DPLL algorithm

backtracking + unit propagation + pure literal rule

The backtracking algorithm

in general, to find a set of values satisfying some conditions:

set a variable to each possible value in turn

for each value, recursively repeat

Backtracking for satisfiability

find values of $x_1, \ldots, x_n$ satisfying the formula F

algorithm:

1. choose a variable x_i
2. check satisfiability of $F + (x_i{=}true)$
3. check satisfiability of $F + (x_i{=}false)$

(more details later)

Satisfiability: recursive calls

the two recursive calls are: "check satisfiability of $F + (x_i{=}value)$"

in general: check satisfiability when some variables already have a value

partial interpretation = assigns true/false to *some* variables

Backtracking with partial interpretation

algorithm (some parts missing):

boolean sat(formula F, partial_interpretation I)

1. ... (see below)
2. choose x_i that I does not assign
3. return sat(F, I ∪ { x_i=true }) or sat(F, I ∪ { x_i=false })

satisfiability of F = satisfiability of F with $I=\emptyset$

missing: base case of recursion, choice of x_i

Base case

recursion adds a x_i=*value* to I

at some point, all variables are assigned

we can now check whether F is true or false

but:

sometimes, we can check whether F is true or false even if some variables are still unassigned

Value of formulae under partial interpretations

in the formula F:

- replace each x_i that is assigned in I with its truth value
  (e.g. if I contains x_i=*true* replace each occurrence of x_i with *true*)
- simplify using rules:
  - *something* $\wedge$ *true* = *something*
  - *something* $\wedge$ *false* = *false*
  - *something* $\vee$ *true* = *true*
  - *something* $\vee$ *false* = *something*

result could be:

1. true
2. false
3. some formula containing only unassigned variables

in the first two cases, the formula has a value that does not depend on the unassigned variables

---

Partial interpretation, example 1

$I=\{x=true,\ z=false\}$

$F = \{\ x \vee y,\ \neg x \vee \neg y \vee z\ \}$

replace variables with values:

$$\begin{aligned} F &= \{\ x \vee y,\ \neg x \vee \neg y \vee z\ \} = \\ &\{\ true \vee y,\ \neg true \vee \neg y \vee false\ \} = \\ &\{\ true,\ \neg y\ \} = \\ &\{\ \neg y\ \} \end{aligned}$$

formula is not true nor false

value depends on the value of variable y

Partial interpretation, example 2

$I=\{x=true,\ z=false\}$

$F = \{\ \neg x \vee y,\ \neg x \vee z\ \}$

replace variables with values:

$$\begin{aligned} F &= \{\ \neg x \vee y,\ \neg x \vee z\ \} = \\ &\{\ \neg true \vee y,\ \neg true \vee false\ \} \\ &\{\ false \vee y,\ false \vee false\ \} = \\ &\{\ y,\ false \vee false\ \} = \\ &\{\ y,\ false\ \} \end{aligned}$$

formula is **false**

all clauses have to be satisfied

even a single false clause implies that the formula is false

(even if the first clause were *true* instead of z, formula would have been false

---

Partial interpretation, example 3

$I=\{x=true,\ z=false\}$

$F = \{x \vee y \vee z,\ \neg y \vee \neg z\}$

$$\begin{aligned} F &= \{x \vee y \vee z,\ \neg y \vee \neg z\} = \\ &\{true \vee y \vee false,\ \neg y \vee \neg false\} = \\ &\{true \vee y \vee false,\ \neg y \vee true\} = \\ &\{true,\ true\} \end{aligned}$$

all clauses are true

formula is true

Partial interpretation and formula

given a partial interpretation, a formula could be:

- true (denoted $I \Rightarrow F$)
- false (denoted $I \Rightarrow \neg F$)
- neither true nor false
  (its value depends on the unassigned variables)

in backtracking:

> if the formula is true or false (first two cases) according to the partial interpretation, there is no need to perform the recursive calls

---

Backtracking with check of partial interpretation

boolean sat(formula F, partial_interpretation I)

- if ($I \Rightarrow F$) return true
- if ($I \Rightarrow \neg F$) return false
- choose x_i that I does not assign
- return sat(F, I ∪ { x_i=true }) or sat(F, I ∪ { x_i=false })

Avoid second recursive calls

implicit in most imperative programming language: if the first argument of an *or* is true, do not evaluate the others

for clarity, backtracking is as follows:

boolean sat(formula F, partial_interpretation I)

- if ($I \Rightarrow F$) return true
- if ($I \Rightarrow \neg F$) return false
- choose x_i that I does not assign
- if sat(F, I ∪ { x_i=true }) return true
- if sat(F, I ∪ { x_i=false }) return true
- return false

Backtracking, first example

$\{ \neg x_1 \vee \neg x_2, x_1 \vee \neg x_2, \neg x_1 \vee \neg x_3 \}$

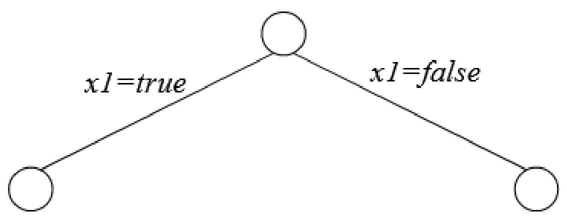
Backtracking, first example (1)

start with empty assignment *{}*

choose a variable, *for example* x_1

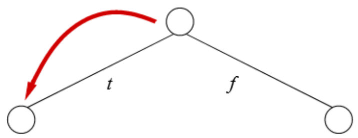
do two recursive calls with assignments $\{x_1=true\}$ and $\{x_1=false$

Backtracking, first example (2)



two recursive calls with assignments $\{x_1=true\}$ and $\{x_1=false\}$

Backtracking, first example (3)



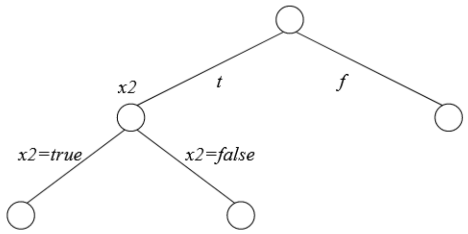
first recursive call is with assignment $\{x_1=true\}$

$\{ \neg x_1 \vee \neg x_2, x_1 \vee \neg x_2, \neg x_1 \vee \neg x_3 \}$

no clause of $\{\neg x_1 \vee \neg x_2, x_1 \vee \neg x_2, \neg x_1 \vee \neg x_3\}$ is falsified by $\{x_1=true\}$

no contradiction: choose an unassigned variable

Backtracking, first example (4)

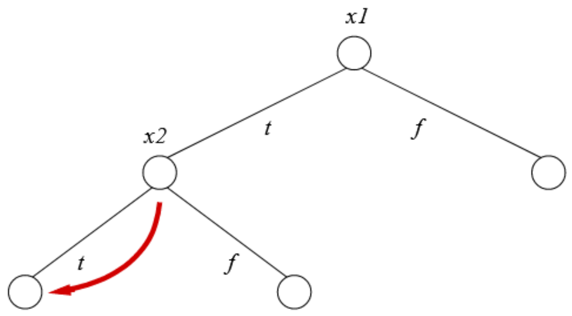


branching variable x_2 (for example)

do two recursive calls adding the two possible evaluations of x_2 to the original one

partial interpretations in the recursive calls are then $\{x_1=true, x_2=true\}$ and $\{x_1=true, x_2=false\}$

Backtracking, first example (5)

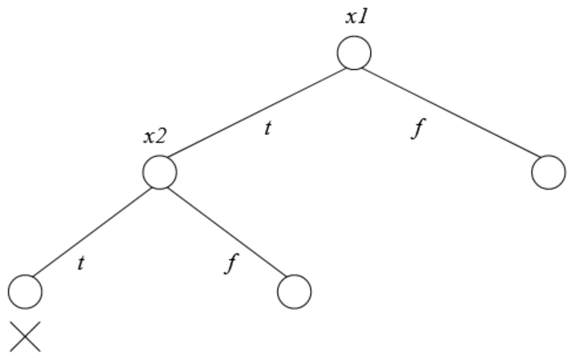


first recursive call with assignment $\{x_1=true,\ x_2=true\}$:

in $\{\neg x_1 \lor \neg x_2,\ x_1 \lor \neg x_2,\ \neg x_1 \lor \neg x_3\}$, the clause $\neg x_1 \lor \neg x_2$ is falsified

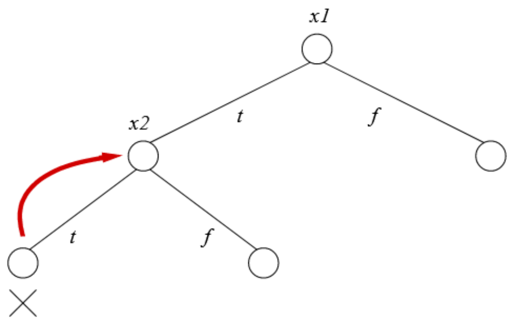
---

Backtracking, first example (6)



contradiction, close branch of the tree

Backtracking, first example (7)

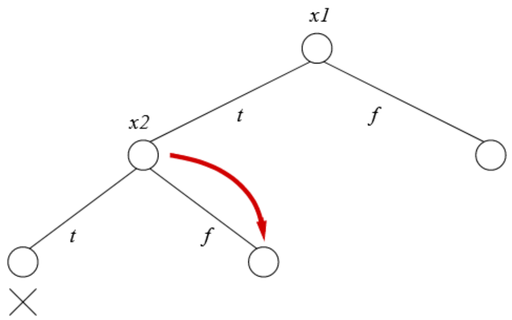


go back to node labeled x_2

$x_2=true$ already tried

now try $x_2=false$

Backtracking, first example (8)

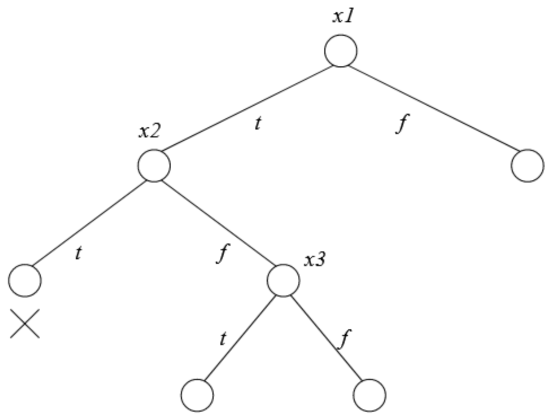


assignment $\{x_1=true,\ x_2=false\}$

formula $\{\neg x_1 \lor \neg x_2,\ x_1 \lor \neg x_2,\ \neg x_1 \lor \neg x_3\}$, is not falsified

