An Introduction to Proof Complexity

Paul Beame

University of Washington

Separating P and NP

- **NP** is characterized by a simple property - having short proofs of membership

- To prove **NP $\neq$ coNP** show that **coNP** doesn't have this property [Cook 70's]
  - would separate **P** from **NP** so probably quite hard
  - Lots of nice, very useful smaller steps towards answering this question

Proving language membership

- Proof of satisfiability
  - Satisfying truth assignment
  - Always short, SAT$\in$NP
- Proof of **unsatisfiability**
  - ?????
  - transcript of failed search for satisfying truth assignment
  - Truth tables, Frege-Hilbert proofs, resolution
  - Can they always be short? If so then NP=co-NP.

Proof systems

- A proof system for a language **L** is a polynomial time algorithm **V** s.t.
  - for all inputs **x**
    - **x**$\in$**L** iff ***there exists a string P s.t. V accepts input (x,P)***
    -
  - think of **P** as a *proof* that **x** is in **L** and **V** as a proof *verifier*

Complexity of proof systems

- **Defn:** The **complexity** a proof system **V** is a function **$f:N\rightarrow N$** defined by

$$\mathbf{f(n)} = \max_{x\in L, |x|=n} \; \min_{P:\, V \text{ accepts } (x,P)} |P|$$

  - i.e. how large **P** has to be as a function of **|x|**
  - **V** is **polynomially-bounded** **iff** its complexity is a polynomial function of **n**

- **NP** = {**L**: **L** has a polynomially-bounded proof system}

Propositional proof systems

- A propositional proof system is a proof system for the set **TAUT** of propositional logic tautologies
  - i.e. polynomial time algorithm **V** s.t. for all formulas **F**
    - ***F is a tautology***

      Û ***there exists a string P s.t.***

      V accepts input (P,F)
    - Note:
      - Ü **direction is usually called soundness**
      - Þ **direction is usually called completeness**

Propositional proof systems

- A propositional proof system is a proof system for the set **UNSAT** of unsatisfiable propositional logic formulas
  - i.e. polynomial time algorithm **V** s.t. for all formulas **F**
    - ***F* is a unsatisfiable**

      Û ***there exists a string P s.t.***

      V accepts input (P,F)

Polynomially-bounded proofs

- **Thm:** There is a polynomially-bounded propositional proof system **iff** **NP=coNP**

- Proof:
  - **SAT** is **NP-complete**
  - **F∈TAUT iff ØF∈UNSAT iff ØF∉SAT**
    - **so TAUT**, **UNSAT** are **coNP-complete**
    - **so TAUT, UNSAT∈NP iff NP=coNP**
  - ∃p-bounded proof system for **L iff L∈NP**

Sample propositional proof systems

- Truth tables
  - proof is a fully filled out truth table
    - easy to verify that it is filled out correctly and all truth assignments yield T
- Axiom/Inference systems
  - inference rules: **e.g. modus ponens** **A, (A $\rightarrow$ B) | B**
  - axioms: **e.g. excluded middle** **| (A Ú ØA)**
  - axioms & inference rules are **schemas**
    - can make consistent substitution of arbitrary formulas for variables in schema
    - e.g. excluded middle yields **((xÙØy) Ú Ø(xÙØy))**
  - more precisely...

Frege Systems

- Finite, implicationally complete set **R** of axioms/inference rules
- Refutation version:
  - Proof of unsatisfiability of **F** - sequence **$F_1, \dots, F_r$** of formulas (called lines) s.t.
    - **$F_1 = F$**
    - each **F_j** follows from an axiom in **R** or follows from previous ones via an inference rule in **R**
    - **F_r** = L trivial falsehood, e.g. **(x ÙØx)**
- Positive version:
  - Start with nothing, end with tautology **F**

