Propositional Logic

Alcino Cunha

Overview

Terminology

- **Propositions** can be either true or false
- **Propositional variables** are identifiers that represent propositions
- Propositional variables are the **atomic formulas** of propositional logic
- **Compound formulas** can be formed using **logical connectives**

Syntax

- Propositional variables
  - $A, B, C, \dots$
- Logical connectives
  - $\top$ (true), $\bot$ (false), $\neg$(not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies), $\leftrightarrow$ (equivalent)
- Auxiliary symbols
  - Parenthesis

Syntax

$$A, B, C, \ldots \in \mathscr{V}$$
$$\phi, \psi, \ldots \in \mathbf{Form}_{\mathscr{V}}$$

$$\begin{aligned}
\phi, \psi \doteq\ & A, B, C, \ldots \\
\mid\ & \top \\
\mid\ & \bot \\
\mid\ & (\neg \phi) \\
\mid\ & (\phi \wedge \psi) \\
\mid\ & (\phi \vee \psi) \\
\mid\ & (\phi \rightarrow \psi) \\
\mid\ & (\phi \leftrightarrow \psi)
\end{aligned}$$

Semantics

- Given an **assignment** $\mathscr{A} : \mathscr{V} \rightarrow \{0,1\}$

- We can extend it to $\mathscr{A} : \mathbf{Form}_{\mathscr{V}} \rightarrow \{0,1\}$ to compute the truth value of a formula

- If $\mathscr{A}(\phi) = 1$ we say that ϕ **holds** under $\mathscr{A}$, denoted by $\mathscr{A} \models \phi$

- If $\mathscr{A}(\phi) = 0$ we say that ϕ **does not hold** under $\mathscr{A}$, denoted by $\mathscr{A} \not\models \phi$

Truth table semantics

$$\frac{\mathscr{A}(\top)}{1} \qquad \frac{\mathscr{A}(\bot)}{0}$$

$\mathscr{A}(\phi)$	$\mathscr{A}(\neg\phi)$
0	1
1	0

$\mathscr{A}(\phi)$	$\mathscr{A}(\psi)$	$\mathscr{A}(\phi \wedge \psi)$	$\mathscr{A}(\phi \vee \psi)$	$\mathscr{A}(\phi \to \psi)$	$\mathscr{A}(\phi \leftrightarrow \psi)$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Inductive semantics

$$
\begin{array}{rcl}
\mathscr{A} \models \top & & \\
\mathscr{A} \not\models \bot & & \\
\mathscr{A} \models p & \text{iff} & \mathscr{A}(p) = 1 \\
\mathscr{A} \models \neg\phi & \text{iff} & \mathscr{A} \not\models \phi \\
\mathscr{A} \models \phi \land \psi & \text{iff} & \mathscr{A} \models \phi \text{ and } \mathscr{A} \models \psi \\
\mathscr{A} \models \phi \lor \psi & \text{iff} & \mathscr{A} \models \phi \text{ or } \mathscr{A} \models \psi \\
\mathscr{A} \models \phi \to \psi & \text{iff} & \mathscr{A} \not\models \phi \text{ or } \mathscr{A} \models \psi \\
\mathscr{A} \models \phi \leftrightarrow \psi & \text{iff} & \mathscr{A} \models \phi \text{ iff } \mathscr{A} \models \psi
\end{array}
$$

More terminology

- A formula ϕ is
  - **valid** or a **tautology** iff it holds under all assignments
  - **satisfiable** iff it holds under some assignment
  - **unsatisfiable** or a **contradiction** iff it does not hold under all assignments
  - **refutable** iff it does not hold under some assignment
- A formula ϕ is valid iff $\neg\phi$ is unsatisfiable

Examples

- $B \vee (A \rightarrow \neg B)$ is valid

A	B	$\neg B$	$A \rightarrow \neg B$	$B \vee (A \rightarrow \neg B)$
0	0	1	1	1
0	1	0	1	1
1	0	1	1	1
1	1	0	0	1

- $A \rightarrow (\neg A \vee B)$ is both satisfiable and refutable

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow (\neg A \vee B)$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	0	1	1

Decidability

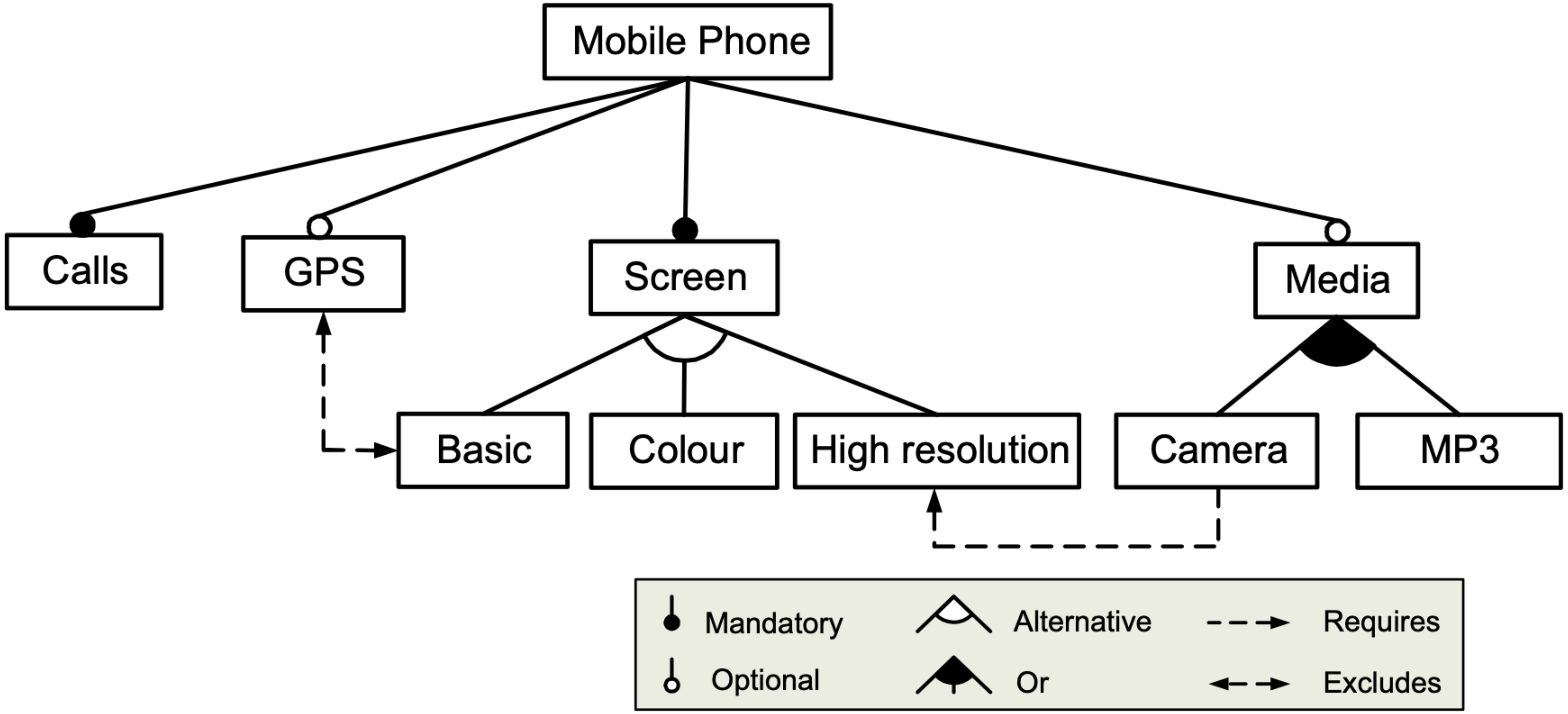
- A decision problem is **decidable** if there exists a mechanical method (aka an algorithm) for determining if it is either true or false
- The decision problem “Is ϕ satisfiable?”, also known as the **Boolean satisfiability problem** or **SAT**, is decidable
  - A naïve approach is to enumerate all possible assignments and check if ϕ holds for some of them using the truth table method
  - Later we will see that many **SAT solvers** implement more sophisticated methods...
- Hence the decision problem “Is ϕ valid?” is also decidable

Feature model analysis

Variability modelling

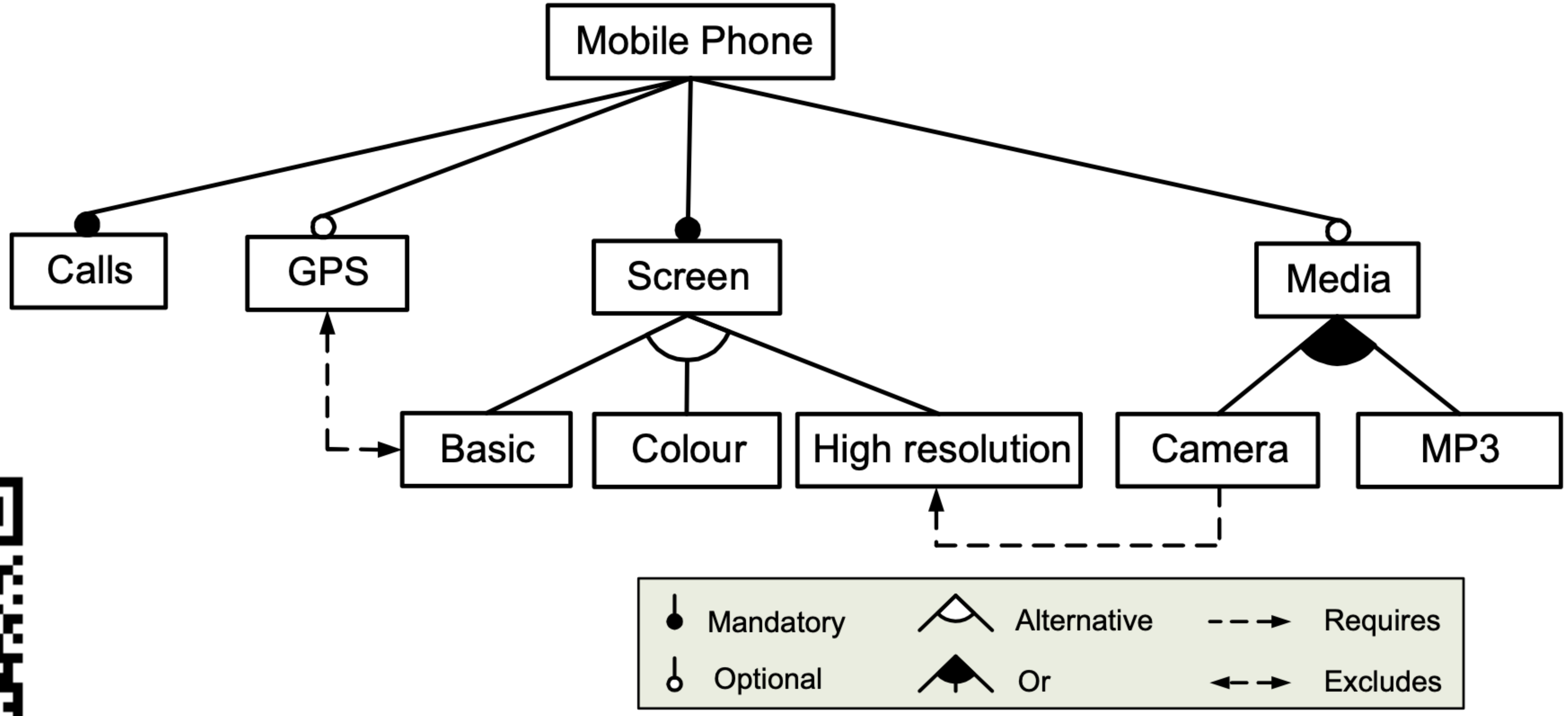
- A **software product line** (SPL) is a family of software products
- Each **product** or **variant** supports different features
- A **feature** is an increment in program functionality
- A **feature model** is a compact representation of the variability of a SPL

Feature models



https://doi.org/10.1016/j.is.2010.01.001

How many phone variants exist?