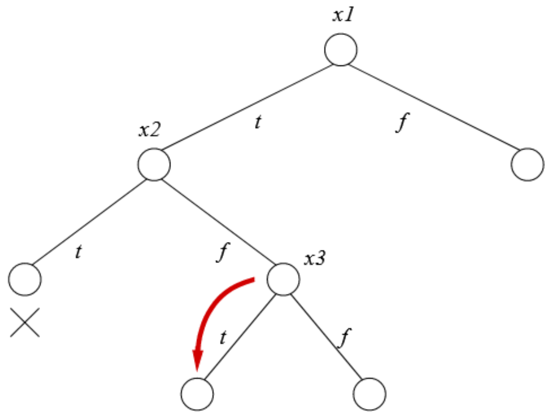
choose variable: only left unassigned is x_3

Backtracking, first example (9)



two recursive calls: $x_3=true$, $x_3=false$

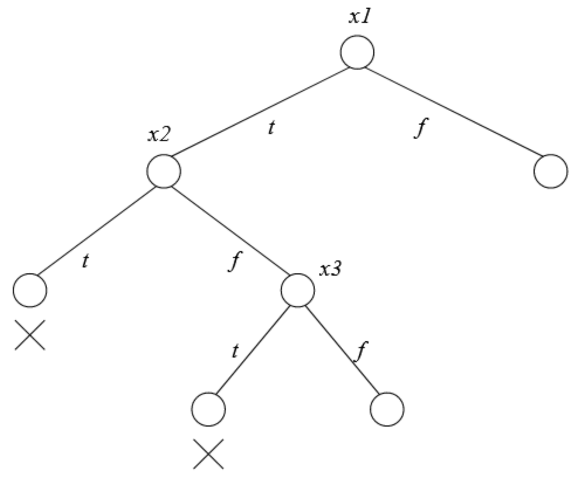
Backtracking, first example (10)



first recursive call has assignment $\{x_1=true,\ x_2=false,\ x_3=true\}$

in formula $\{\neg x_1 \lor \neg x_2,\ x_1 \lor \neg x_2,\ \neg x_1 \lor \neg x_3\}$, the clause $\neg x_1 \lor \neg x_3$ is falsified

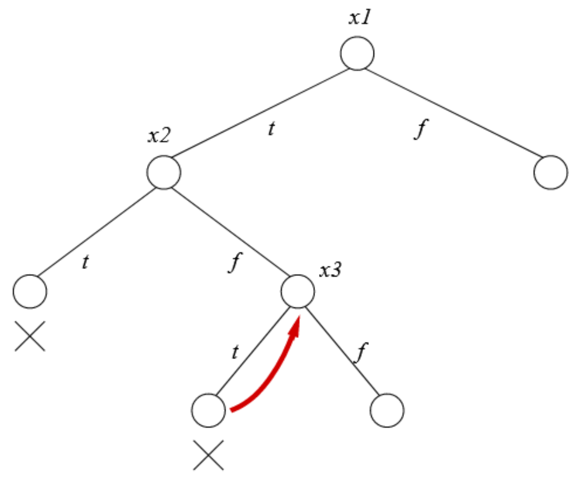
Backtracking, first example (11)



clause is falsified=formula is falsified

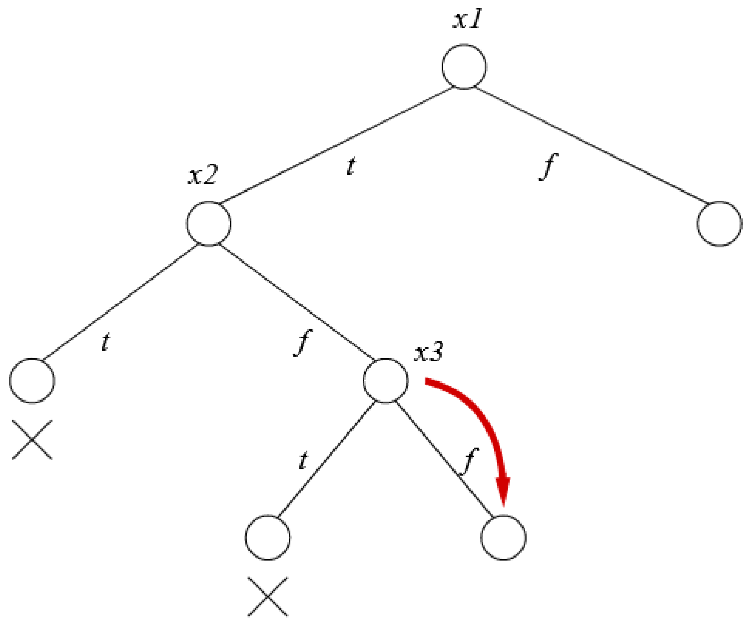
close branch

Backtracking, first example (12)



backtrack to node labeled x_3

Backtracking, first example (13)



second recursive call for x_3

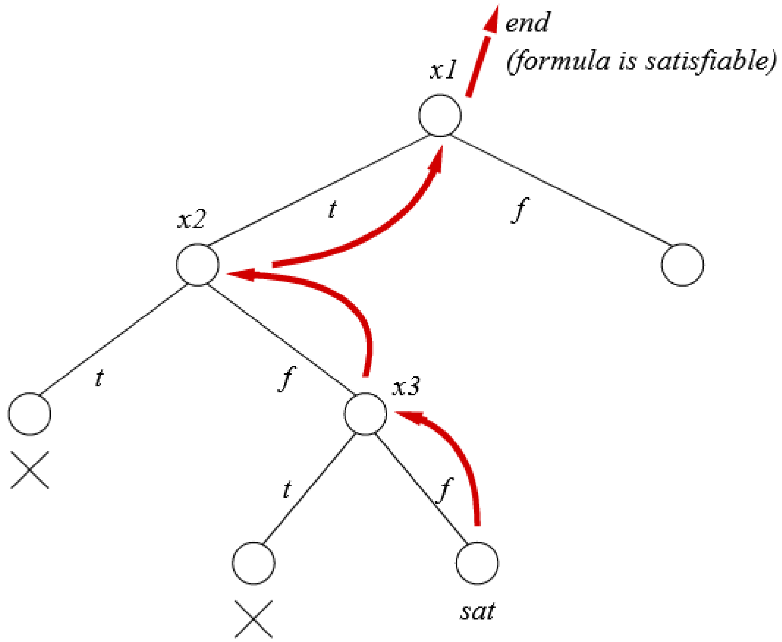
value x_3=*false*

assignment is $\{x_1=true,\ x_2=false,\ x_3=false\}$

all clauses in $\{\neg x_1 \vee \neg x_2,\ x_1 \vee \neg x_2,\ \neg x_1 \vee \neg x_3\}$, are satisfied!

$\neg x_1 \vee \neg x_2$
    because x_2=*false*
$x_1 \vee \neg x_2$
    because x_1=*true*
$\neg x_1 \vee \neg x_3$
    because x_3=*false*

Backtracking, first example (14)



no other recursive calls

if a subcall returns *true*, the call returns *true* as well

this means: in this case, we go back to original call and return *true*

model found, no need to go ahead

formula is satisfiable

Backtracking, second example

$\{ \neg x_1 \vee \neg x_2, \neg x_1 \vee x_2, x_1 \vee \neg x_2, x_2 \vee \neg x_3, x_1 \vee x_3 \}$

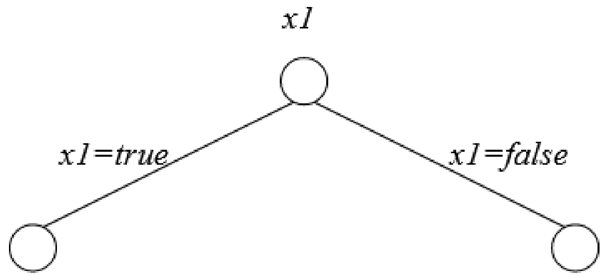
start with empty assignment

formula is not false under this interpretation

choose a variable

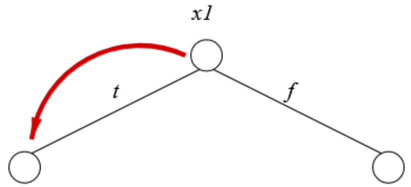
as an example, we choose x_1

Backtracking, second example (1)



branch on x_1

Backtracking, second example (2)

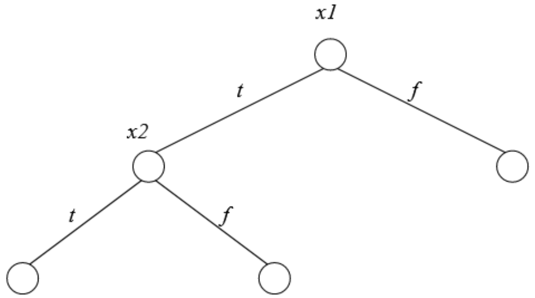


first recursive calls with x_1=*true*

formula $\{ \neg x_1 \vee \neg x_2, \neg x_1 \vee x_2, x_1 \vee \neg x_2, x_2 \vee \neg x_3, x_1 \vee x_3 \}$ not made false by this assignment

choose an unassigned variable

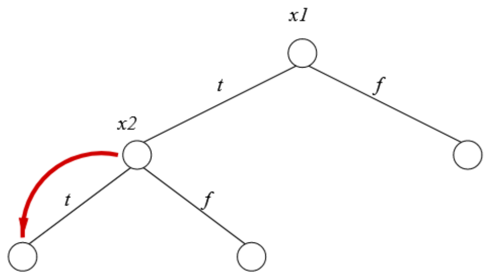
Backtracking, second example (3)



as an example, we choose x_2

two other recursive calls, with assignments $\{x_1=true, x_2=true\}$ and $\{x_1=true, x_2=false\}$