Sample Frege Refutation

Subset of rules

a. **A**, **(A** ® **B)** | **B**

b. **(A** Ù **B)** | **A**

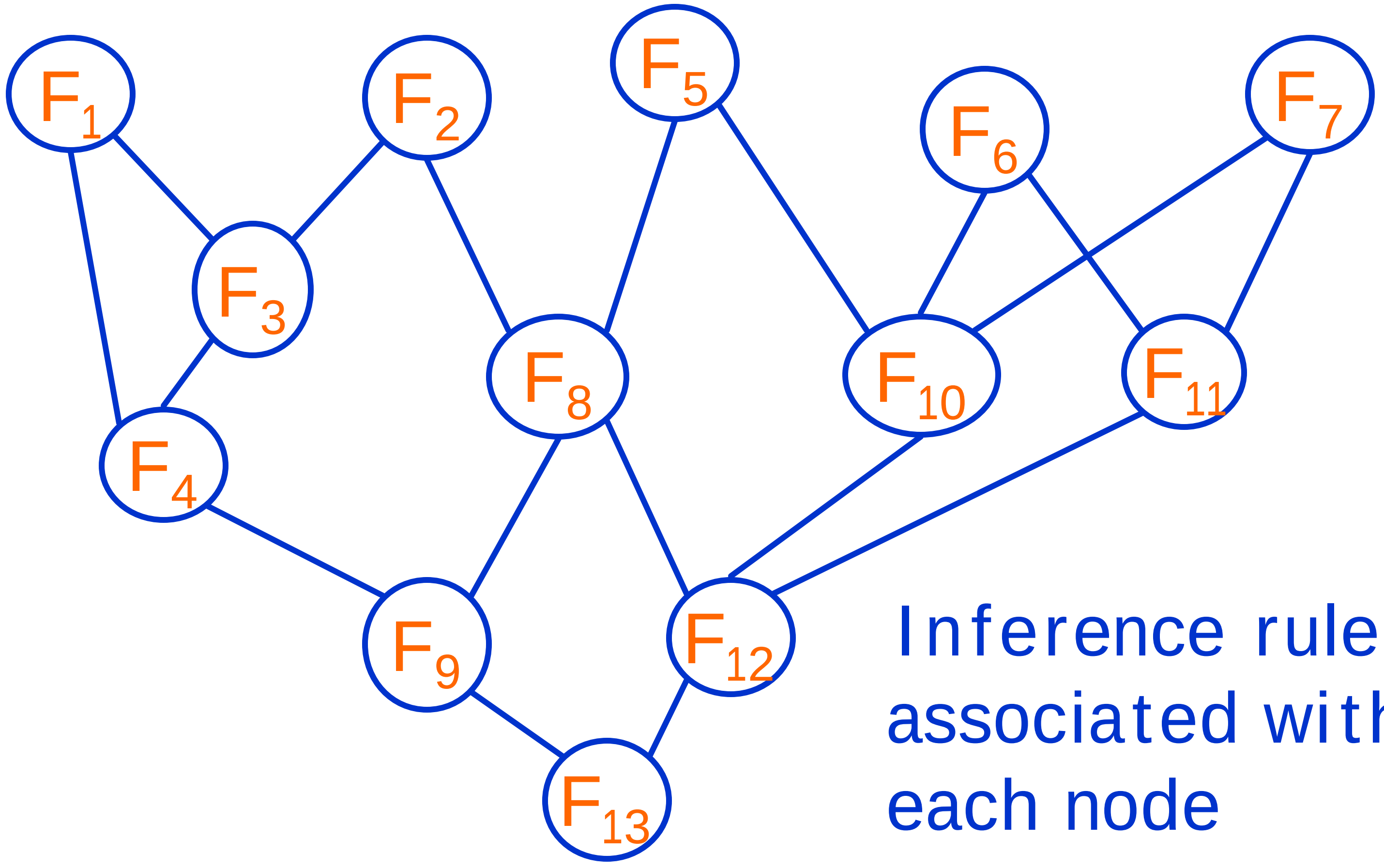
c. **(A** Ù **B)** | **B**

d. **A**, **B** | **(A** Ù **B)**

1. **((xÙ(x® y))Ù((x Ù y)®Øx))**	Given
2. **(x Ù(x ® y))**	From 1 by b
3. **((x Ù y) ® Øx)**	From 1 by c
4. **x**	From 2 by b
5. **(x ® y)**	From 2 by c
6. **y**	From 4,5 by a
7. **(x Ù y)**	From 4,6 by d
8. **Øx**	From 6,3 by a
9. **(x Ù Øx)** **=** L	From 4,8 by d

The graph of a proof

Axioms/inputs are sources



Inference rule associated with each node

Sink labelled by tautology (or Λ for refutation)

p-simulation

- **Defn:** Proof system **U** **polynomially simulates** proof system **V** iff
  - **they prove the same language L**