Feature model analysis

- Relevant analyses of a feature model
  - check if it is not void (there are products)
  - check if a feature is “dead” (no product can implement it)
  - check if a feature is “core” (all products implement it)
  - count how many different products exist
- All these analyses can be easily implemented using a SAT solver

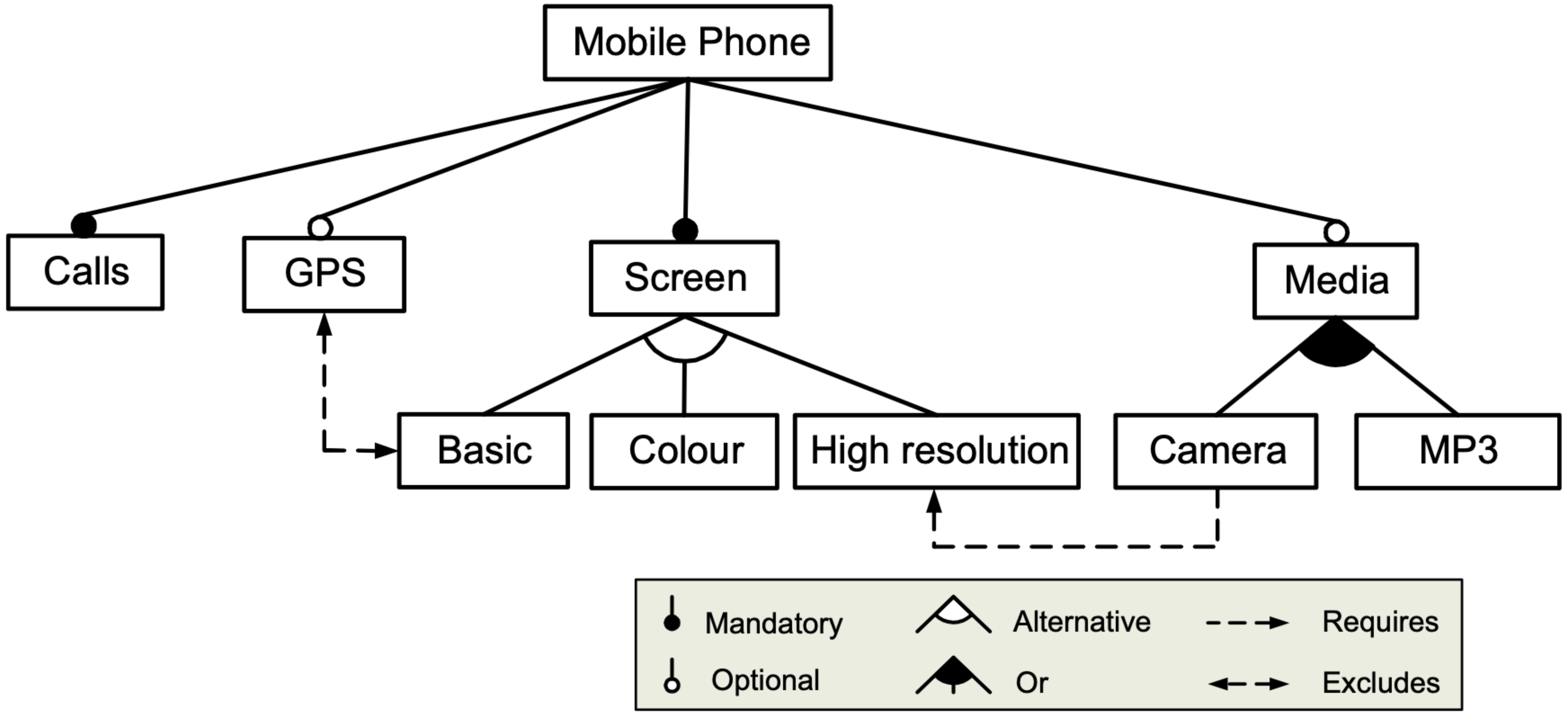
Feature model semantics

- A feature model can be encoded with a propositional formula ϕ
- Each feature corresponds to a propositional variable
- Each feature model primitive corresponds to a conjunct of ϕ

Feature model semantics

r is the root feature	r
f_1 mandatory sub-feature of f	$f_1 \leftrightarrow f$
f_1 optional sub-feature of f	$f_1 \rightarrow f$
$f_1, \ldots, f_n$ or sub-features of f	$f_1 \vee \ldots \vee f_n \leftrightarrow f$
$f_1, \ldots, f_n$ xor sub-features of f	$(f_1 \vee \ldots \vee f_n \leftrightarrow f) \wedge \bigwedge_{i<j} \neg (f_i \wedge f_j)$
f_1 requires f_2	$f_1 \rightarrow f_2$
f_1 excludes f_2	$\neg (f_1 \wedge f_2)$

Feature model semantics



Phone

Calls $\leftrightarrow$ Phone

GPS $\rightarrow$ Phone

Screen $\leftrightarrow$ Phone

Media $\rightarrow$ Phone

(Basic $\vee$ Color $\vee$ Highres) $\leftrightarrow$ Screen

$\neg$(Basic $\wedge$ Color) $\wedge$ $\neg$(Basic $\wedge$ Highres) $\wedge$ $\neg$(Color $\wedge$ Highres)

(Camera $\vee$ MP3) $\leftrightarrow$ Media

$\neg$(GPS $\wedge$ Basic)

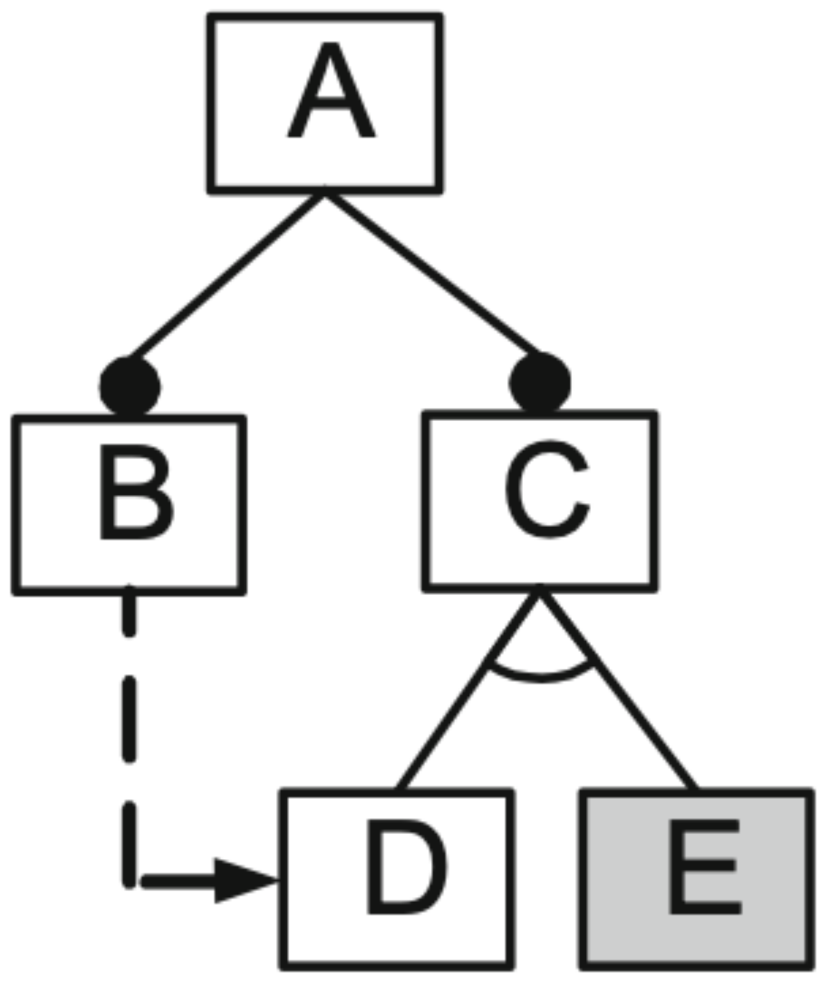
Camera $\rightarrow$ Highres

Non voidness

- Given a formula ϕ that encodes the semantics of a feature model
- To check if the feature model is not void
  - check if ϕ is satisfiable

Dead feature

- Given a formula ϕ that encodes the semantics of a feature model
- To check if feature f is dead
  - check if $\phi \wedge f$ is unsatisfiable



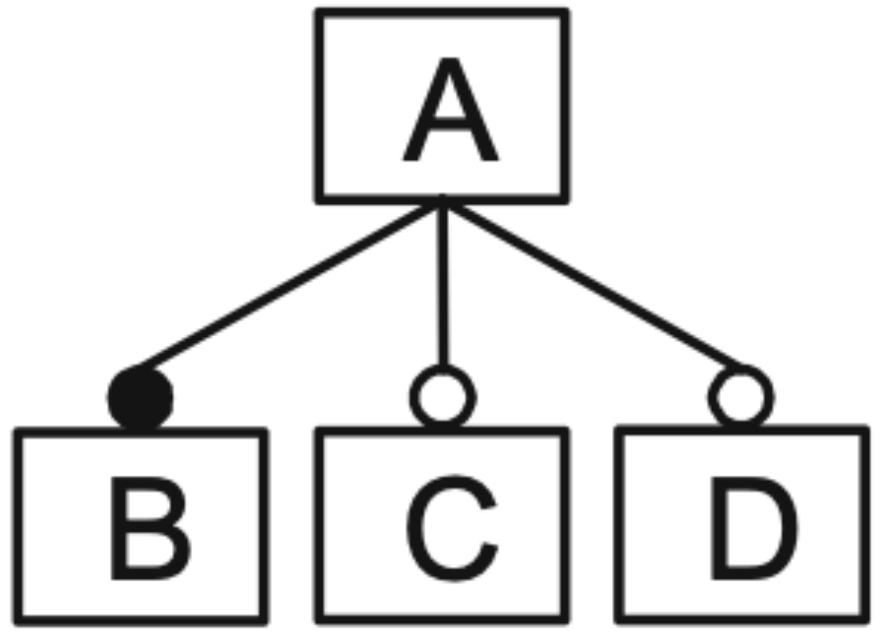
Core feature

- Given a formula ϕ that encodes the semantics of a feature model
- To check if feature f is core
  - check if $\phi \wedge \neg f$ is unsatisfiable

Counting products

- Given a formula ϕ that encodes the semantics of a feature model
- To count the number of products count the number of iterations of the following cycle
  - Repeat while ϕ is satisfiable
    - extract an assignment $\mathscr{A}$ under which ϕ holds
    - convert $\mathscr{A}$ to a formula $\overline{\mathscr{A}}$ that holds only for $\mathscr{A}$
    - add $\neg\overline{\mathscr{A}}$ as a new conjunct of ϕ

Counting products



$$\phi_0 \equiv A \wedge (A \leftrightarrow B) \wedge (C \rightarrow A) \wedge (D \rightarrow A)$$
$$\mathscr{A}_1 \equiv \{A \mapsto 1, B \mapsto 1, C \mapsto 0, D \mapsto 0\}$$
$$\phi_1 \equiv A \wedge (A \leftrightarrow B) \wedge (C \rightarrow A) \wedge (D \rightarrow A) \wedge \neg(A \wedge B \wedge \neg C \wedge \neg D)$$
$$\mathscr{A}_2 \equiv \{A \mapsto 1, B \mapsto 1, C \mapsto 1, D \mapsto 0\}$$
$$\phi_2 \equiv A \wedge (A \leftrightarrow B) \wedge (C \rightarrow A) \wedge (D \rightarrow A) \wedge \neg(A \wedge B \wedge \neg C \wedge \neg D) \wedge \neg(A \wedge B \wedge C \wedge \neg D)$$
$$\mathscr{A}_3 \equiv \{A \mapsto 1, B \mapsto 1, C \mapsto 0, D \mapsto 1\}$$
$$\phi_3 \equiv \ldots$$
$$\mathscr{A}_4 \equiv \{A \mapsto 1, B \mapsto 1, C \mapsto 1, D \mapsto 1\}$$
$$\phi_4 \equiv \ldots$$

Let’s do it with Z3!

