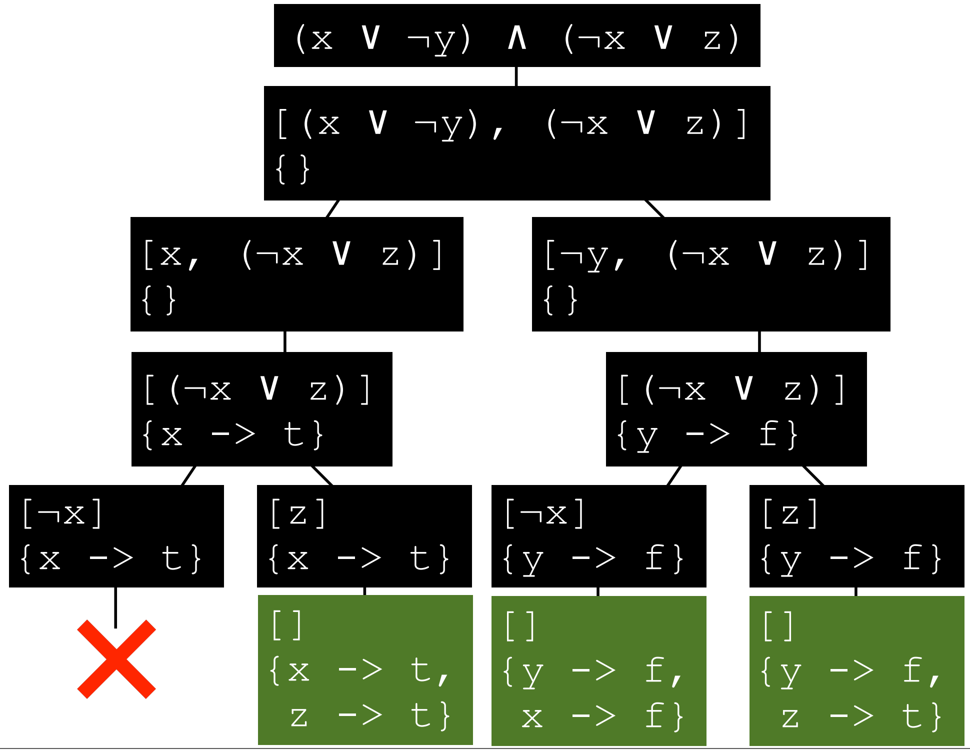
(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[¬y, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[(¬x ∨ z)]
{y -> f}
[¬x]
{x -> t}
[z]
{x -> t}
[]
{x -> t,
z -> t}


(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[¬y, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[(¬x ∨ z)]
{y -> f}
[¬x]
{x -> t}
[z]
{x -> t}
[¬x]
{y -> f}
[]
{x -> t,
z -> t}



(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[¬y, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[(¬x ∨ z)]
{y -> f}
[¬x]
{x -> t}
[z]
{x -> t}
[¬x]
{y -> f}
[]
{x -> t,
z -> t}
[]
{y -> f,
x -> f}


(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[¬y, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[(¬x ∨ z)]
{y -> f}
[¬x]
{x -> t}
[z]
{x -> t}
[¬x]
{y -> f}
[z]
{y -> f}
[]
{x -> t,
z -> t}
[]
{y -> f,
x -> f}


(x ∨ ¬y) ∧ (¬x ∨ z)
[(x ∨ ¬y), (¬x ∨ z)]
{}
[x, (¬x ∨ z)]
{}
[¬y, (¬x ∨ z)]
{}
[(¬x ∨ z)]
{x -> t}
[(¬x ∨ z)]
{y -> f}
[¬x]
{x -> t}
[z]
{x -> t}
[¬x]
{y -> f}
[z]
{y -> f}
[]
{x -> t,
z -> t}
[]
{y -> f,
x -> f}
[]
{y -> f,
z -> t}

Exercise: Second Side of SAT Sheet