    $P. **V** accepts **(x,P)** Û $P'. **U** accepts **(x,P')**

  - **proofs in V can be efficiently converted into proofs in U**
    - i.e. there is a polynomial-time computable function **f** such that

      V accepts **(x,P)** Û **U** accepts **(x,f(P))**

- **Defn: U** and **V** are **polynomially equivalent** iff they polynomially simulate each other

All Frege systems are p-equivalent

- Two Frege systems given by axiom/inference rule sets R_1, R_2
  - The general form of an axiom/inference rule is

    $$G_1,\ G_2, \ldots,\ G_k \quad | \quad H$$

    i.e. given $G_1,\ldots,G_k$ conclude H (if $k=0$ then rule is an axiom)
  - since R_1 is complete and R_2 is sound & finite,
    - for every schema s in R_2 as above there is a constant sized proof in R_1 of the tautology $(G_1$ Ù G_2 Ù $\ldots$ Ù $G_k) \rightarrow H$
  - For every deduction of F from $F_1,\ldots,F_k$ in R_2 using s (i.e. $F_i=G_i[x:y]$, $F=H[x:y]$ for some substitution $[x:y]$)
    - derive $(F_1$ Ù F_2 Ù $\ldots$ Ù $F_k)$ which has a constant size proof from $F_1,\ldots,F_k$ in R_1
    - copy the R_1 proof of s but use the substitution $[x:y]$ at the start to prove $(F_1$ Ù F_2 Ù $\ldots$ Ù $F_k) \rightarrow F$
    - derive F from $(F_1$ Ù F_2 Ù $\ldots$ Ù $F_k)$ and $(F_1$ Ù F_2 Ù $\ldots$ Ù $F_k)\rightarrow F$ again constant size

Gentzen/Sequent Calculus

- Statements of the form $F_1,\ldots,F_k$ ® $G_1,\ldots,G_l$
  - meaning is $(F_1$ Ù…Ù $F_k)$ ® $(G_1$Ú … Ú $G_l)$
  - axioms F ® F
  - derive ® F to prove it
  - derive F® to refute it
- two rules for each connective, one for each side

$$\frac{G,F\text{®}D \quad G,G\text{®}D}{G,(F\text{Ú}G)\text{®}D} \qquad \frac{G\text{®}D,F}{G\text{®}D,(F\text{Ú}G)}$$

$$\frac{G\text{®}D,F}{G,\text{Ø}F\text{®}D} \qquad \frac{G,F\text{®}D}{G\text{®}D,\text{Ø}F}$$

- cut rule

$$\frac{G\text{®}D,F \quad P,F\text{®}S}{G,P\text{®}D,S}$$

Sequent Calculus & Frege

- Sequent calculus is p-equivalent to Frege
  - is still a proof system without the cut rule but is much weaker without it

- Can parametrize Sequent Calculus cleanly based on what kinds of formulas **F** used in the cut rule so it is often used in proof complexity but proofs are often cumbersome to write down so we don't use it here

Proof systems using CNF input

- By the same trick [Tseitin 68] that reduces **SAT** to **CNFSAT**, we can assume w.l.o.g. that propositional proof systems are for the languages **CNF-UNSAT** or **DNF-TAUT**
  - Add an extra variable y_G corresponding to each sub-formula **G** of propositional formula **F**
  - C_F includes clauses (or terms in the DNF case) expressing the fact that y_G takes on the value **G** determined by the inputs to the formula
  - Add clause y_F to express the truth value of **F**

  - **CLAIM:** \$b s.t.(a,b) satisfies C_F **iff** a satisfies **F**

Clauses

- if **G = H** Ú **J** include clauses
  - **(Øy_H** Ú **y_G)**
  - **(Øy_J** Ú **y_G)**
  - **(Øy_G** Ú **y_H** Ú **y_J)**
- if **G = H** Ù **J** include clauses
  - **(Øy_G** Ú **y_H)**
  - **(Øy_G** Ú **y_J)**
  - **(Øy_H** Ú **Øy_J** Ú **y_G)**
- if **G = ØH** include clauses
  - **(Øy_G** Ú **Øy_H)**
  - **(y_G** Ú **y_H)**