DEMO

SAT solving

Complexity

- Given a propositional formula ϕ, the decision problem "Is ϕ satisfiable?" is known as the **Boolean satisfiability problem** or **SAT**
  - SAT is decidable
  - SAT is NP-complete

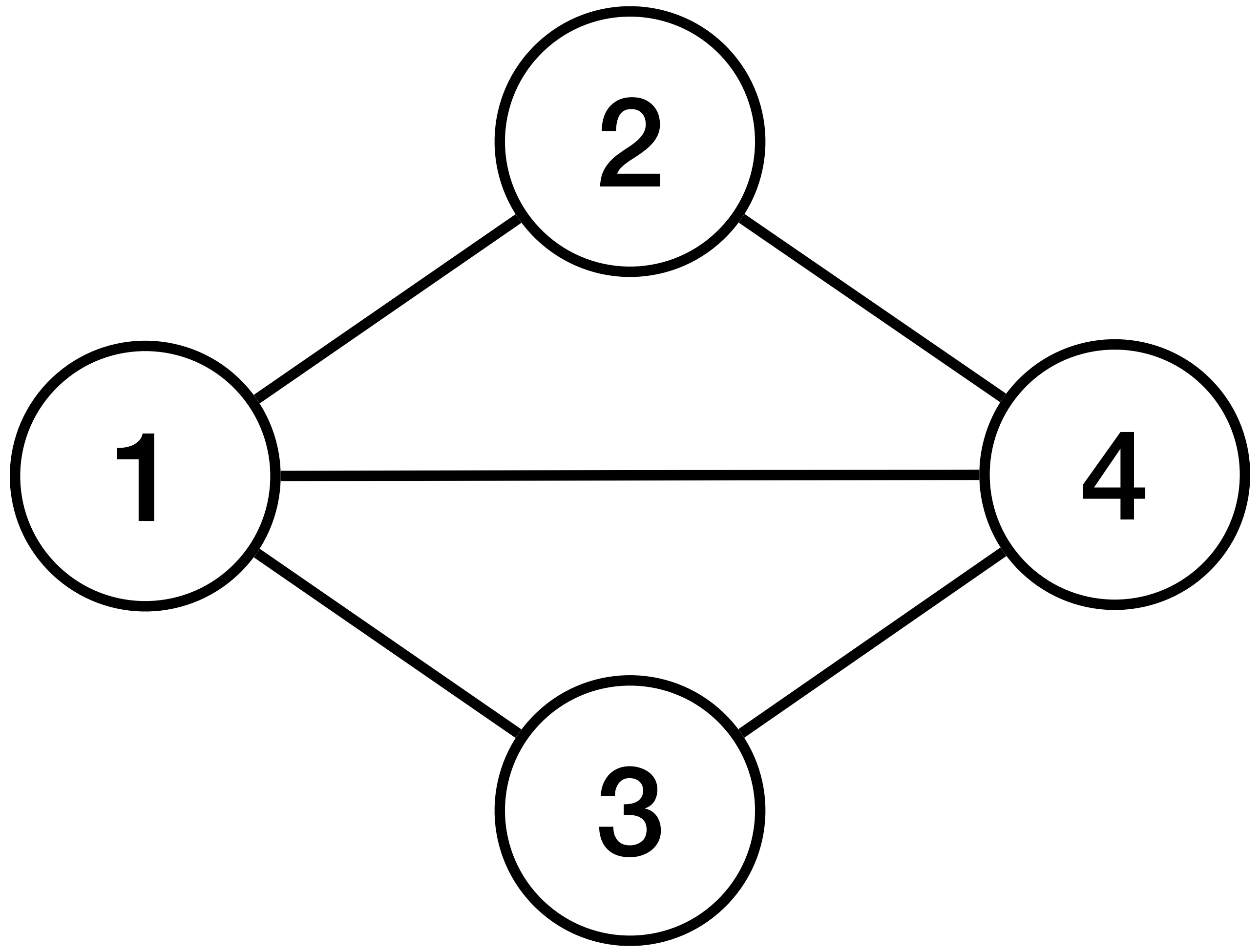
NP-completeness

- **Nondeterministic polynomial** time (NP) is a complexity class for decision problems
  - Problem instances have “proofs” verifiable in polynomial time
  - SAT is in NP (proofs are assignments)
- A NP problem is **NP-complete** if every problem in NP is reducible to it in polynomial time
  - SAT was the first problem to be show to be NP-complete
  - $P \subseteq NP$ but we do not known if $P = NP$ or $P \neq NP$

Graph colouring

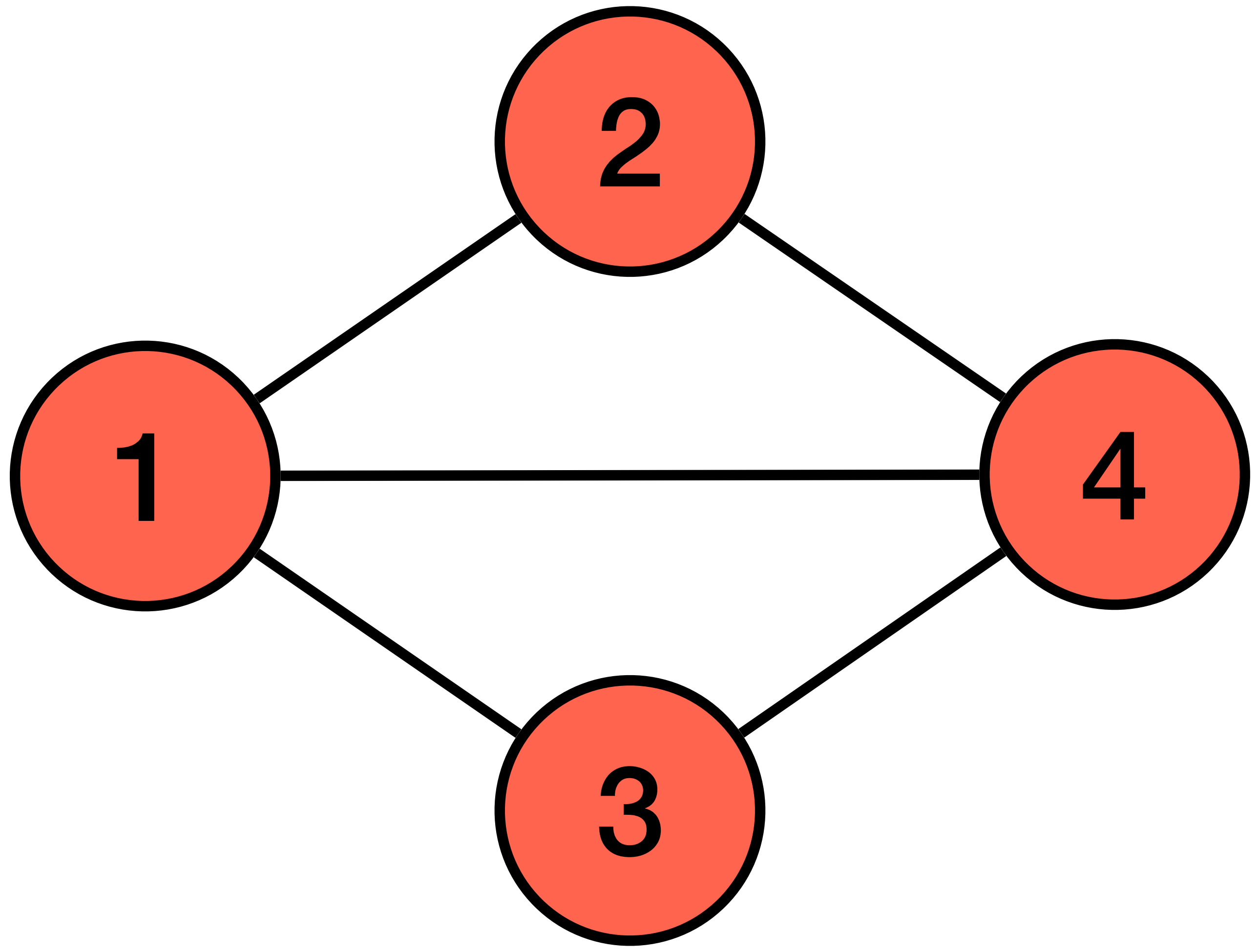
- The decision problem “Can an undirected graph be coloured with k colours”? is also NP-complete

- Can the following graph be coloured with 3 colours?