Backtracking, second example (4)



first recursive (sub)call:
assignment $\{x_1=true,\ x_2=true\}$

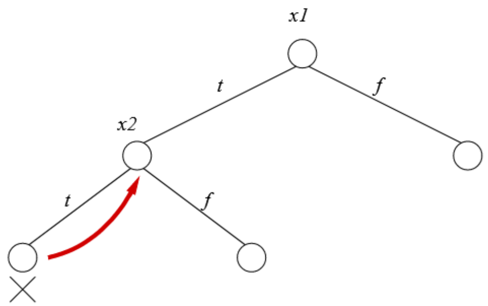
formula was $\{\ \neg x_1 \lor \neg\, x_2,\ \neg x_1 \lor x_2,\ x_1 \lor \neg x_2,\ x_2 \lor \neg x_3,\ x_1 \lor x_3\ \}$

clause $\neg x_1 \lor \neg\, x_2$ false

call returns *false*

no need to proceed any further, even if x_3 is still unassigned

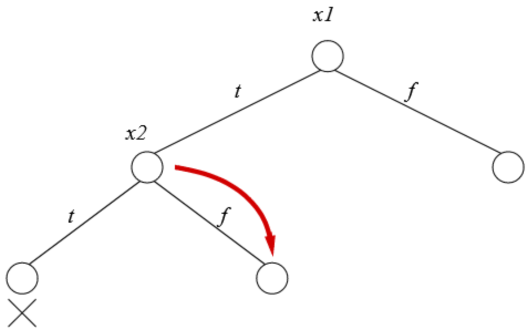
Backtracking, second example (5)



recursion goes back to node marked x_2

partial assignment were $\{x_1=true\}$ there

Backtracking, second example (6)



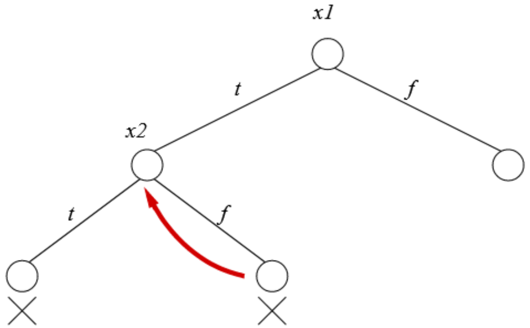
do second recursive (sub)call adding $x_2=false$ to $x_1=true$

$\{\ \neg x_1 \lor \neg\, x_2,\ \neg x_1 \lor x_2,\ x_1 \lor \neg x_2,\ x_2 \lor \neg x_3,\ x_1 \lor x_3\ \}$

clause $\neg x_1 \lor x_2$ false

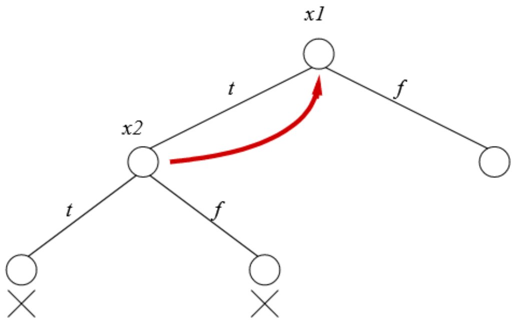
close branch

Backtracking, second example (7)



branch closed, go back to x_2

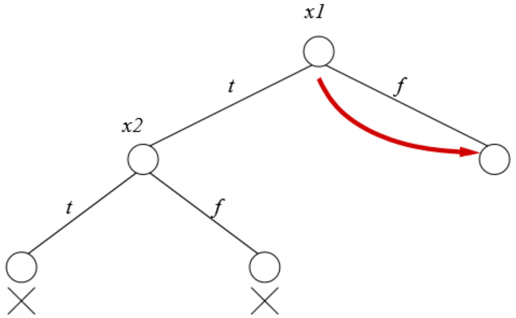
Backtracking, second example (8)



both recursive subcalls returned false, call returns false

go back to the first call, where $x_1=false$ is left to try

Backtracking, second example (9)



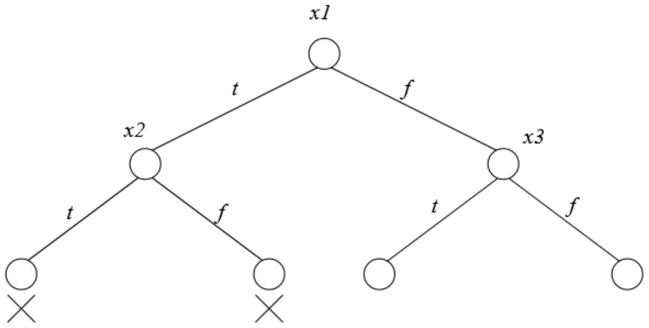
partial assignment is $\{x_1=false\}$

$\{\ \neg x_1 \lor \neg\, x_2,\ \neg x_1 \lor x_2,\ x_1 \lor \neg x_2,\ x_2 \lor \neg x_3,\ x_1 \lor x_3\ \}$

formula is not false

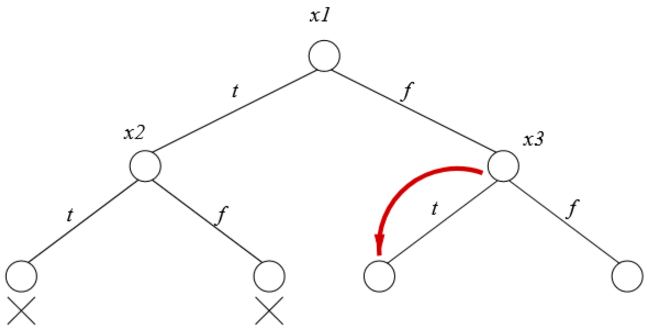
choose a variable

Backtracking, second example (10)



as an example, we choose x_3

Backtracking, second example (11)



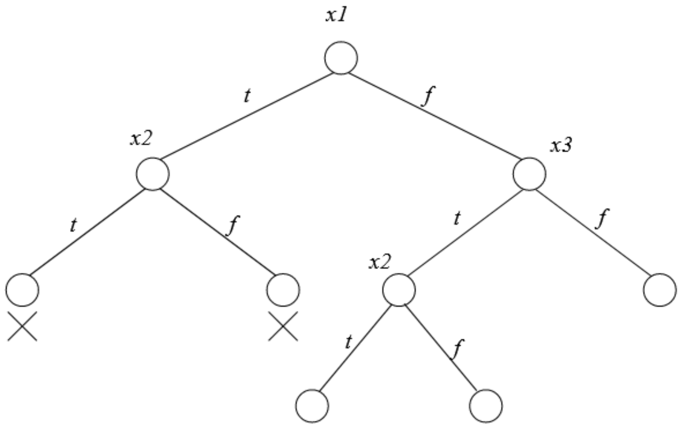
recursive call with partial assignment $\{x_1=false,\ x_3=true\}$

$\{\neg x_1 \vee \neg x_2,\ \neg x_1 \vee x_2,\ x_1 \vee \neg x_2,\ x_2 \vee \neg x_3,\ x_1 \vee x_3\}$

formula is not false in this assignment

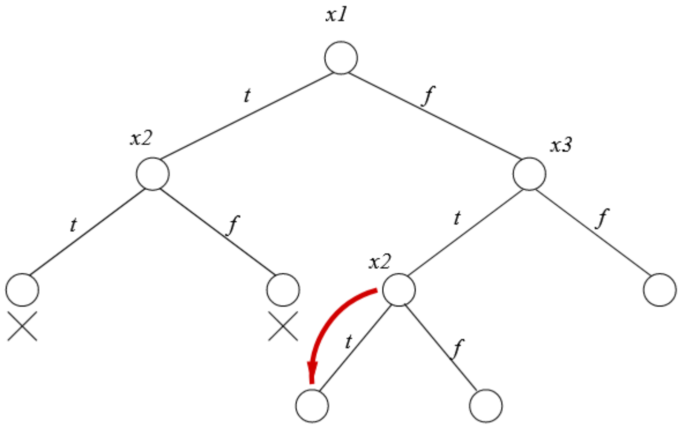
choose another variable and set it to *true* and *false*

Backtracking, second example (12)



only unassigned variable left is x_2

Backtracking, second example (13)

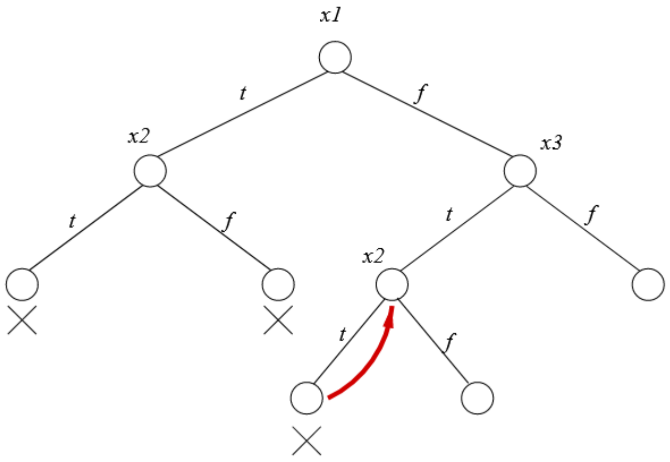


assignment $\{x_1=false,\ x_3=true,\ x_2=true\}$

$\{\neg x_1 \vee \neg x_2,\ \neg x_1 \vee x_2,\ x_1 \vee \neg x_2,\ x_2 \vee \neg x_3,\ x_1 \vee x_3\}$

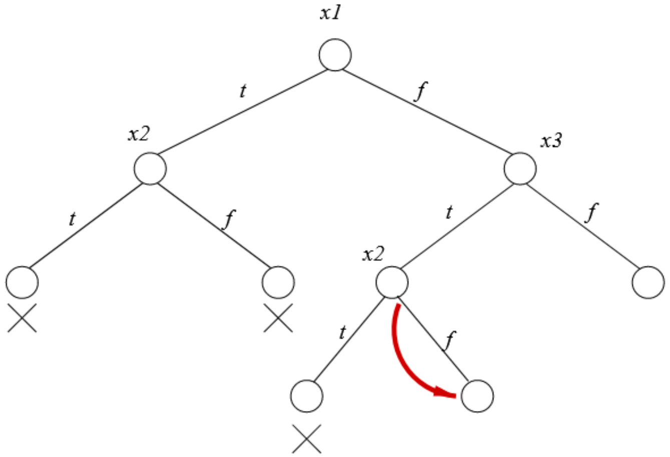
clause $x_1 \vee \neg x_2$ is falsified

Backtracking, second example (14)



backtrack to x_2

Backtracking, second example (15)