Resolution

- Frege-like system using CNF clauses only
- Start with original input clauses of CNF **F**
- Resolution rule
  - **(A Ú x), (B Ú Øx) | (A Ú B)**
- Goal: derive empty clause L
  - **equivalent to sequent calculus with cuts on literals**
- Most-popular systems for practical theorem-proving

C-Frege proof systems

- Many circuit complexity classes **C** are defined as follows:
  - **C**={**f**: **f** is computed by polynomial-size circuits with structural property P_C}
  - e.g. non-uniform classes **NC^1**, **AC^0**, **$AC^0[p]$**, **ACC**, **TC^0**, **P/poly**

- Define **C-Frege** to be the p-equivalence class of Frege-style proof systems s.t.
  - each line has structural property P_C
  - finite number of axioms/inference rules
  - complete for circuits with property P_C

Circuit Complexity

- **P/poly** - polysize circuits
- **NC^1** - polysize formulas = O(log n) depth fan-in 2
- **CNF** - polysize CNF formulas
- **AC^0** - constant-depth unbounded fan-in polysize circuits using and/or/not gates

- **$AC^0[m]$** - also = 0 mod m tests

- **TC^0** - threshold instead

What we know in circuit complexity

- **CNF** Ì **AC^0** Ì **$AC^0[p]$** Ì **TC^0** for **p** prime
- **TC^0** Í **NC^1** Í **P/poly** Í **NP/poly**
- **$AC^0[m]$** Ì # **P**

Examples

- **Frege** = **NC^1-Frege**
  - **NC^1** (logarithmic depth fan-in 2) circuits can be expanded into trees (formulas) of polynomial size
  - Formulas can always be re-balanced so they have logarithmic depth with only polynomial size increase

- **Resolution** is a special case of **'CNF-Frege'**
  - **CNF** is not strong enough to express the p-simulation among Frege systems
  - **Semantic Tableau** arbitrary sound **CNF** inferences of constant size

Extended Frege Proofs

- Like Frege proofs plus extra **extension** steps
  - that define new propositional variables to stand for arbitrary formulas on current set of variables (like the variables y_G in the conversion to CNF but for more than just the input formula)
  - after extension may write formulas more succinctly using newly-defined variables

- Each extension variable describes a circuit in the input variables
  - **Extended-Frege** = **P/poly-Frege**

Davis-Putnam (DLL) Procedure

- Both
  - a proof system
  - a collection of algorithms for finding proofs
- As a proof system
  - a special case of **resolution** where the proof graph forms a **tree**.
- The most widely used family of complete algorithms for satisfiability

Simple Davis-Putnam Algorithm

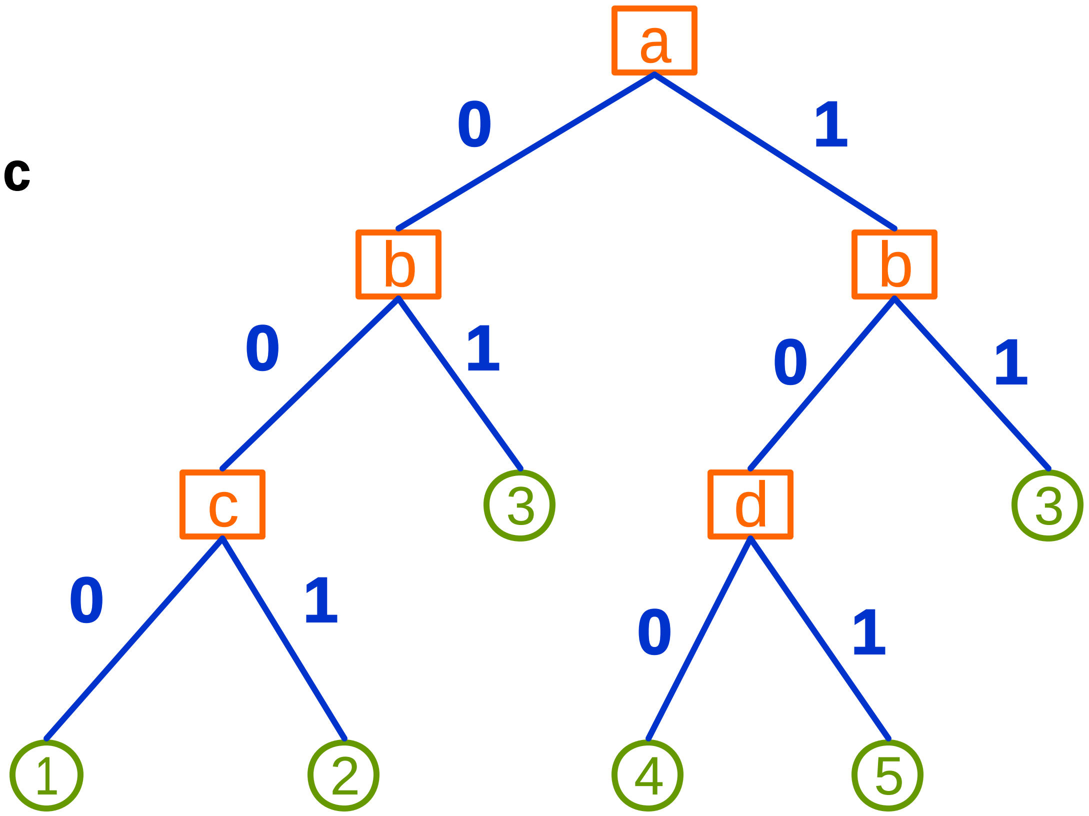
- Refute(**F**)
  - While (**F** contains a clause of size **1**)
    - set variable to make that clause true
    - simplify all clauses using this assignment
  - If **F** has no clauses then
    - output "**F** is satisfiable" and HALT
  - If F does not contain an empty clause then
    - Choose smallest-numbered unset variable **x**
    - Run Refute($\mathbf{F}_{\mathbf{x} \neg \mathbf{0}}$)
    - Run Refute($\mathbf{F}_{\mathbf{x} \neg \mathbf{1}}$)