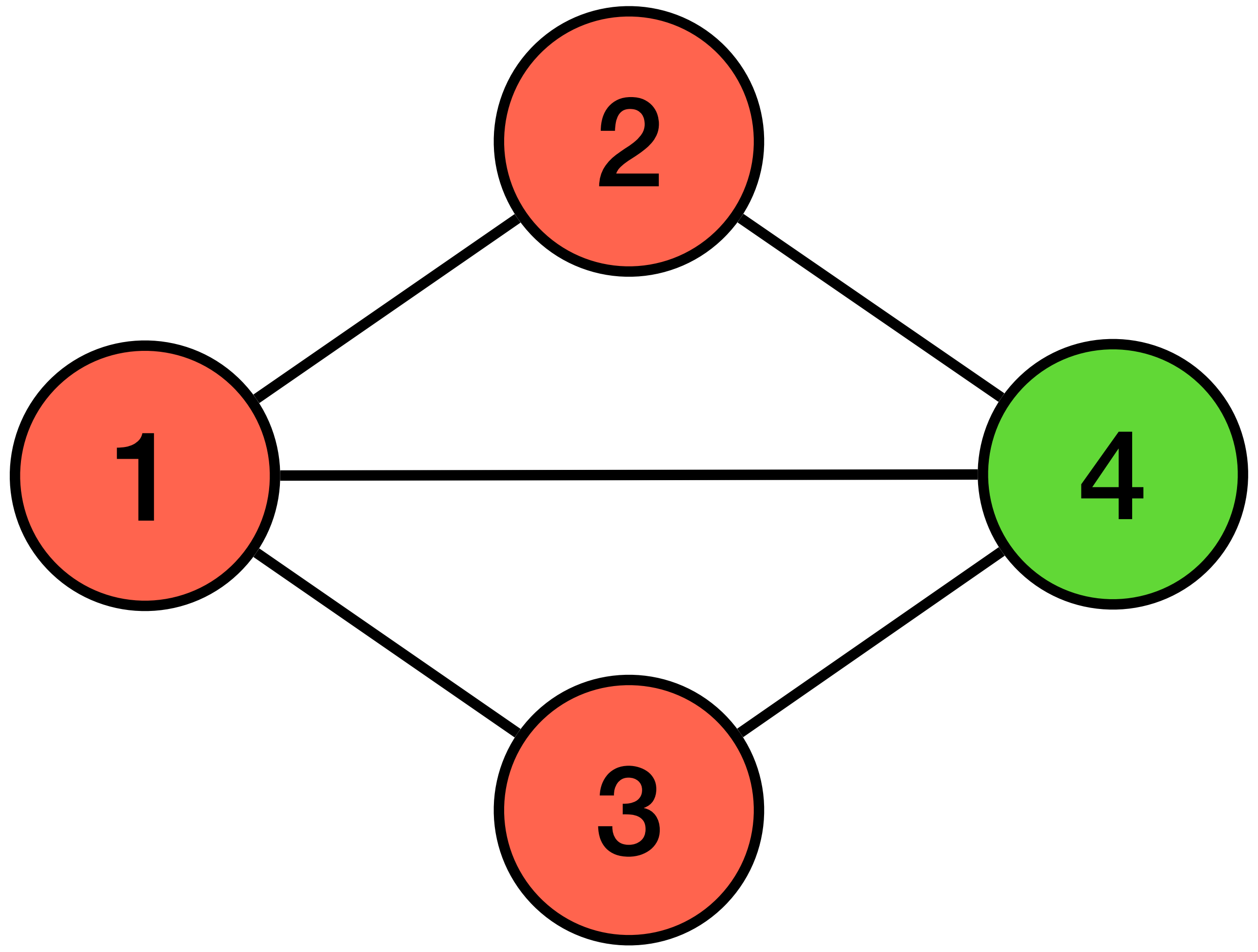
Brute force

1	2	3	4	
R	R	R	R	✗



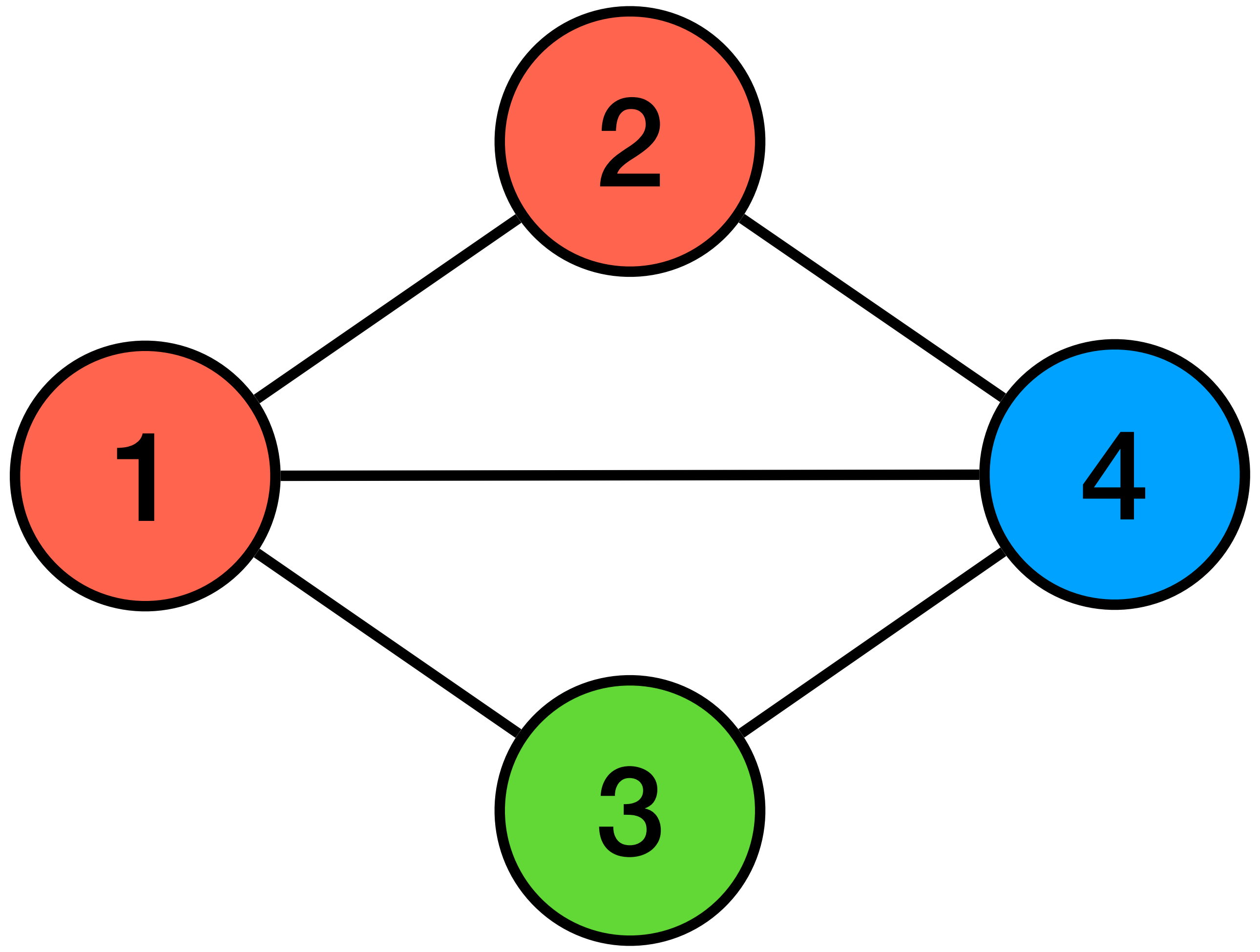
Brute force

1	2	3	4	
R	R	R	R	✗
R	R	R	G	✗



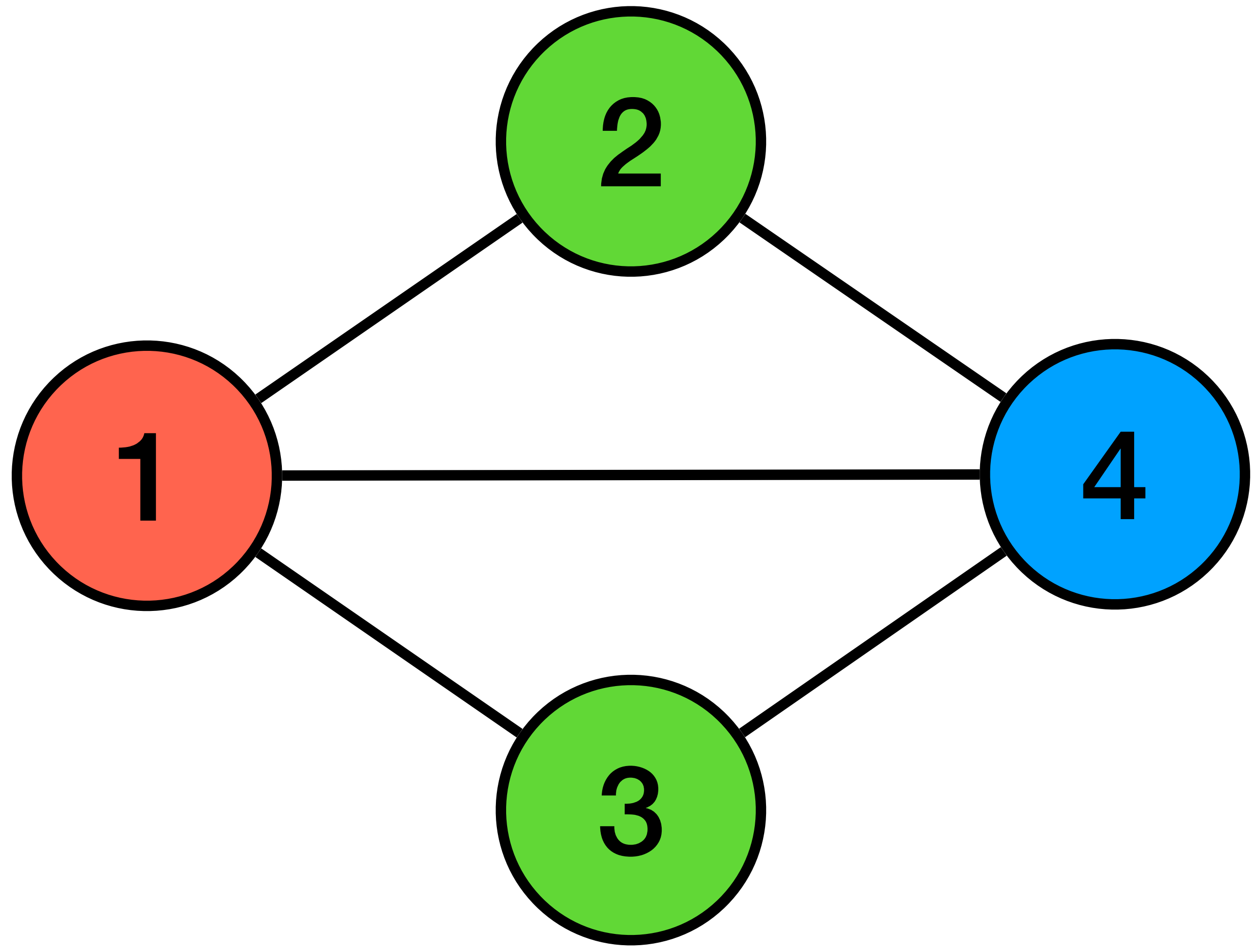
Brute force

1	2	3	4	
R	R	R	R	✗
R	R	R	G	✗
R	R	R	B	✗
R	R	G	R	✗
R	R	G	G	✗
R	R	G	B	✗



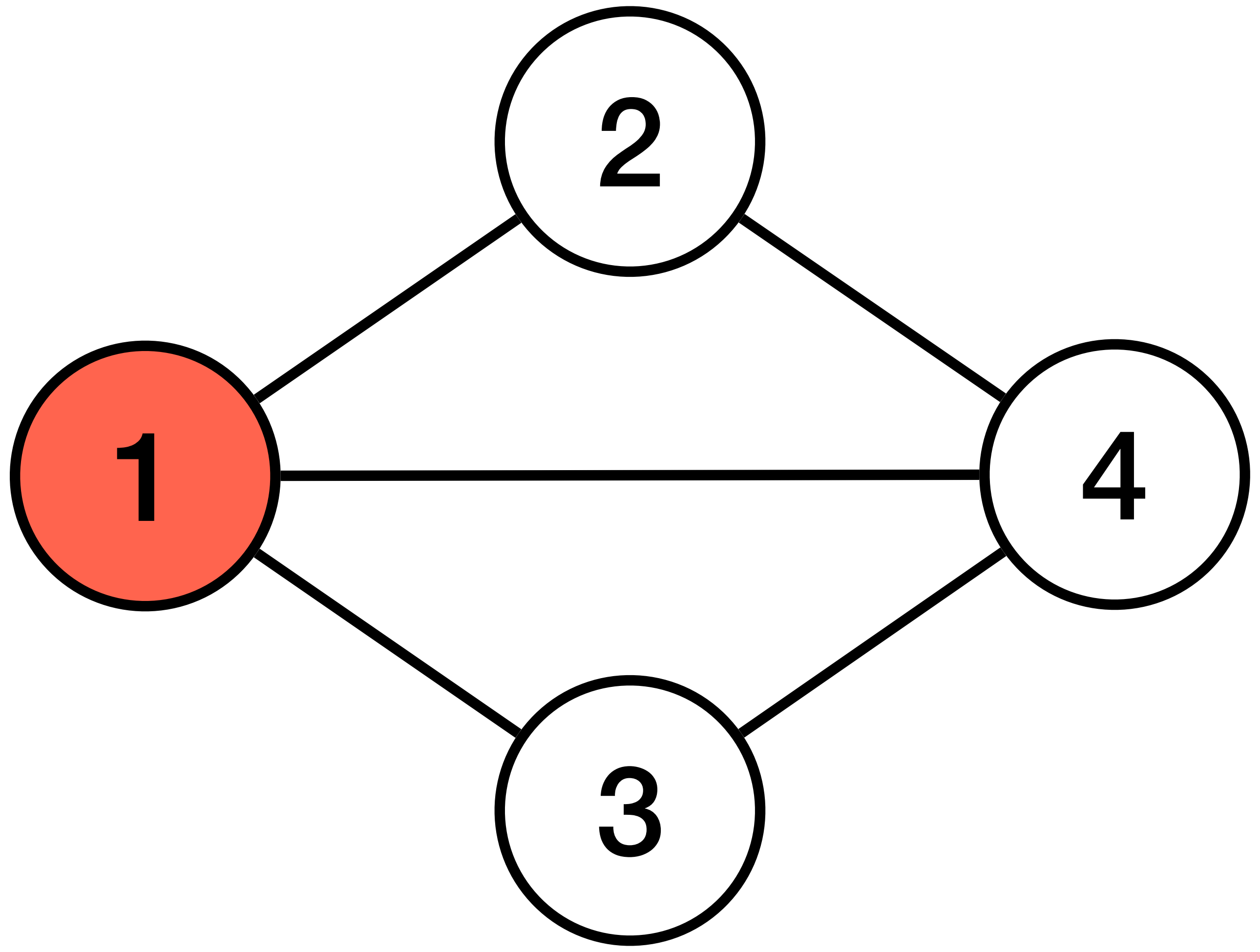
Brute force

1	2	3	4	
R	R	R	R	✗
R	R	R	G	✗
R	R	R	B	✗
R	R	G	R	✗
R	R	G	G	✗
R	R	G	B	✗
R	R	B	R	✗
		...		
R	G	G	B	✓



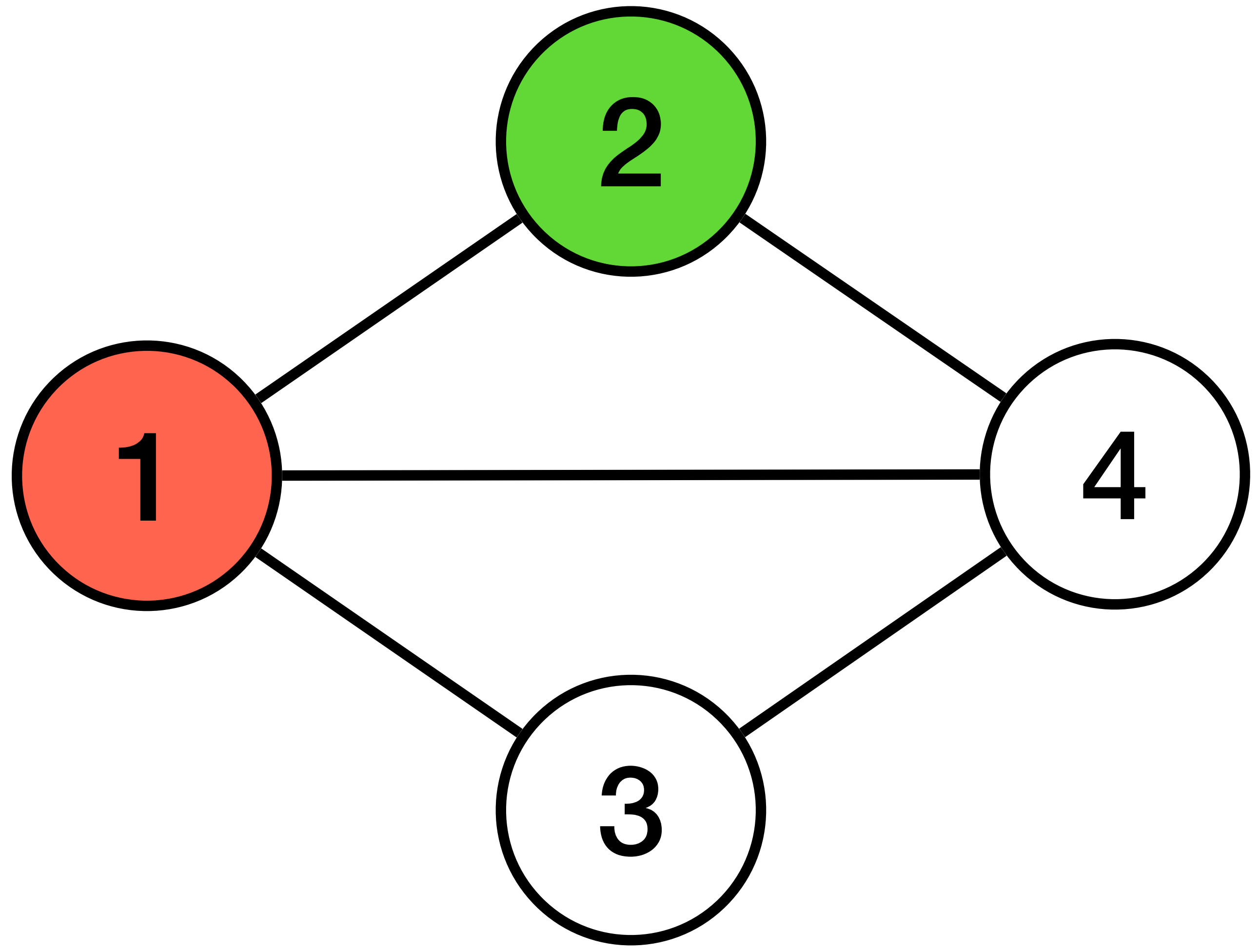
Backtracking

1	2	3	4	
R				



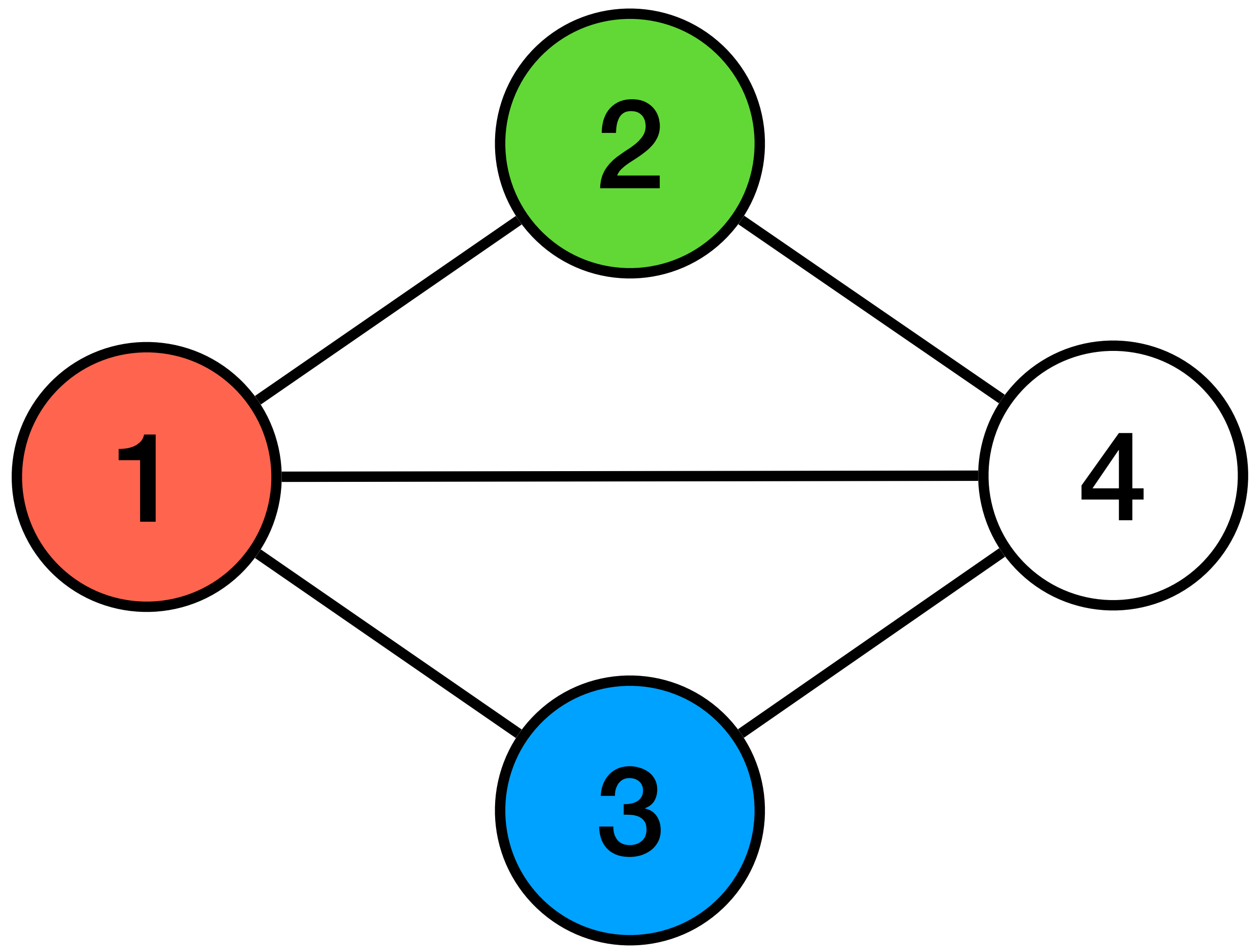
Backtracking

1	2	3	4	
R				
R	G			



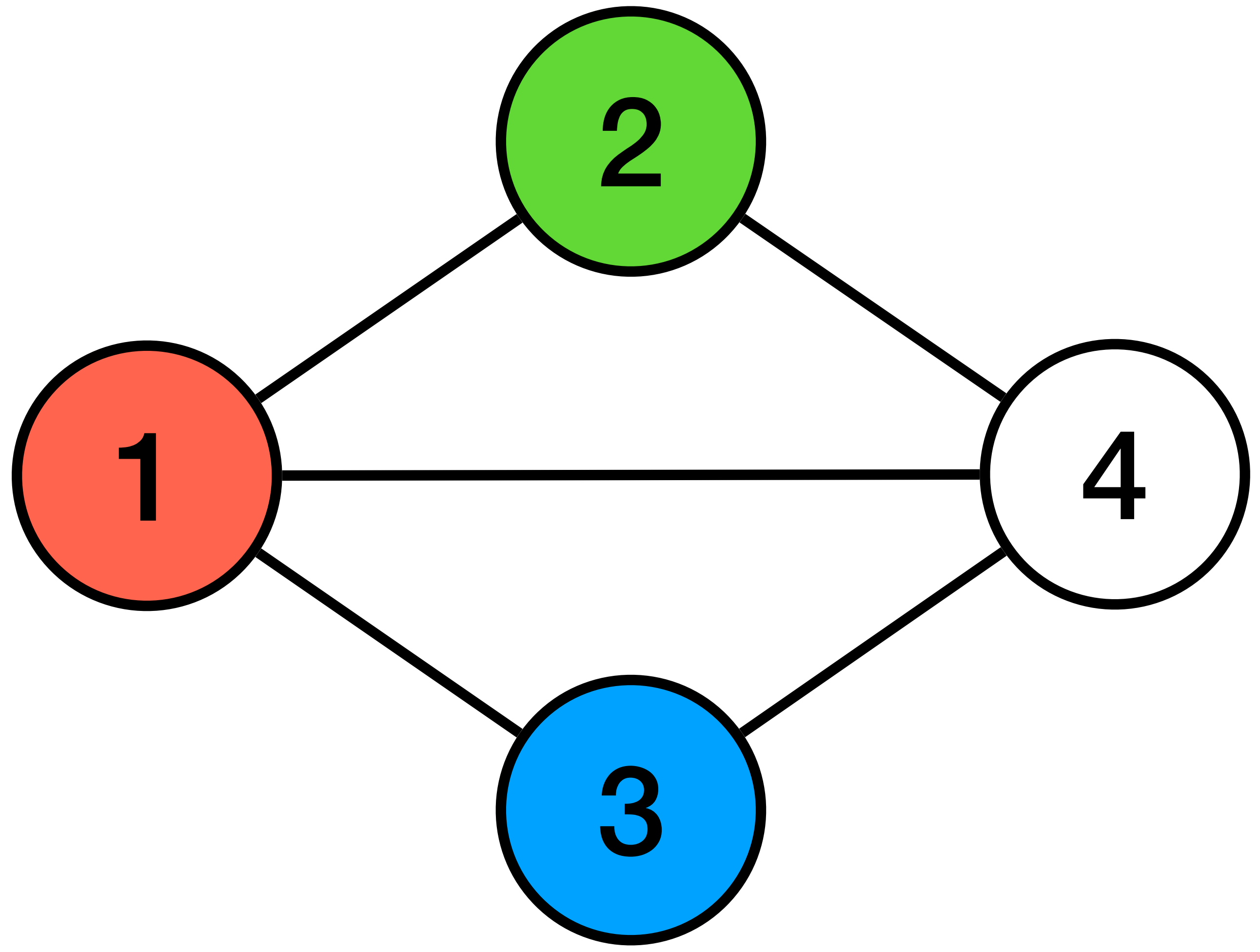
Backtracking

1	2	3	4	
R				
R	G			
R	G	B		



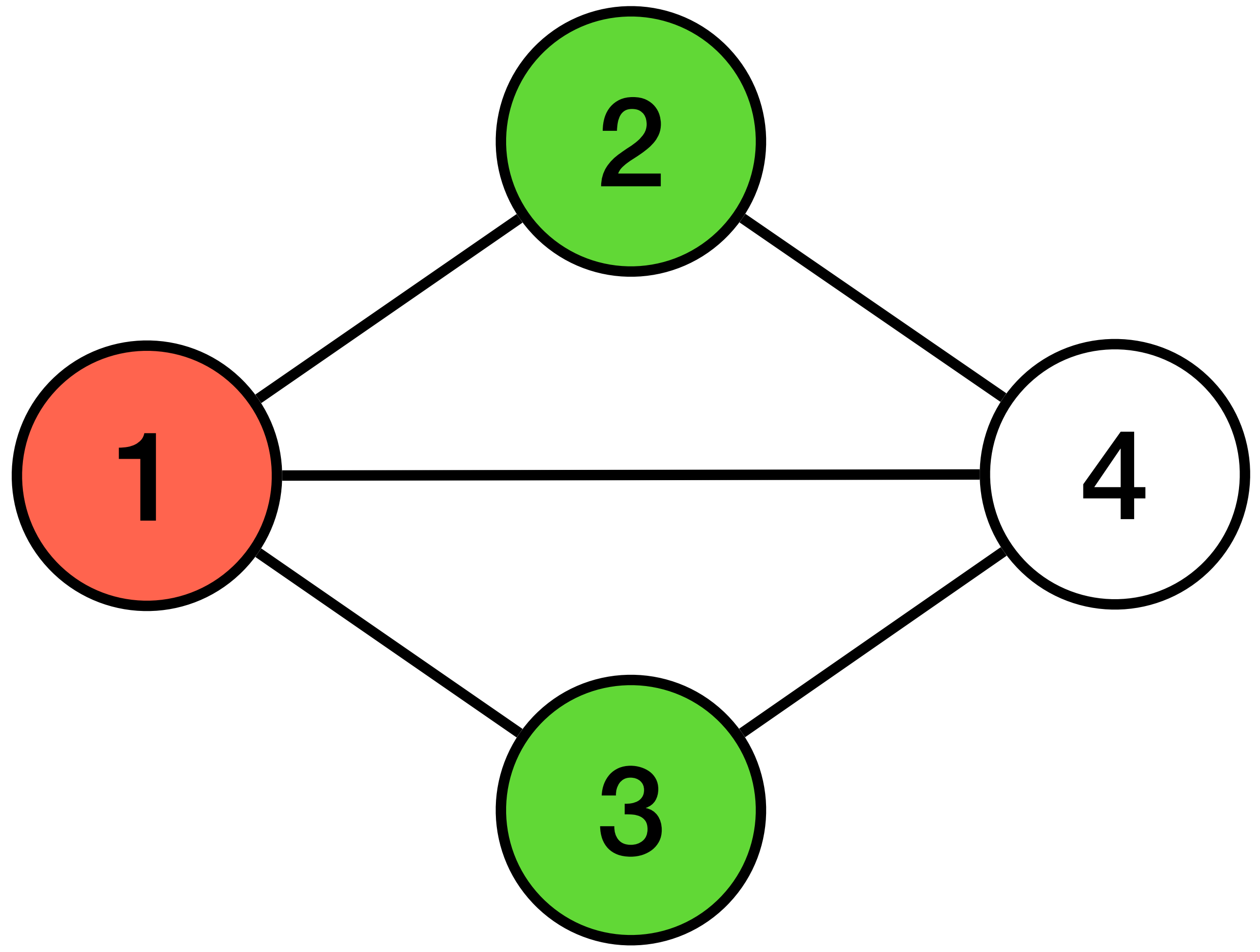
Backtracking

1	2	3	4	
R				
R	G			
R	G	B		
R	G	B	?	✗



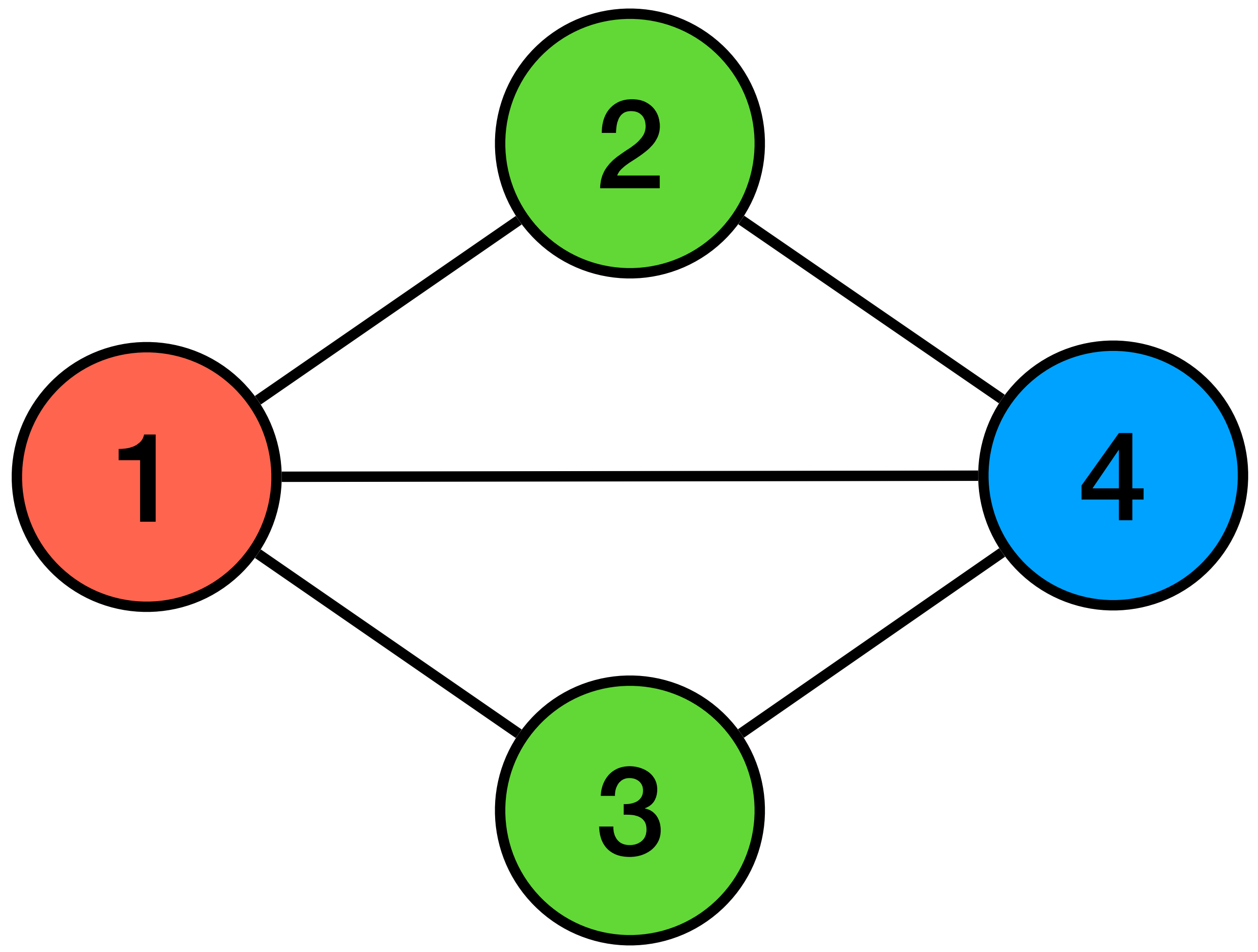
Backtracking

1	2	3	4	
R				
R	G			
R	G	B		
R	G	B	?	✗
R	G	G		



Backtracking

1	2	3	4	
R				
R	G			
R	G	B		
R	G	B	?	✗
R	G	G		
R	G	G	B	✓



DPLL algorithm

- Introduced by Davis, Putman, Logemann, and Loveland
- Backtracking algorithm that incrementally searches for a satisfiable assignment