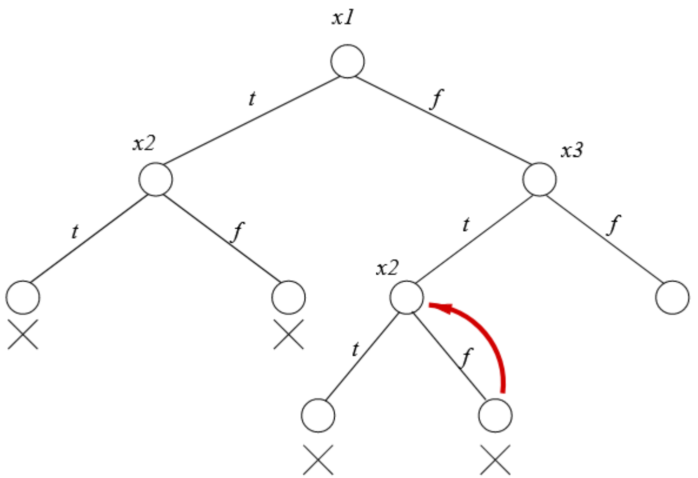


assignment $\{x_1=false,\ x_3=true,\ x_2=false\}$

$\{\neg x_1 \vee \neg x_2,\ \neg x_1 \vee x_2,\ x_1 \vee \neg x_2,\ x_2 \vee \neg x_3,\ x_1 \vee x_3\}$

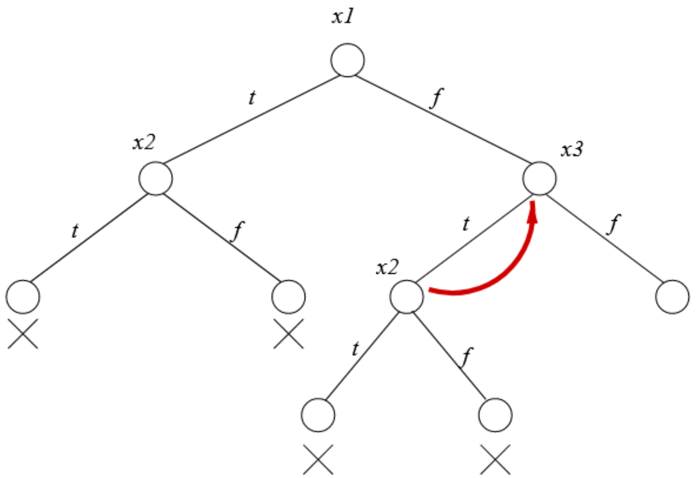
clause $x_2 \vee \neg x_3$ is falsified

Backtracking, second example (16)



backtrack to x_2

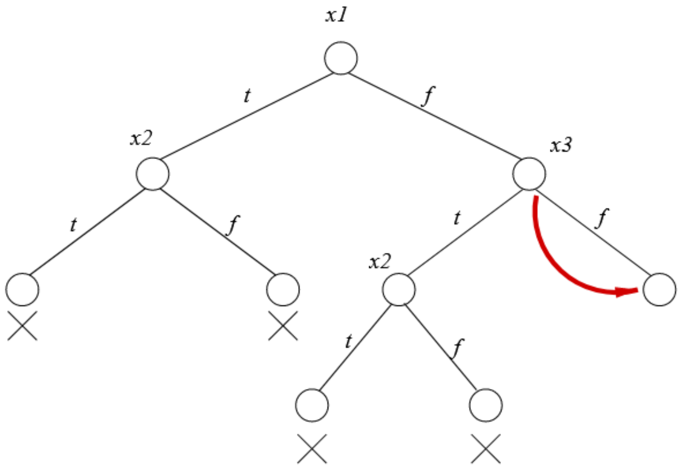
Backtracking, second example (17)



both calls from node x_2 returned *false*

go back to node x_3

Backtracking, second example (18)

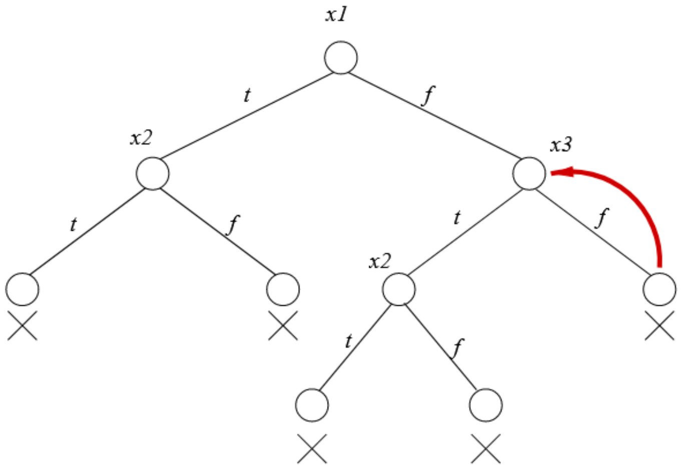


assignment $\{x_1=false,\ x_3=false\}$

$\{ \neg x_1 \vee \neg x_2,\ \neg x_1 \vee x_2,\ x_1 \vee \neg x_2,\ x_2 \vee \neg x_3,\ x_1 \vee x_3 \}$

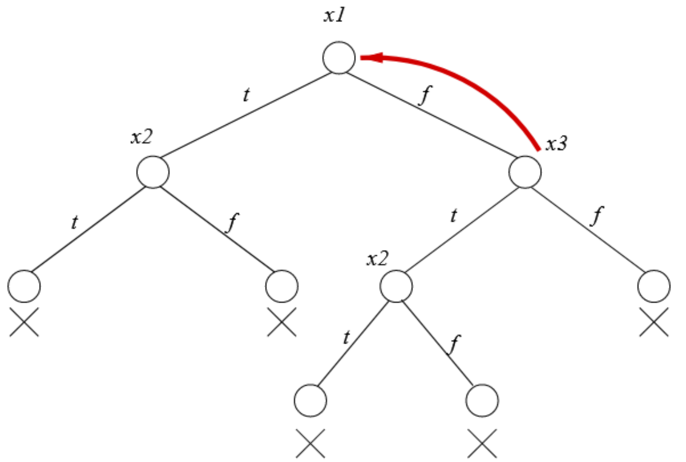
clause $x_1 \vee x_3$ falsified

Backtracking, second example (19)



calls from x_3 both returned *false*

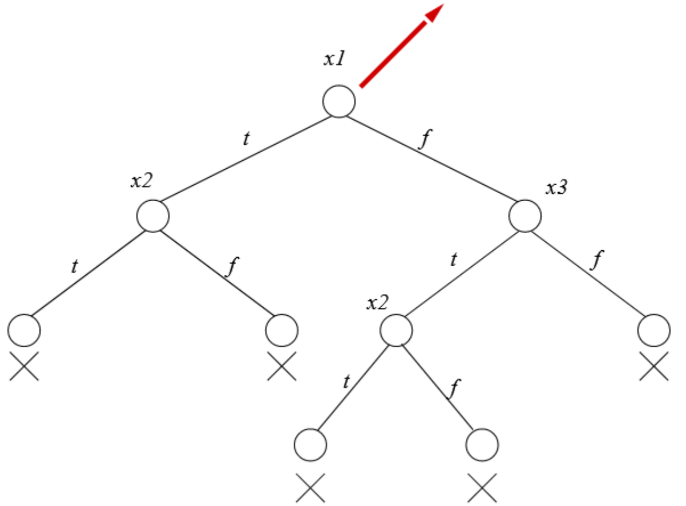
Backtracking, second example (20)



go back to x_1

we already tried x_1=*true* and x_1=*false*

Backtracking, second example (21)



return *false*

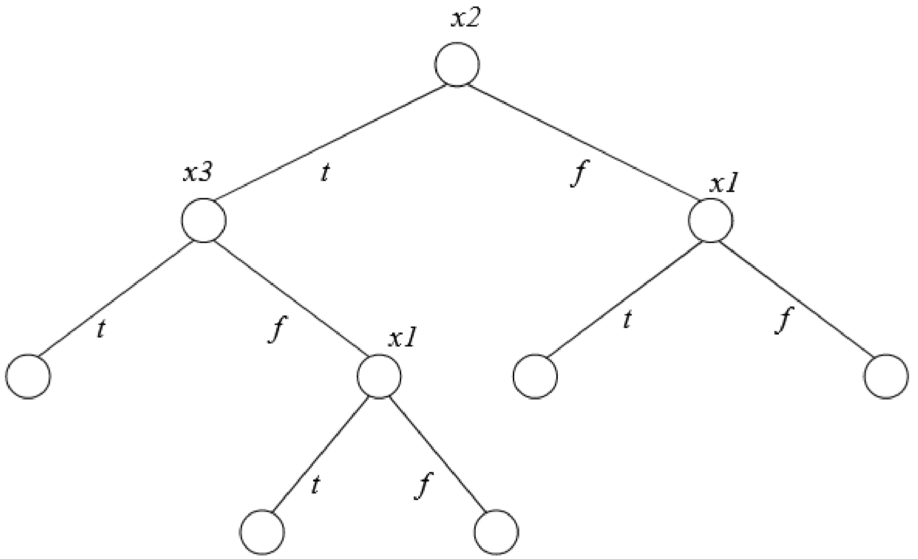
formula is unsatisfiable

Backtracking, third example

$\{ x_2 \lor x_1, \neg x_1, \neg x_2 \lor \neg x_3, x_3 \lor x_1 \}$

Backtracking, third example

$\{ x_2 \lor x_1, \neg x_1, \neg x_2 \lor \neg x_3, x_3 \lor x_1 \}$



observation: set contains the unit clause x_1

Unit propagation

in DPLL can be used for:

- simplify F (using unit clauses and values in I)
- obtain new assignments to add to I

second point is especially useful:

- base case of recursion: when $I \Rightarrow F$ or $I \Rightarrow \neg F$
- both are more likely with more variables evaluated in I
- better to have as many evaluated variables as possible

variables get a value by:

- performing the **two recursive calls** sat(F, I ∪ $\{x_i$=value$\}$)
- by unit propagation, **in the same same call**

each recursive call generates a subtree of recursive calls

one instead of two means half recursive calls (on average)

DPLL with UP

boolean sat(formula F, partial_interpretation I)

- if ($I \Rightarrow F$) return true
- if ($I \Rightarrow \neg F$) return false
- **F,I = up(F,I)**
- **if I is inconsistent return false**
- choose x_i that I does not assign
- if sat(F, I ∪ { x_i=true }) return true
- if sat(F, I ∪ { x_i=false }) return true
- return false

extra advantage: UP may discover inconsistency

Unit propagation: example

in the last of examples above, the set contains a unit clause:

$\{ x_2 \vee x_1,\ \neg x_1,\ \neg x_2 \vee \neg x_3,\ x_3 \vee x_1 \}$

up says x_1 is false

remove from clauses where occurs positive:

$x_2 \vee x_1$
  becomes $x_2 \vee \cancel{x_1}$, which is x_2

$x_3 \vee x_1$
  becomes $x_3 \vee \cancel{x_1}$, which is x_3

as a result, both x_2 and x_3 are true

clause $\neg x_2 \vee \neg x_3$ is contradicted

Unit clauses, in general

in the example, a unit clause was in the original set

may also show up with a partial assignment

Unit clauses from partial assignment

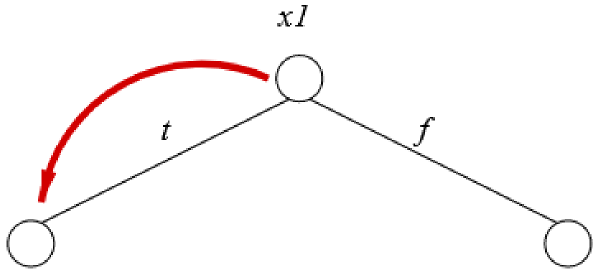
second of the examples above:

$\{ \neg x_1 \vee \neg x_2,\ \neg x_1 \vee x_2,\ x_1 \vee \neg x_2,\ x_2 \vee \neg x_3,\ x_1 \vee x_3 \}$

no unit clause in the original set

two recursive calls

first recursive call with x_1=*true*



x_1=*true* is like an additional unit clause $\{x_1\}$

apply unit propagation

Unit propagation in a recursive call

$\{ \neg x_1 \lor \neg x_2, \neg x_1 \lor x_2, x_1 \lor \neg x_2, x_2 \lor \neg x_3, x_1 \lor x_3 \}$

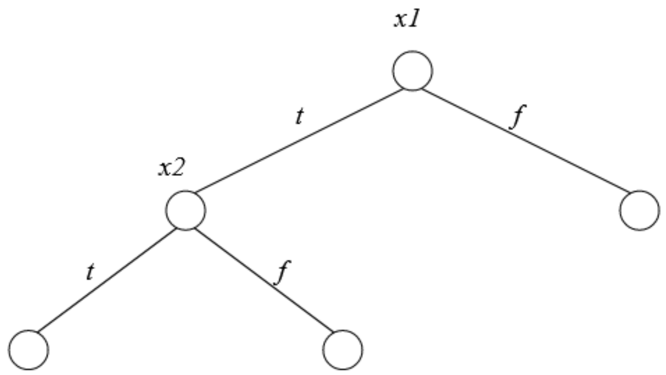
recursive call with x_1=*true*

remove x_1 where negative:

$\neg x_1 \lor \neg x_2$
  $\cancel{\neg x_1} \lor \neg x_2$ becomes $\neg x_2$
$\neg x_1 \lor x_2$
  $\cancel{\neg x_1} \lor x_2$ becomes x_2

contradiction is reached

recall that backtracking does a recursive call instead:



Unit propagation: savings

in this case, only two recursive calls are saved

more generally, the subtree rooted in the node could have been exponentially large

Pure literal rule

what about *a* in the following formula?

$\{ a \lor \neg b \lor \neg c, a \lor c, b \lor \neg d\}$

Constraining a single value

$\{ a \lor \neg b \lor \neg c, a \lor c, b \lor \neg c\}$

some occurrences of *a*

no occurrence of $\neg a$

if a variable is always positive or always negative in a formula, we say it is **pure**

Choice of value of pure literals

in general (*a* not pure):

$\{ a \lor \neg b \lor \neg c, a \lor c, b \lor \neg c, \neg a \lor b\}$

- *a=true* → a literal is made true in the first two clauses and false in the last
- *a=false* → a literal is made true in the last clause and false in the first two ones

if *a* is pure:

$\{ a \lor \neg b \lor \neg c, a \lor c, b \lor \neg c\}$

- *a=true* → a literal is made true in the first two clauses
- *a=false* → a literal is made false in the first two clauses

setting *a=true* has some advantage and no disadvantage

Pure literal rule

if a variable only occurs positively in a formula, set it to true
if a variable only occurs negated in a formula, set it to false

remove clauses containing the literal (as usual)

may create new pure literals

New pure literals

in the example $\{ a \vee \neg b \vee \neg c, a \vee c, b \vee \neg c\}$:

a only positive, set to true

remove clauses containing a

remains $\{b \vee \neg c\}$

both b and c pure
(first positive, second negative)

Pure literal rule, in practice

keep count of how many clauses contain a and $\neg a$

if a clause is removed by UP, decrease

when a counter reach zero, variable is pure

DPLL

complete algorithm:

boolean sat(formula F, partial_interpretation I)

- if ($I \Rightarrow F$) return true
- if ($I \Rightarrow \neg F$) return false
- F,I = up(F,I)
- if I is inconsistent return false
- F,I = pure(F,I)
- if F = ∅ return true
- choose x_i that I does not assign
- if sat(F, I ∪ { x_i=true }) return true
- if sat(F, I ∪ { x_i=false }) return true
- return false

Some observation about pure(F,I)

- if a is pure, it sets a=*value* (changes I)
- if a is for example positive, setting a=*true* means that all clauses containing a can be removed (already satisfied)
- same for a negative

both I and F change

but F changes only because of the removal of some clauses

we remove clauses that are satisfied:
if we remove them all, formula is satisfied

New pure literals

up(F,i) may create new pure literals

example: b is not pure here:

$\{a, a \vee \neg b, \neg a \vee \neg b \vee \neg c, \neg b \vee c\}$

performing up, we get:

$\{\neg b \vee \neg c, \neg b \vee c\}$

b is now pure (all occurrences are negative)

New unit clauses

pure(F,I) cannot create new unit clauses

reason: it only removes some clauses

no non-unary clause becomes unary by pure(F,i), since this procedure does not modify individual clauses

---

Why first up then pure

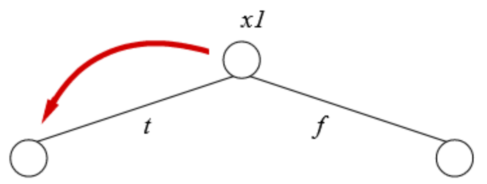
up might create new pure literals

pure cannot create new unit clauses

DPLL, complete example

$\{ \neg x_1 \vee x_3 \vee x_4, \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \neg x_3,$
$\neg x_4 \vee \neg x_2, x_2 \vee \neg x_3 \vee \neg x_1, x_2 \vee x_6 \vee x_3,$
$x_2 \vee \neg x_6 \vee \neg x_4, x_1 \vee x_5, x_1 \vee x_6,$
$\neg x_6 \vee x_3 \vee \neg x_5, x_1 \vee \neg x_3 \vee \neg x_5 \}$

DPLL, complete example (2)



choose branching variable x_1 (for example)

try x_1=*true* first

apply up and pure

DPLL, complete example (3)

with $\{x_1=true\}$ the clauses become:

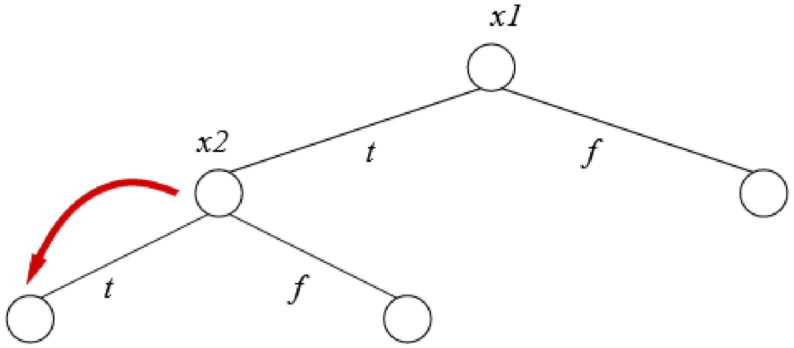
$\{ \cancel{\neg x_1} \vee x_3 \vee x_4, \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \neg x_3,$
$\neg x_4 \vee \neg x_2, x_2 \vee \neg x_3 \vee \cancel{\neg x_1}, x_2 \vee x_6 \vee x_3,$
$x_2 \vee \neg x_6 \vee \neg x_4, \cancel{x_1 \vee x_5}, \cancel{x_1 \vee x_6},$
$\neg x_6 \vee x_3 \vee \neg x_5, \cancel{x_1 \vee \neg x_3 \vee \neg x_5} \}$
$=$
$\{ x_3 \vee x_4, \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \neg x_3,$
$\neg x_4 \vee \neg x_2, x_2 \vee \neg x_3, x_2 \vee x_6 \vee x_3,$
$x_2 \vee \neg x_6 \vee \neg x_4,$
$\neg x_6 \vee x_3 \vee \neg x_5 \}$

x_5 only occurs negated

can be set to false, removing clause

$\{ x_3 \vee x_4, \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \neg x_3,$
$\neg x_4 \vee \neg x_2, x_2 \vee \neg x_3, x_2 \vee x_6 \vee x_3,$
$x_2 \vee \neg x_6 \vee \neg x_4 \}$

DPLL, complete example (4)



choose variable x_2, value *true* first

DPLL, complete example (5)

with $\{x_1=true,\ x_2=true\}$ the clauses become:

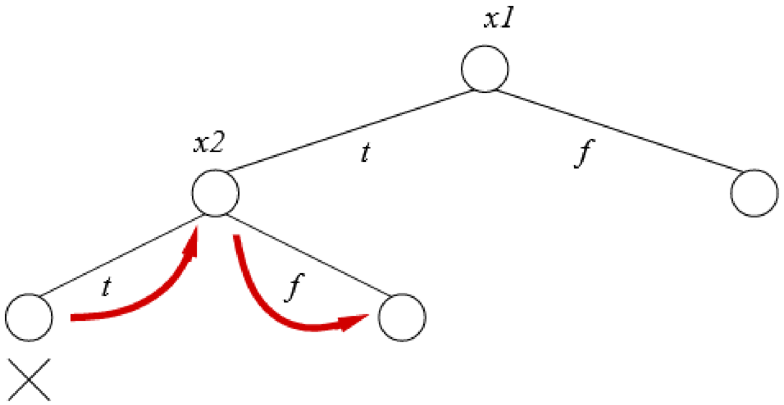
$\{\ x_3 \lor x_4,\ \cancel{\neg x_2} \lor x_6 \lor x_4,\ \cancel{\neg x_2} \lor \neg x_6 \lor \neg x_3,$
$\neg x_4 \lor \cancel{\neg x_2},\ \cancel{x_2 \lor \neg x_3},\ \cancel{x_2 \lor x_6 \lor x_3},$
$\cancel{x_2 \lor \neg x_6 \lor \neg x_4}\ \}$
$=$
$\{\ x_3 \lor x_4,\ x_6 \lor x_4,\ \neg x_6 \lor \neg x_3,$
$\neg x_4\ \}$

from $\neg x_4$ we derive x_3 and x_6

they falsify the clause $\neg x_6 \lor \neg x_3$

contradiction, no need to apply pure

DPLL, complete example (5)



contradiction reached, backtrack

DPLL, complete example (5)

with $\{x_1=true,\ x_2=false\}$, clauses become:

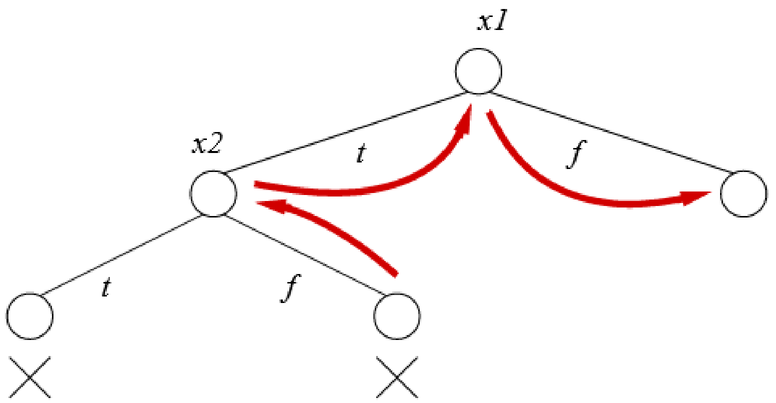
$\{\ x_3 \lor x_4,\ \cancel{\neg x_2 \lor x_6 \lor x_4},\ \cancel{\neg x_2 \lor \neg x_6 \lor \neg x_3},$
$\cancel{\neg x_4 \lor \neg x_2},\ \cancel{x_2} \lor \neg x_3,\ \cancel{x_2} \lor x_6 \lor x_3,$
$\cancel{x_2} \lor \neg x_6 \lor \neg x_4\ \}$
$=$
$\{\ x_3 \lor x_4,$
$\neg x_3,\ x_6 \lor x_3,$
$\neg x_6 \lor \neg x_4\ \}$

from $\neg x_3$ we derive x_4 and x_6

they contradict clause $\neg x_6 \lor \neg x_4$

contradiction, no need to apply pure

DPLL, complete example (6)



backtrack to first node, try other branch

DPLL, complete example (6)

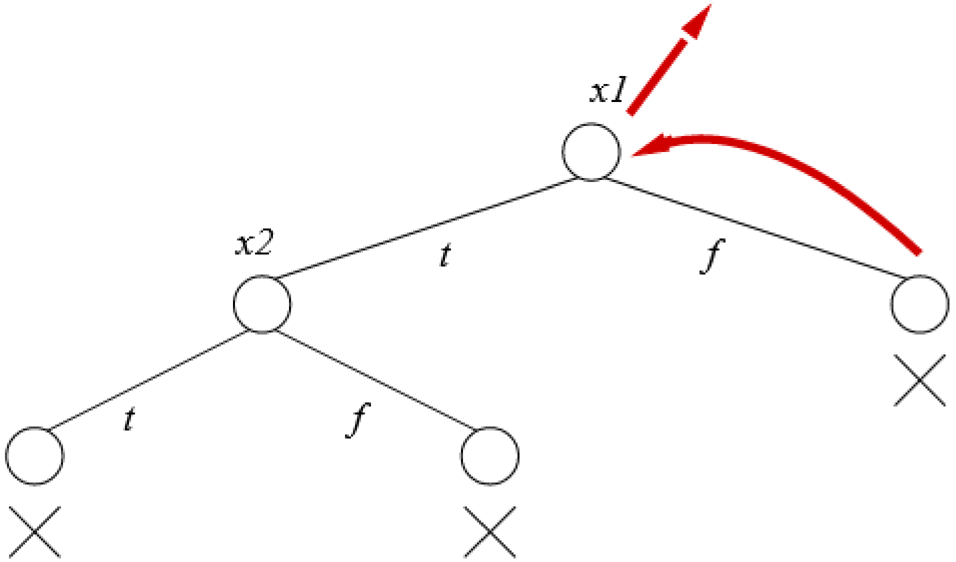
with $\{x_1 = false\}$ clauses become:

$\{ \cancel{\neg x_1 \vee x_3 \vee x_4}, \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \neg x_3,$
$\neg x_4 \vee \neg x_2, \cancel{x_2 \vee \neg x_3 \vee \neg x_1}, x_2 \vee x_6 \vee x_3,$
$x_2 \vee \neg x_6 \vee \neg x_4, \cancel{x_1} \vee x_5, \cancel{x_1} \vee x_6,$
$\neg x_6 \vee x_3 \vee \neg x_5, \cancel{x_1} \vee \neg x_3 \vee \neg x_5 \}$
$=$
$\{ \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \neg x_3,$
$\neg x_4 \vee \neg x_2, x_2 \vee x_6 \vee x_3,$
$x_2 \vee \neg x_6 \vee \neg x_4, x_5, x_6,$
$\neg x_6 \vee x_3 \vee \neg x_5, \neg x_3 \vee \neg x_5 \}$

from x_5 we derive $\neg x_3$

since x_6 is true, clause $\neg x_6 \vee x_3 \vee \neg x_5$ is falsified

DPLL, complete example (7)



contradiction reached on last node

set is unsatisfiable

Choice of branching variable

which is the best, among the unassigned ones?

does it make any difference?

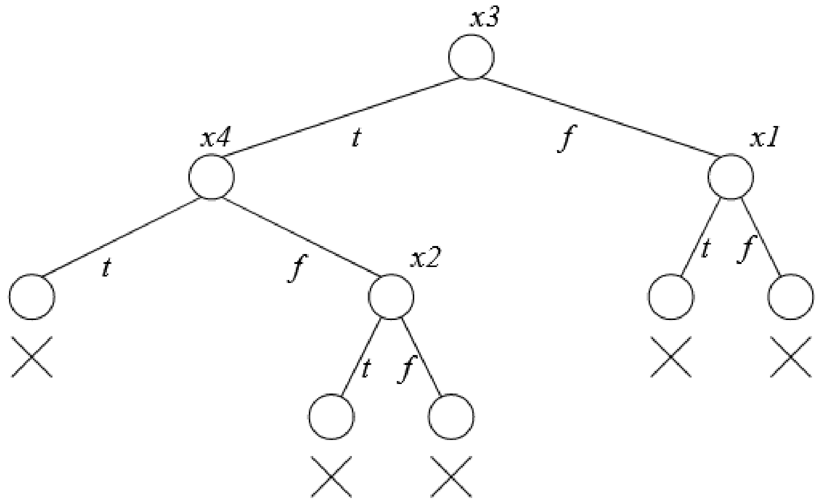
Same example, different choices

same set as previous example

$\{ \neg x_1 \vee x_3 \vee x_4, \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \neg x_3,$
$\neg x_4 \vee \neg x_2, x_2 \vee \neg x_3 \vee \neg x_1, x_2 \vee x_6 \vee x_3,$
$x_2 \vee \neg x_6 \vee \neg x_4, x_1 \vee x_5, x_1 \vee x_6,$
$\neg x_6 \vee x_3 \vee \neg x_5, x_1 \vee \neg x_3 \vee \neg x_5 \}$

choose x_3 first
then other variables

Same example, different choices



(execution details)

larger tree ⟶ longer running time

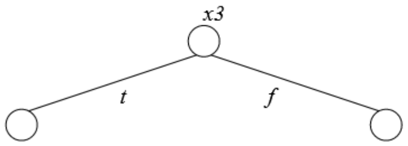
in general: difference may be exponential

Different choice of branching variables

set of clauses:

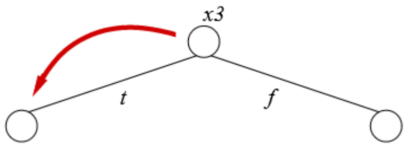
$\{ \neg x_1 \vee x_3 \vee x_4, \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \neg x_3, \neg x_4 \vee \neg x_2, x_2 \vee \neg x_3 \vee \neg x_1, x_2 \vee x_6 \vee x_3, x_2 \vee \neg x_6 \vee \neg x_4, x_1 \vee x_5, x_1 \vee x_6, \neg x_6 \vee x_3 \vee \neg x_5, x_1 \vee \neg x_3 \vee \neg x_5 \}$

First branching



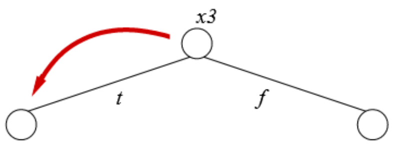
branch on x_3

First recursive call



recursive call with x_3=*true*

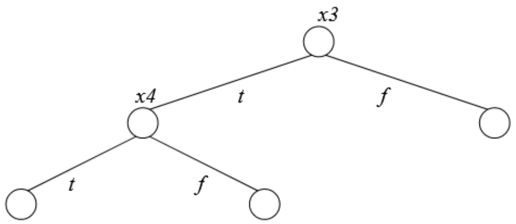
Propagate x_3=*true*



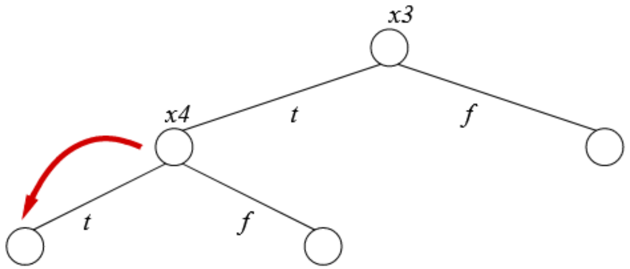
$\{ \cancel{\neg x_1 \vee x_3 \vee x_4}, \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6 \vee \cancel{\neg x_3}, \neg x_4 \vee \neg x_2, x_2 \vee \cancel{\neg x_3} \vee \neg x_1, \cancel{x_2 \vee x_6 \vee x_3}, x_2 \vee \neg x_6 \vee \neg x_4, x_1 \vee x_5, x_1 \vee x_6, \cancel{\neg x_6 \vee x_3 \vee \neg x_5}, x_1 \vee \cancel{\neg x_3} \vee \neg x_5 \}$
=
$\{ \neg x_2 \vee x_6 \vee x_4, \neg x_2 \vee \neg x_6, \neg x_4 \vee \neg x_2, x_2 \vee \neg x_1, x_2 \vee \neg x_6 \vee \neg x_4, x_1 \vee x_5, x_1 \vee x_6, x_1 \vee \neg x_5 \}$