splitting rule

DLL Refutation

Clauses

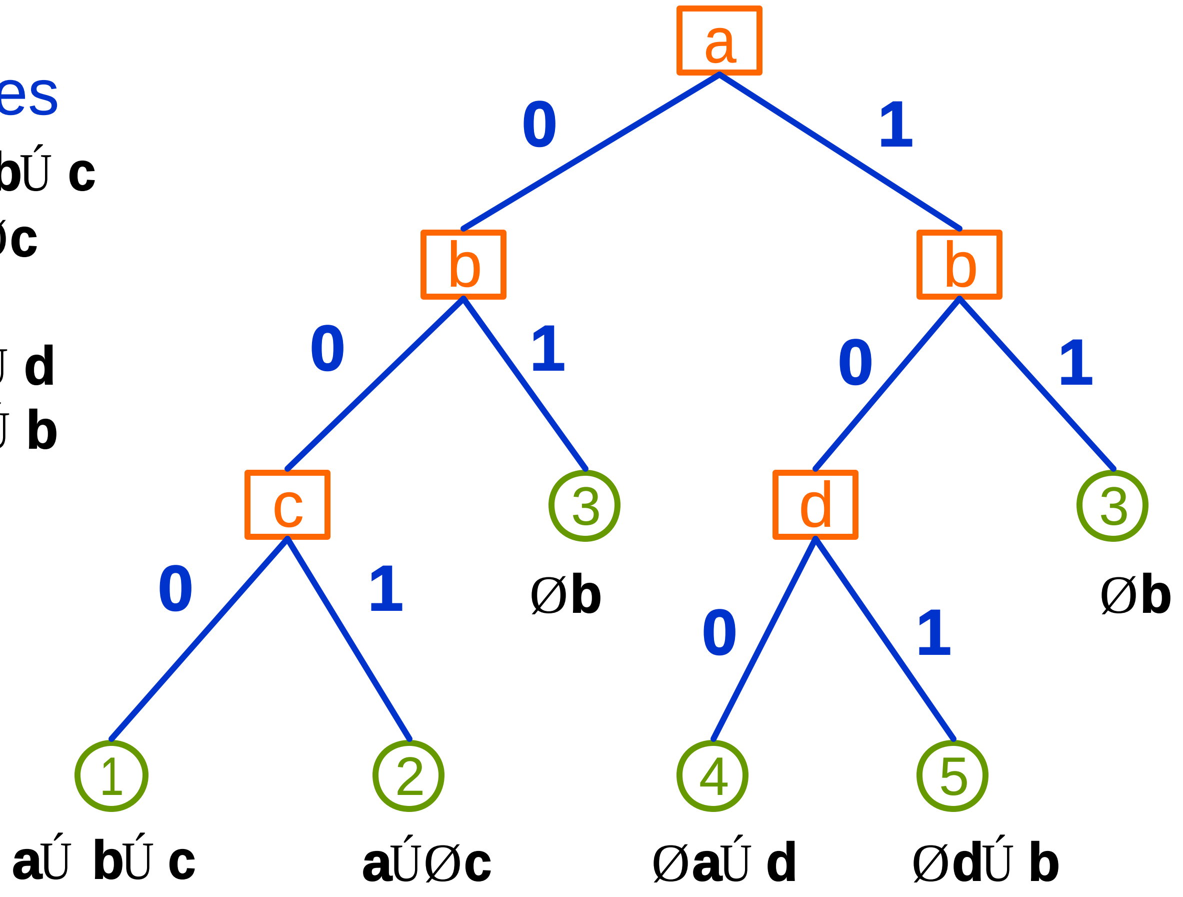
1. **aÚ bÚ c**
2. **aÚØc**
3. **Øb**
4. **ØaÚ d**
5. **ØdÚ b**