- Works on propositional logic formulas in **conjunctive normal form** (CNF)

CNF

- A formula in CNF is a conjunction of clauses

$$(A \vee B) \wedge (B \vee \neg C \vee D) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg C \vee \neg D) \wedge A$$

- A **clause** is disjunction of literals
- A **literal** is a propositional variable or its negation
- Any formula can converted to CNF by applying well-known rewrite rules
- Formulas in CNF can be represented in a compact way

$$\{\{A, B\}, \{B, \overline{C}, D\}, \{\overline{A}, \overline{B}\}, \{\overline{A}, \overline{C}, \overline{D}\}, \{A\}\}$$

Conversion to CNF

$$
\begin{array}{ccc}
\phi \leftrightarrow \psi & \Rightarrow & (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \\
\phi \rightarrow \psi & \Rightarrow & \neg\phi \vee \psi \\
\neg\neg\phi & \Rightarrow & \phi \\
\neg(\phi \vee \psi) & \Rightarrow & \neg\phi \wedge \neg\psi \\
\neg(\phi \wedge \psi) & \Rightarrow & \neg\phi \vee \neg\psi \\
\phi \vee (\psi \wedge \theta) & \Rightarrow & (\phi \vee \psi) \wedge (\phi \vee \theta) \\
(\psi \wedge \theta) \vee \phi & \Rightarrow & (\psi \vee \phi) \wedge (\theta \vee \phi)
\end{array}
$$

DPLL algorithm

- Start from an empty assignment
- Repeat the following steps
  - **Deduce** the value of literals and perform **unit propagation**
  - If deduce is not possible then **guess** the value of literal
  - If a clause becomes empty we reached a contradiction (aka **conflict**)
    - Backtrack to last “open” guess and try opposite value
  - If the formula becomes empty we reached a satisfiable assignment
- If the search terminates without finding a satisfiable assignment the formula is unsatisfiable

Unit propagation

- A **unit clause** contains only one literal

$$\{\{A, B\}, \{B, \overline{C}, D\}, \{\overline{A}, \overline{B}\}, \{\overline{A}, \overline{C}, \overline{D}\}, \{A\}\}$$

- The formula can only be satisfied if that literal is made true
- And **unit propagation** can be performed
  - Discard all clauses containing the literal
  - Discard the literal complement from all clauses

$$\{\{B, \overline{C}, D\}, \{\overline{B}\}, \{\overline{C}, \overline{D}\}\}$$

Example SAT

A	B	C	D	

$$\{\{A, B\}, \{B, \overline{C}, D\}, \{\overline{A}, \overline{B}\}, \{\overline{A}, \overline{C}, \overline{D}\}, \{A\}\}$$