no contradiction, choose a branching variable

Branch on x_4

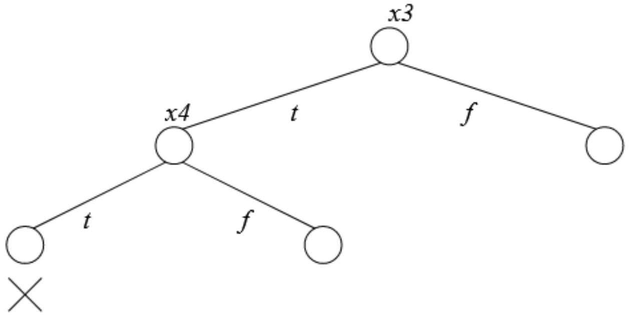


Propagate x_4=*true*

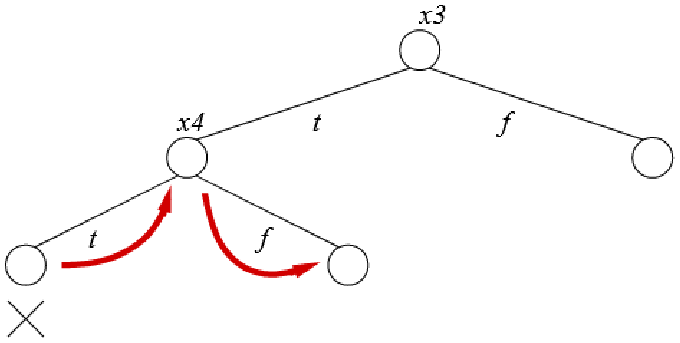


$\{ \cancel{\neg x_2 \vee x_6 \vee x_4}, \neg x_2 \vee \neg x_6, \cancel{\neg x_4} \vee \neg x_2, x_2 \vee \neg x_1, x_2 \vee \neg x_6 \vee \cancel{\neg x_4}, x_1 \vee x_5, x_1 \vee x_6, x_1 \vee \neg x_5 \}$
=
$\{ \neg x_2 \vee \neg x_6, \neg x_2, x_2 \vee \neg x_1, x_2 \vee \neg x_6, x_1 \vee x_5, x_1 \vee x_6, x_1 \vee \neg x_5 \}$
= (with $\neg x_2$)
$\{ \cancel{\neg x_2 \vee \neg x_6}, \cancel{\neg x_2}, \cancel{x_2} \vee \neg x_1, \cancel{x_2} \vee \neg x_6, x_1 \vee x_5, x_1 \vee x_6, x_1 \vee \neg x_5 \}$
=
$\{ \neg x_1, \neg x_6, x_1 \vee x_5, x_1 \vee x_6, x_1 \vee \neg x_5 \}$
=
contradiction: $\neg x_1, \neg x_6, x_1 \vee x_6$

Contradiction generated



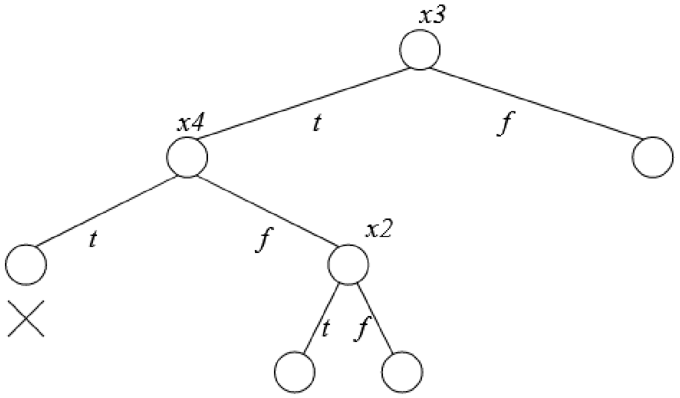
Backtrack, propagate x_4=*false*



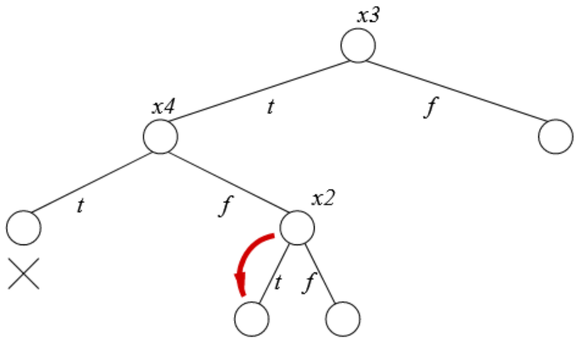
$\{ \neg x_2 \vee x_6 \vee \cancel{x_4},\ \neg x_2 \vee \neg x_6,\ \cancel{\neg x_4 \vee \neg x_2},\ x_2 \vee \neg x_1,\ \cancel{x_2 \vee \neg x_6 \vee \neg x_4},\ x_1 \vee x_5,\ x_1 \vee x_6,\ x_1 \vee \neg x_5 \}$
$=$
$\{ \neg x_2 \vee x_6,\ \neg x_2 \vee \neg x_6,\ x_2 \vee \neg x_1,\ x_1 \vee x_5,\ x_1 \vee x_6,\ x_1 \vee \neg x_5 \}$

no contradiction, no unit clause

Choice of another branching variable



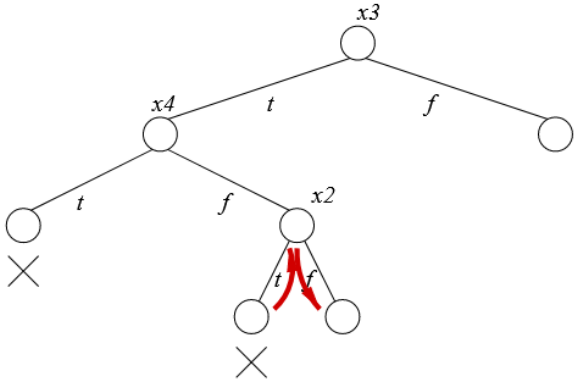
Set x_2=*true*



$\{ \cancel{\neg x_2} \vee x_6,\ \cancel{\neg x_2} \vee \neg x_6,\ \cancel{x_2 \vee \neg x_1},\ x_1 \vee x_5,\ x_1 \vee x_6,\ x_1 \vee \neg x_5 \}$
$=$
$\{ x_6,\ \neg x_6,\ x_1 \vee x_5,\ x_1 \vee x_6,\ x_1 \vee \neg x_5 \}$

contradiction

Backtrack, propagate x_2=*false*

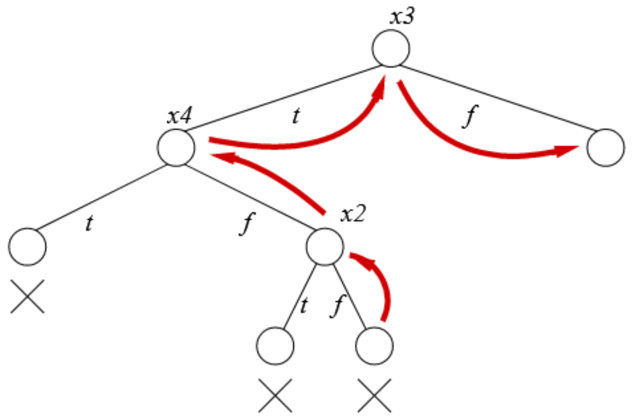


clauses-3T-4F-2F.txt

$\{ \cancel{\neg x_2 \vee x_6},\ \cancel{\neg x_2 \vee \neg x_6},\ \cancel{x_2} \vee \neg x_1,\ x_1 \vee x_5,\ x_1 \vee x_6,\ x_1 \vee \neg x_5 \}$
$=$
$\{ \neg x_1,\ x_1 \vee x_5,\ x_1 \vee x_6,\ x_1 \vee \neg x_5 \}$

contradictin: from $\neg x_1$ both x_5 and $\neg x_5$ follow

Backtrack, propagate x_3=*false*



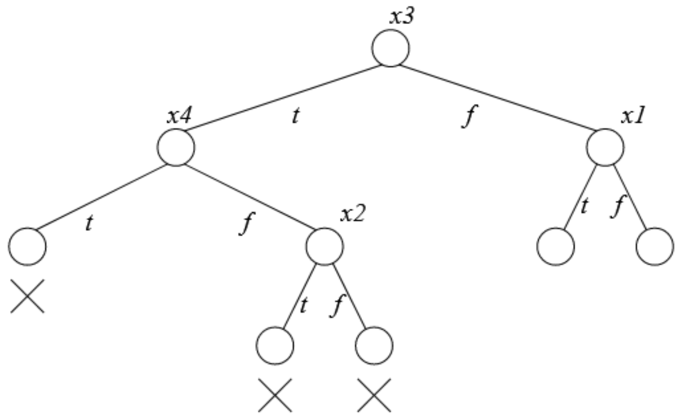
$\{ \neg x_1 \vee \cancel{x_3} \vee x_4,\ \neg x_2 \vee x_6 \vee x_4,\ \cancel{\neg x_2 \vee \neg x_6 \vee \neg x_3},\ \neg x_4 \vee \neg x_2,\ \cancel{x_2 \vee \neg x_3 \vee \neg x_1},\ x_2 \vee x_6 \vee \cancel{x_3},\ x_2 \vee \neg x_6 \vee \neg x_4,\ x_1 \vee x_5,\ x_1 \vee x_6,\ \neg x_6 \vee \cancel{x_3} \vee \neg x_5,\ \cancel{x_1 \vee \neg x_3 \vee \neg x_5} \}$

=

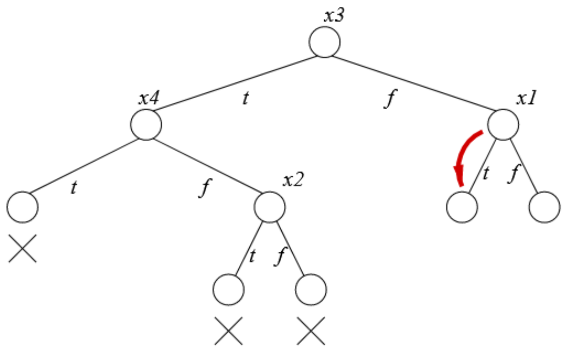
$\{ \neg x_1 \vee x_4,\ \neg x_2 \vee x_6 \vee x_4,\ \neg x_4 \vee \neg x_2,\ x_2 \vee x_6,\ x_2 \vee \neg x_6 \vee \neg x_4,\ x_1 \vee x_5,\ x_1 \vee x_6,\ \neg x_6 \vee \neg x_5 \}$

no contradiction, no unit clause

Choose another branching variable



Set x_1=*true*, propagate



$\{ \cancel{\neg x_1} \vee x_4,\ \neg x_2 \vee x_6 \vee x_4,\ \neg x_4 \vee \neg x_2,\ x_2 \vee x_6,\ x_2 \vee \neg x_6 \vee \neg x_4,\ \cancel{x_1 \vee x_5},\ \cancel{x_1 \vee x_6},\ \neg x_6 \vee \neg x_5 \}$