DLL Refutation

Clauses

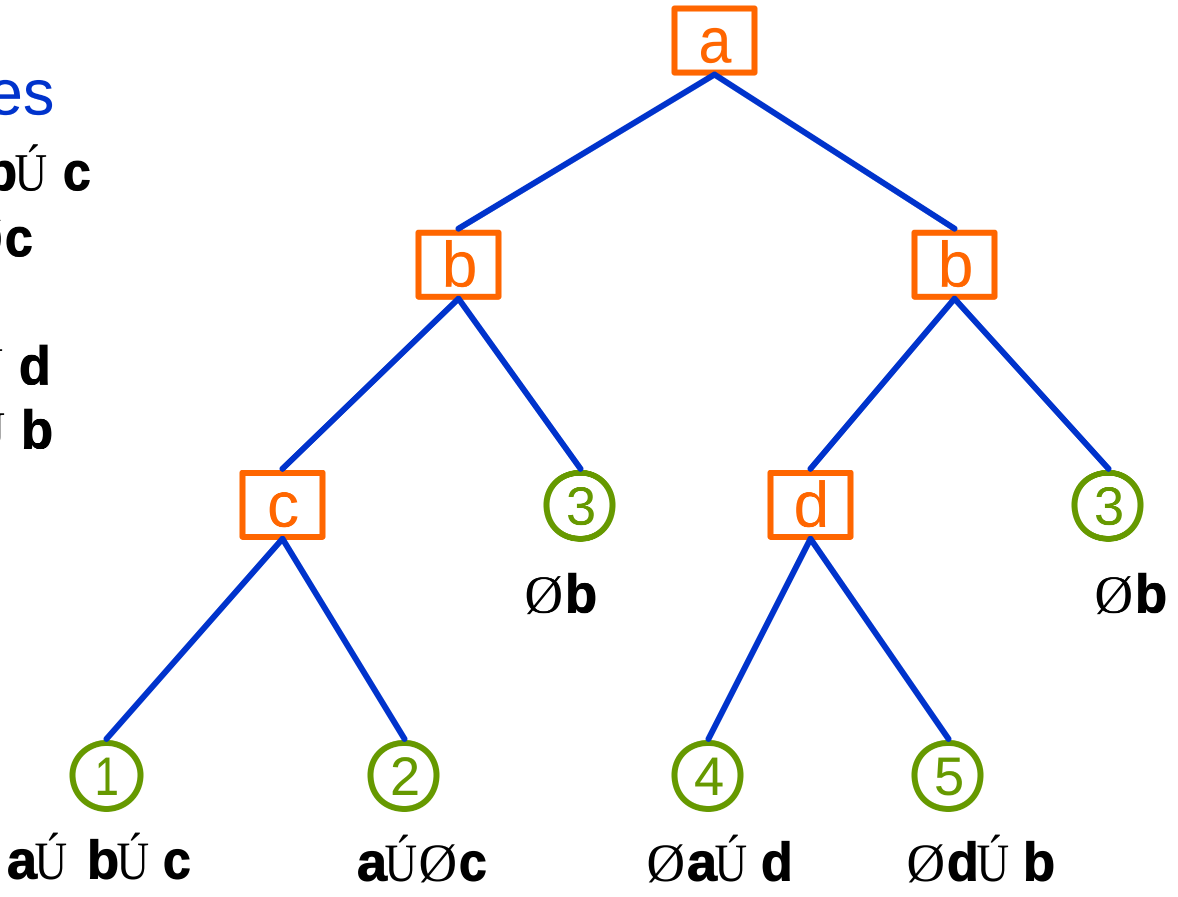
1. **aÚ bÚ c**
2. **aÚØc**
3. **Øb**
4. **ØaÚ d**
5. **ØdÚ b**