Example SAT

A	B	C	D	
1				deduce A

$$\{\{A, B\}, \{B, \overline{C}, D\}, \{\overline{A}, \overline{B}\}, \{\overline{A}, \overline{C}, \overline{D}\}, \{A\}\}$$

$$\{\{B, \overline{C}, D\}, \{\overline{B}\}, \{\overline{C}, \overline{D}\}\}$$

Example SAT

A	B	C	D	
1				deduce A
1	0			deduce $\overline{B}$

$$\{\{A, B\}, \{B, \overline{C}, D\}, \{\overline{A}, \overline{B}\}, \{\overline{A}, \overline{C}, \overline{D}\}, \{A\}\}$$

$$\{\{B, \overline{C}, D\}, \{\overline{B}\}, \{\overline{C}, \overline{D}\}\}$$

$$\{\{\overline{C}, D\}, \{\overline{C}, \overline{D}\}\}$$

Example SAT

A	B	C	D	
1				deduce A
1	0			deduce $\overline{B}$
1	0	1		guess C

$$\{\{A, B\}, \{B, \overline{C}, D\}, \{\overline{A}, \overline{B}\}, \{\overline{A}, \overline{C}, \overline{D}\}, \{A\}\}$$

$$\{\{B, \overline{C}, D\}, \{\overline{B}\}, \{\overline{C}, \overline{D}\}\}$$

$$\{\{\overline{C}, D\}, \{\overline{C}, \overline{D}\}\}$$

$$\{\{D\}, \{\overline{D}\}\}$$

Example SAT

A	B	C	D		
					$\{\{A,B\},\{B,\overline{C},D\},\{\overline{A},\overline{B}\},\{\overline{A},\overline{C},\overline{D}\},\{A\}\}$
1				deduce A	$\{\{B,\overline{C},D\},\{\overline{B}\},\{\overline{C},\overline{D}\}\}$
1	0			deduce $\overline{\text{B}}$	$\{\{\overline{C},D\},\{\overline{C},\overline{D}\}\}$
1	0	1		guess C	$\{\{D\},\{\overline{D}\}\}$
1	0	1	1	deduce D	$\{\{\}\}$

Example SAT

A	B	C	D		
					$\{\{A,B\},\{B,\overline{C},D\},\{\overline{A},\overline{B}\},\{\overline{A},\overline{C},\overline{D}\},\{A\}\}$
1				deduce A	$\{\{B,\overline{C},D\},\{\overline{B}\},\{\overline{C},\overline{D}\}\}$
1	0			deduce $\overline{\text{B}}$	$\{\{\overline{C},D\},\{\overline{C},\overline{D}\}\}$
1	0	1		guess C	$\{\{D\},\{\overline{D}\}\}$
1	0	1	1	deduce D	$\{\{\}\}$
1	0	0		guess $\overline{\text{C}}$	$\{\}$

Example UNSAT

A	B	C	

$$\{\{A, B\}, \{\overline{A}, B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

Example UNSAT

A	B	C	
1			guess A

$$\{\{A, B\}, \{\overline{A}, B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

Example UNSAT

A	B	C	
1			guess A
1	1		deduce B

$$\{\{A, B\}, \{\overline{A}, B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{\overline{C}\}, \{C\}\}$$

Example UNSAT

A	B	C	
1			guess A
1	1		deduce B
1	1	1	deduce C

$$\{\{A, B\}, \{\overline{A}, B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{\overline{C}\}, \{C\}\}$$

$$\{\{\}\}$$

Example UNSAT

A	B	C	
1			guess A
1	1		deduce B
1	1	1	deduce C
0			guess $\overline{\text{A}}$

$$\{\{A, B\}, \{\overline{A}, B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{\overline{C}\}, \{C\}\}$$

$$\{\{\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

Example UNSAT

A	B	C	
1			guess A
1	1		deduce B
1	1	1	deduce C
0			guess $\overline{A}$
0	1		deduce B

$$\{\{A, B\}, \{\overline{A}, B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{\overline{C}\}, \{C\}\}$$

$$\{\{\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{\overline{C}\}, \{C\}\}$$

Example UNSAT

A	B	C	
1			guess A
1	1		deduce B
1	1	1	deduce C
0			guess $\overline{A}$
0	1		deduce B
0	1	1	deduce C

$$\{\{A, B\}, \{\overline{A}, B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{\overline{C}\}, \{C\}\}$$

$$\{\{\}\}$$

$$\{\{B\}, \{\overline{B}, \overline{C}\}, \{\overline{B}, C\}\}$$

$$\{\{\overline{C}\}, \{C\}\}$$

$$\{\{\}\}$$

CDCL algorithm

- Conflict-Driven Clause Learning improves DPLL significantly
- CDCL keeps an implication graph with the decisions that led to a conflict
- From this graph it can “learn” a new clause that rules out that conflict
- Backtrack directly jumps (non-chronologically) to a decision where the conflict can still be avoided
- CDCL is at the basis of state-of-the-art SAL solvers

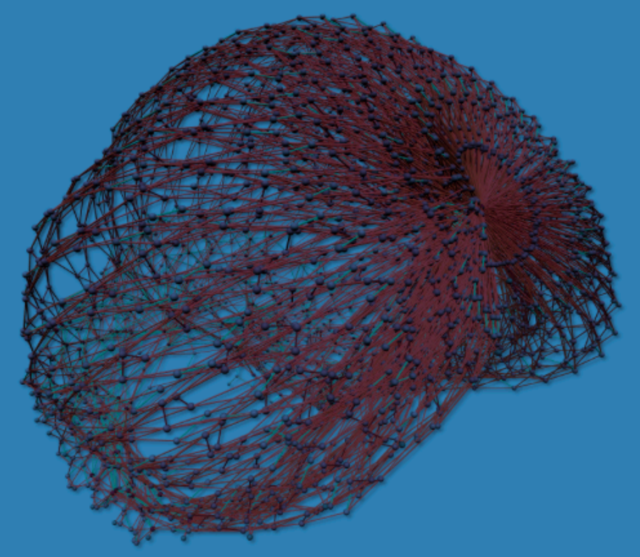
DIMACS

- Standard textual format for CNF
- Used by most SAT solvers
- Starts by declaring the number of variables and the number of clauses
- Each clause is described by a sequence of literals ended by 0
- Each literal is either a positive number identifying a variable or its negation

DIMACS

$$\{\{A, B\}, \{B, \overline{C}, D\}, \{\overline{A}, \overline{B}\}, \{\overline{A}, \overline{C}, \overline{D}\}, \{A\}\}$$

```
c example
p cnf 4 5
1 2 0
2 -3 4 0
-1 -2 0
-1 -3 -4 0
1 0
```



SAT Competition 2024

Overview

Competition Tracks

Solver Submission

- Output Format
- StarExec Cluster
- AWS Cloud

Benchmark Submission

Results

Organizers

SAT Competition 2024 is a competitive event for solvers of the Boolean Satisfiability (SAT) problem. The competition is organized as a satellite event to the SAT Conference 2024 and continues the series of the annual SAT Competitions and SAT-Races / Challenges.

Objective

The area of SAT Solving has seen tremendous progress over the last years. Many problems that seemed to be out of reach a decade ago can now be handled routinely. Besides new algorithms and better heuristics, refined implementation techniques turned out to be vital for this success. To keep up the driving force in improving SAT solvers, we want to motivate implementers to present their work to a broader audience and to compare it with that of others. Researchers from both academia and industry are invited to submit their solvers and benchmarks.

News

- **2024-08-27** A separate download of the benchmark metadata is now available.
- **2024-08-27** The benchmarks are available for download at Zenodo and can be accessed in annotated form under https://benchmark-database.de?track=main_2024
- **2024-08-27** The sources of sequential solvers, parallel solvers, and distributed solvers are available for download.
- **2024-08-27** The HTML result tables of the SAT Competition 2024 are available and the detailed results can be downloaded.
- **2024-08-25** The Proceedings of the SAT Competition are available online.
- **2024-08-25** The presentation of results is now available for download.
- **2024-01-26** We will replace the satisfying assignments checker that has traditionally been used in SAT competitions. This will allow partial models to be accepted as long as every clause of the original SAT instance is satisfied.
- **2024-01-22** The competition website is online.

Tracks

Allocation problems

Allocation problem

- Decide if a set of “items” can be placed in a set of “containers”
- Subject to a set of generic (often implicit) constraints
  - At least / at most one item per container
  - At least / at most one container per item
- And a set of problem specific constraints
  - A specific item does not want to be placed in a specific container
  - Two specific items do not want to be placed together
  - ...

Allocating with SAT

- Declare a matrix of $|\text{Items}| \times |\text{Containers}|$ of propositional variables
    - Variable $p_{x,a}$ is true iff item x is allocated to container a
- Specify the allocation constraints with propositional formulas
- SAT solve to get allocation

Generic constraints

- At least one container per item

$$(p_{a,1} \vee p_{a,2} \vee p_{a,3}) \wedge (p_{b,1} \vee p_{b,2} \vee p_{b,3})$$

- At most one container per item

<table>
<tr><td colspan="2"></td><td colspan="3">Container</td></tr>
<tr><td colspan="2"></td><td>1</td><td>2</td><td>3</td></tr>
<tr><td rowspan="2">Item</td><td>a</td><td>$p_{a,1}$</td><td>$p_{a,2}$</td><td>$p_{a,3}$</td></tr>
<tr><td>b</td><td>$p_{b,1}$</td><td>$p_{b,2}$</td><td>$p_{b,3}$</td></tr>
</table>

Generic constraints

- At least one container per item

$$\bigwedge_{x \in \text{Item}} (p_{x,1} \vee p_{x,2} \vee p_{x,3})$$

- At most one container per item

		Container		
		1	2	3
Item	a	$p_{a,1}$	$p_{a,2}$	$p_{a,3}$
	b	$p_{b,1}$	$p_{b,2}$	$p_{b,3}$

Generic constraints

- At least one container per item

$$\bigwedge_{x \in \text{Item}} \bigvee_{a \in \text{Container}} p_{x,a}$$

- At most one container per item

		Container		
		1	2	3
Item	a	$p_{a,1}$	$p_{a,2}$	$p_{a,3}$
	b	$p_{b,1}$	$p_{b,2}$	$p_{b,3}$

Generic constraints

- At least one container per item

$$\bigwedge_{x \in \text{Item}} \bigvee_{a \in \text{Container}} p_{x,a}$$

- At most one container per item

$$\bigwedge_{x \in \text{Item}} \bigwedge_{a < b \in \text{Container}} (\neg p_{x,a} \vee \neg p_{x,b})$$

		Container		
		1	2	3
Item	a	$p_{a,1}$	$p_{a,2}$	$p_{a,3}$
	b	$p_{b,1}$	$p_{b,2}$	$p_{b,3}$

Generic constraints

- At least one container per item

$$\bigwedge_{x \in \text{Item}} \bigvee_{a \in \text{Container}} p_{x,a}$$

		Container		
		1	2	3
Item	a	$p_{a,1}$	$p_{a,2}$	$p_{a,3}$
	b	$p_{b,1}$	$p_{b,2}$	$p_{b,3}$

- At most one container per item

$$(\neg p_{a,1} \vee \neg p_{a,2}) \wedge (\neg p_{a,1} \vee \neg p_{a,3}) \wedge (\neg p_{a,2} \vee \neg p_{a,3})$$
$$\wedge$$
$$(\neg p_{b,1} \vee \neg p_{b,2}) \wedge (\neg p_{b,1} \vee \neg p_{b,3}) \wedge (\neg p_{b,2} \vee \neg p_{b,3})$$

Generic constraints

- At least one container per item

$$\bigwedge_{x \in \text{Item}} \bigvee_{a \in \text{Container}} p_{x,a}$$

		Container		
		1	2	3
Item	a	$p_{a,1}$	$p_{a,2}$	$p_{a,3}$
	b	$p_{b,1}$	$p_{b,2}$	$p_{b,3}$

- At most one container per item

$$\bigwedge_{x \in \text{Item}} ((\neg p_{x,1} \vee \neg p_{x,2}) \wedge (\neg p_{x,1} \vee \neg p_{x,3}) \wedge (\neg p_{x,2} \vee \neg p_{x,3}))$$

Generic constraints

- At least one container per item

$$\bigwedge_{x \in \text{Item}} \bigvee_{a \in \text{Container}} p_{x,a}$$

- At most one container per item

$$\bigwedge_{x \in \text{Item}} \bigwedge_{a < b \in \text{Container}} (\neg p_{x,a} \vee \neg p_{x,b})$$

		Container		
		1	2	3
Item	a	$p_{a,1}$	$p_{a,2}$	$p_{a,3}$
	b	$p_{b,1}$	$p_{b,2}$	$p_{b,3}$

Checking assertions

- After adding the constraints describing the allocation problem we can check the validity of assertions