=

$\{ x_4,\ \neg x_2 \vee x_6 \vee x_4,\ \neg x_4 \vee \neg x_2,\ x_2 \vee x_6,\ x_2 \vee \neg x_6 \vee \neg x_4,\ \neg x_6 \vee \neg x_5 \}$

= (with x_4=*true*)

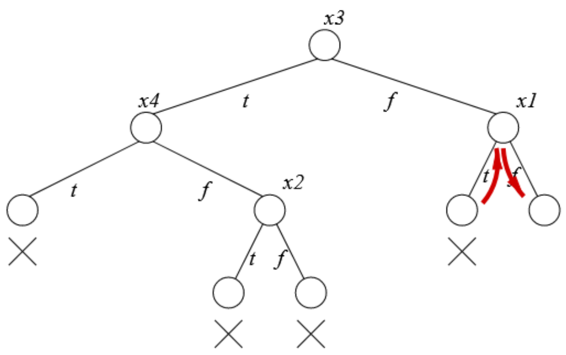
$\{ \cancel{x_4},\ \cancel{\neg x_2 \vee x_6 \vee x_4},\ \cancel{\neg x_4} \vee \neg x_2,\ x_2 \vee x_6,\ x_2 \vee \neg x_6 \vee \cancel{\neg x_4},\ \neg x_6 \vee \neg x_5 \}$

=

$\{ \neg x_2,\ x_2 \vee x_6,\ x_2 \vee \neg x_6,\ \neg x_6 \vee \neg x_5 \}$

contradiction: $\neg x_2$ generates both x_6 and $\neg x_6$

Backtrack, propagate x_1=*false*



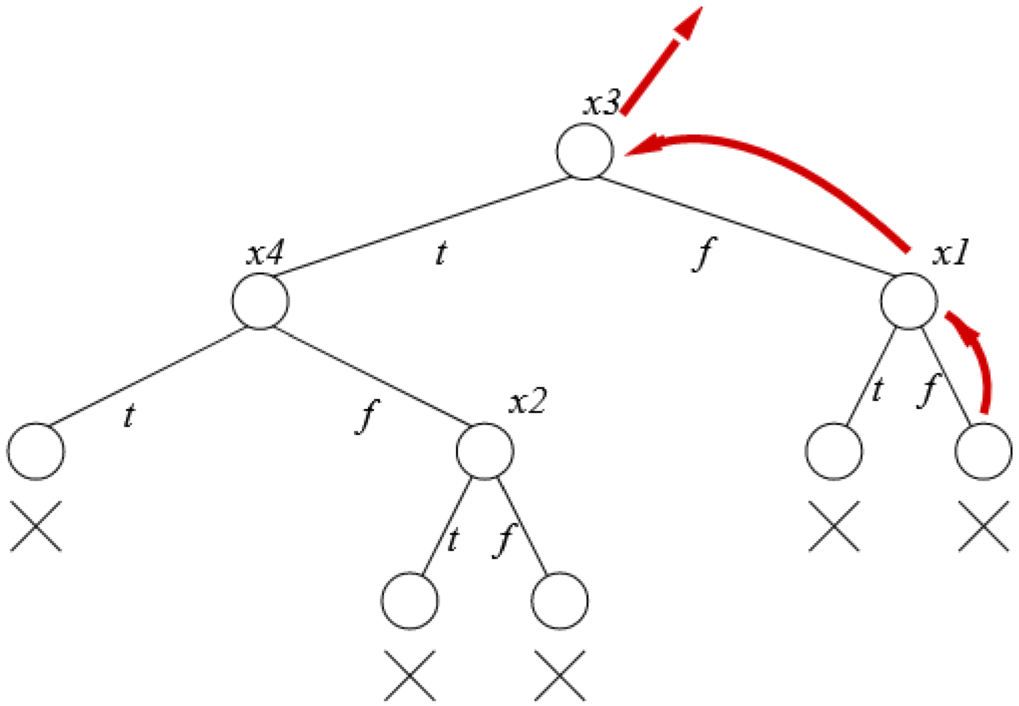
$\{ \cancel{\neg x_1 \vee x_4},\ \neg x_2 \vee x_6 \vee x_4,\ \neg x_4 \vee \neg x_2,\ x_2 \vee x_6,\ x_2 \vee \neg x_6 \vee \neg x_4,\ \cancel{x_1} \vee x_5,\ \cancel{x_1} \vee x_6,\ \neg x_6 \vee \neg x_5 \}$

=

$\{ \neg x_2 \vee x_6 \vee x_4,\ \neg x_4 \vee \neg x_2,\ x_2 \vee x_6,\ x_2 \vee \neg x_6 \vee \neg x_4,\ x_5,\ x_6,\ \neg x_6 \vee \neg x_5 \}$

contradiction: $x_5,\ x_6,\ \neg x_6 \vee \neg x_5$

Backtrack



set is unsatisfiable

Choice of branching variable: principle

try to reduce the number of the subsequent recursive calls in sat(F, I ∪ {x_i=true}) and sat(F, I ∪ {x_i=false})

Heuristics based on binary clauses

many binary clauses containing $\neg x_i$ = many assignments obtained by unit propagation in sat(F, I ∪ {x_i=true})

same for x_i and sat(F, I ∪ {x_i=false})

choose x_i that is contained in many binary clauses

- heuristics based on the *first step only* of unit propagation, but…
- many unit propagations are likely to lead to many ones more

---

Sign of variable

x_3 positive in 10 binary clauses and negative in none

x_8 positive in 4 binary clauses and negative in 4

how large the two subtrees are?

Evaluation of trees

assume that no further propagation is done after first step

- evaluation is qualitative
  (impossible to foresee the actual size of subtrees without specifying the whole formula)
- likely that many propagations in first step lead to many in further steps

Assignments in first step of unit propagation

x_3 (positive in 10, negative in none)

=true: zero
=false: 10

x_8 (positive in 4, negative in 4)

=true: 4
=false: 4

cost is exponential in the number of variables
assume 15 total

x_3

cost= $2^{15-1-10}+2^{15-1}= 2^4+2^{14}= 8+16384= 16394$

x_8

cost= $2^{15-1-4}+2^{15-1-4}= 2^{10}+2^{10}= 1024+1024= 2048$

better savings obtained by variables where positive and negative occurrences in binary clauses are balanced

---

A possible choice

old method based on an heuristics

for each variable x_i

- p_i is the number of binary clauses containing x_i
- n_i is the number of binary clauses containing $\neg x_i$

choose variable x_i that maximizes $1024p_1n_i+p_i+n_i$

idea: variables that have some positive *and* negative occurrences are preferred over some having many positive but few negative (or vice versa)

Finding the model

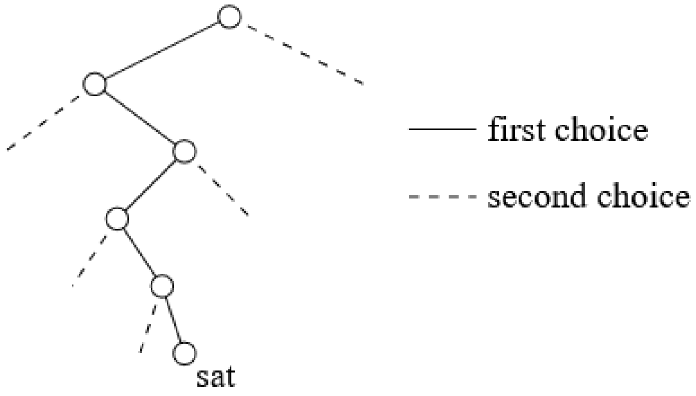
what if the formula is satisfiable?

different way of choosing the branching variable?

a first (wrong) principle: concentrate on choosing between x_i and $\neg x_i$

(wrong) principle: try to guess the sign of x_i in a model

Satisfiability and partial unsatisfiability

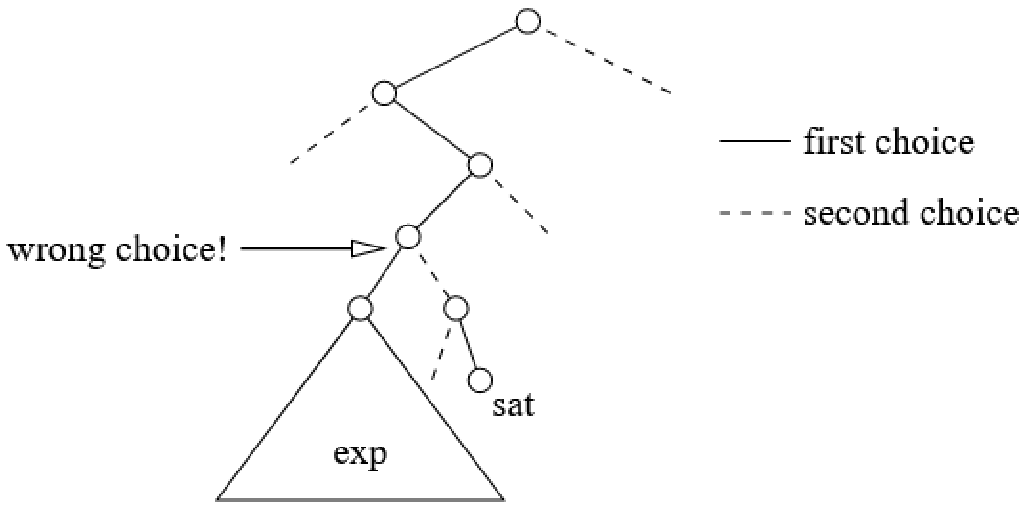


all choices correct: model found in linear time

impossible to make **all** choices right

what if one is wrong?

One wrong choice



wrong choice=no model in the subtree

formula unsatisfiable with partial model

unsatisfiability: search tree may be exponential

therefore: most of the time spent on the unsatisfiable subformula

even if formula is satisfiable, the hard part of the problem is still dealing with unsatisfiable formulae