- To check if ϕ is valid add $\neg\phi$ as a constraint and check if the problem is unsat

Placement of guests

- We have three chairs in a row and need to place Anne, Susan and Peter
    - Anne does not want to sit next to Peter
    - Anne does not want to sit in the left chair
    - Susan does not want to sit to the left of Peter
- Implicit generic constraints
    - Everyone must be sited in a chair (at least one chair per guest)
    - No more than one guest per chair (at most one guest per chair)

Variables

		Chair		
		Left	Center	Right
Guest	Anne	$x_{a,l}$	$x_{a,c}$	$x_{a,r}$
	Susan	$x_{s,l}$	$x_{s,c}$	$x_{s,r}$
	Peter	$x_{p,l}$	$x_{p,c}$	$x_{p,r}$

At least one chair per guest

$$\bigwedge_{i \in \text{Guest}} \bigvee_{k \in \text{Chair}} x_{i,k}$$
$$\equiv$$
$$(x_{a,l} \lor x_{a,c} \lor x_{a,r}) \land (x_{s,l} \lor x_{s,c} \lor x_{s,r}) \land (x_{p,l} \lor x_{p,c} \lor x_{p,r})$$

At most one guest per chair

$$\bigwedge_{k \in \text{Chair}} \bigwedge_{i<j \in \text{Guest}} (\neg x_{i,k} \vee \neg x_{j,k})$$
$$\equiv$$
$$(\neg x_{a,l} \vee \neg x_{s,l}) \wedge (\neg x_{a,l} \vee \neg x_{p,l}) \wedge (\neg x_{s,l} \vee \neg x_{p,l})$$
$$\wedge$$
$$(\neg x_{a,c} \vee \neg x_{s,c}) \wedge (\neg x_{a,c} \vee \neg x_{p,c}) \wedge (\neg x_{s,c} \vee \neg x_{p,c})$$
$$\wedge$$
$$(\neg x_{a,r} \vee \neg x_{s,r}) \wedge (\neg x_{a,r} \vee \neg x_{p,r}) \wedge (\neg x_{s,r} \vee \neg x_{p,r})$$

Problem specific constraints

- Anne does not want to sit next to Peter

$$(x_{a,c} \rightarrow (\neg x_{p,l} \wedge \neg x_{p,r})) \wedge ((x_{a,l} \vee x_{a,r}) \rightarrow \neg x_{p,c})$$

- Anne does not want to sit in the left chair

$$\neg x_{a,l}$$

- Susan does not want to sit to the left of Peter

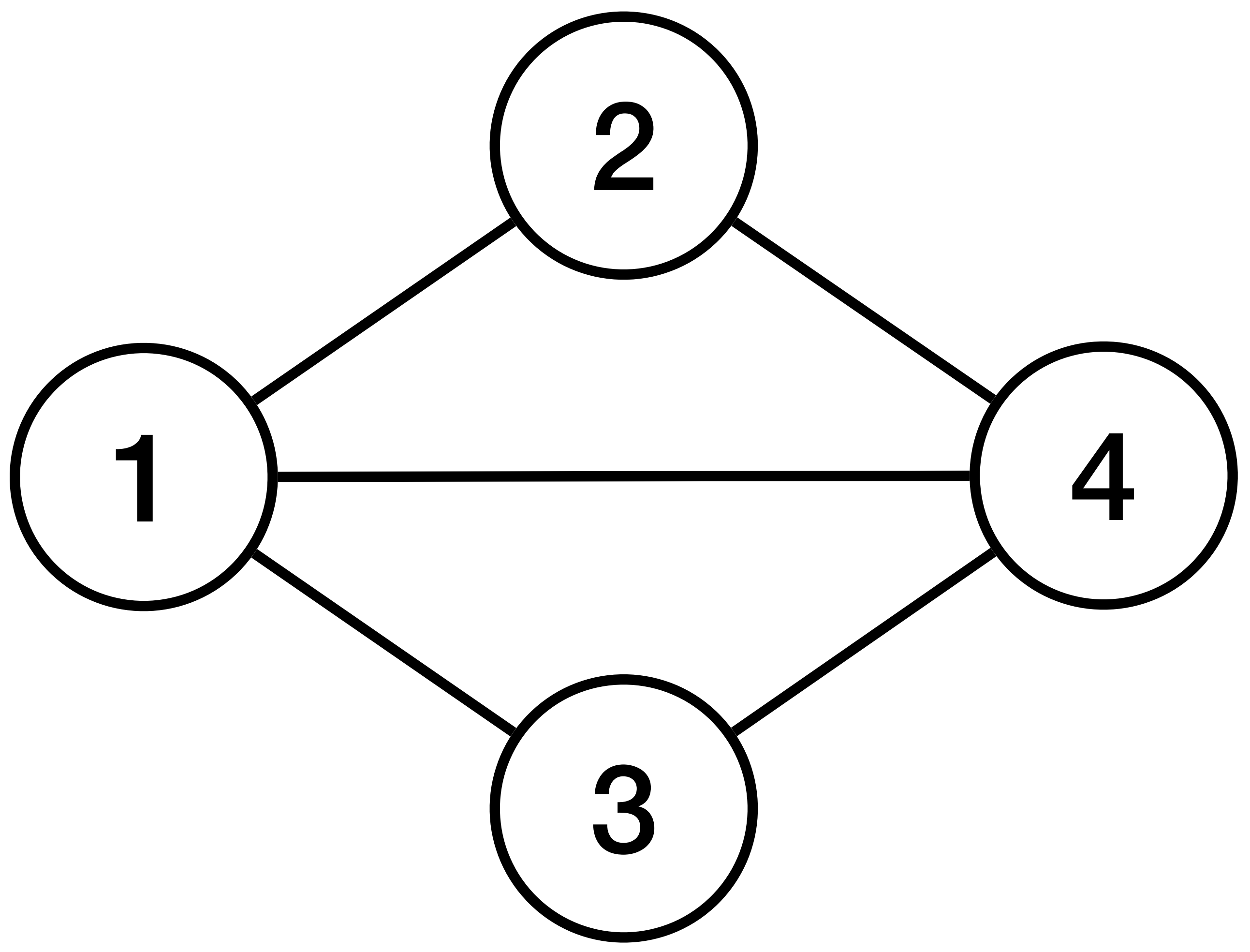
$$(x_{p,r} \rightarrow \neg x_{s,c}) \wedge (x_{p,c} \rightarrow \neg x_{s,l})$$

Let’s do it with Z3!

DEMO

Graph colouring

- Can the following graph be coloured with 3 colours?
- Allocate colours to vertices such that
  - Every vertex has one colour
    - At least one colour per vertex
    - At most one colour per vertex
  - Adjacent vertices have different colours



Variables

		Colour		
		Red	Green	Blue
Vertex	1	$x_{1,r}$	$x_{1,g}$	$x_{1,b}$
	2	$x_{2,r}$	$x_{2,g}$	$x_{2,b}$
	3	$x_{3,r}$	$x_{3,g}$	$x_{3,b}$
	4	$x_{4,r}$	$x_{4,g}$	$x_{4,b}$

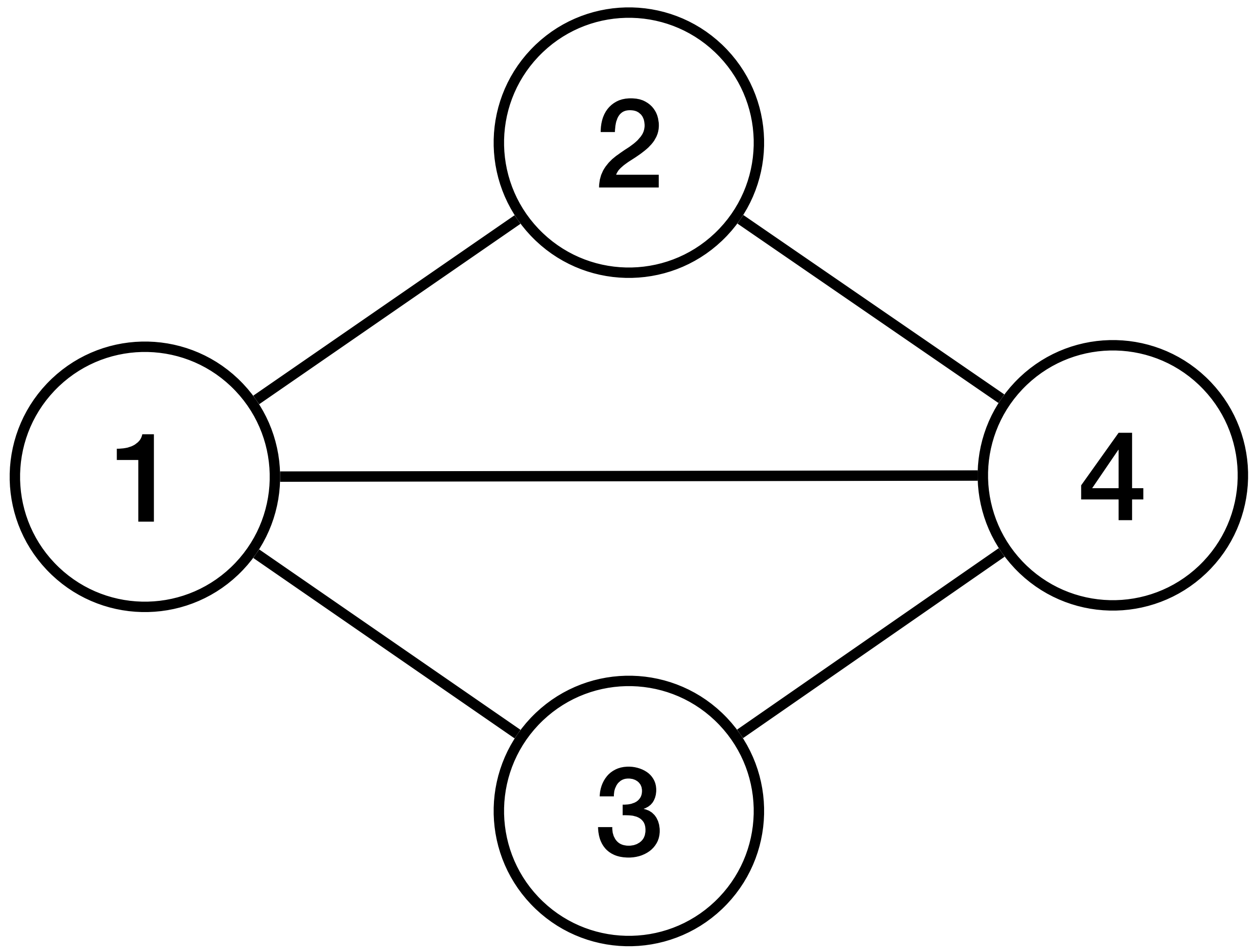
Every vertex has one colour

$$\bigwedge_{i \in \text{Vertex}} \bigvee_{k \in \text{Colour}} x_{i,k}$$

$$\bigwedge_{i \in \text{Vertex}} \bigwedge_{k < l \in \text{Colour}} (\neg x_{i,k} \vee \neg x_{i,l})$$

Adjacent vertices have different colours

$$\bigwedge_{k \in \text{Colour}} (\neg x_{1,k} \vee \neg x_{2,k})$$
$$\wedge$$
$$\bigwedge_{k \in \text{Colour}} (\neg x_{1,k} \vee \neg x_{3,k})$$
$$\wedge$$
$$\bigwedge_{k \in \text{Colour}} (\neg x_{1,k} \vee \neg x_{4,k})$$
$$\wedge$$
$$\bigwedge_{k \in \text{Colour}} (\neg x_{2,k} \vee \neg x_{4,k})$$
$$\wedge$$
$$\bigwedge_{k \in \text{Colour}} (\neg x_{3,k} \vee \neg x_{4,k})$$



Let’s do it with Z3!

DEMO