Tree Resolution

Clauses

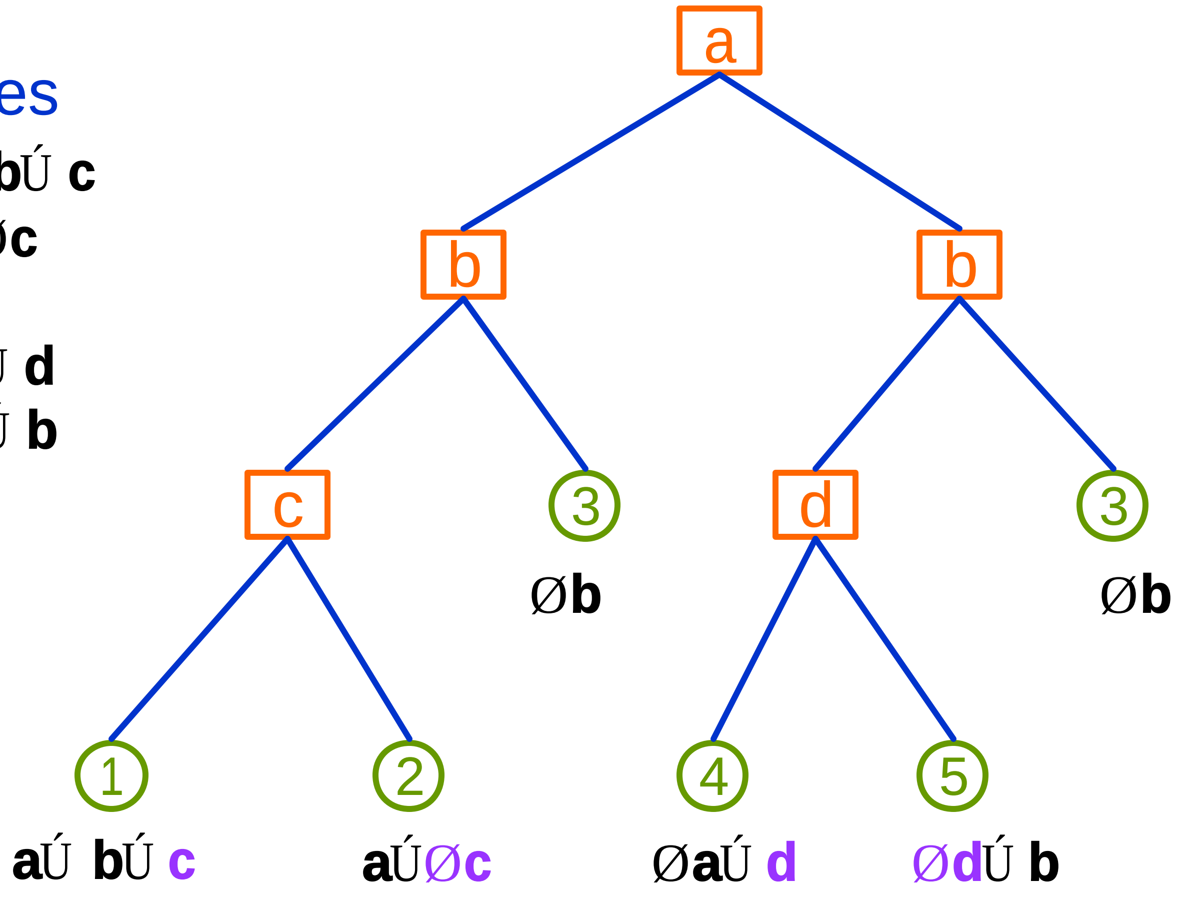
1. **aÚ bÚ c**
2. **aÚØc**
3. **Øb**
4. **ØaÚ d**
5. **ØdÚ b**



Tree Resolution

Clauses

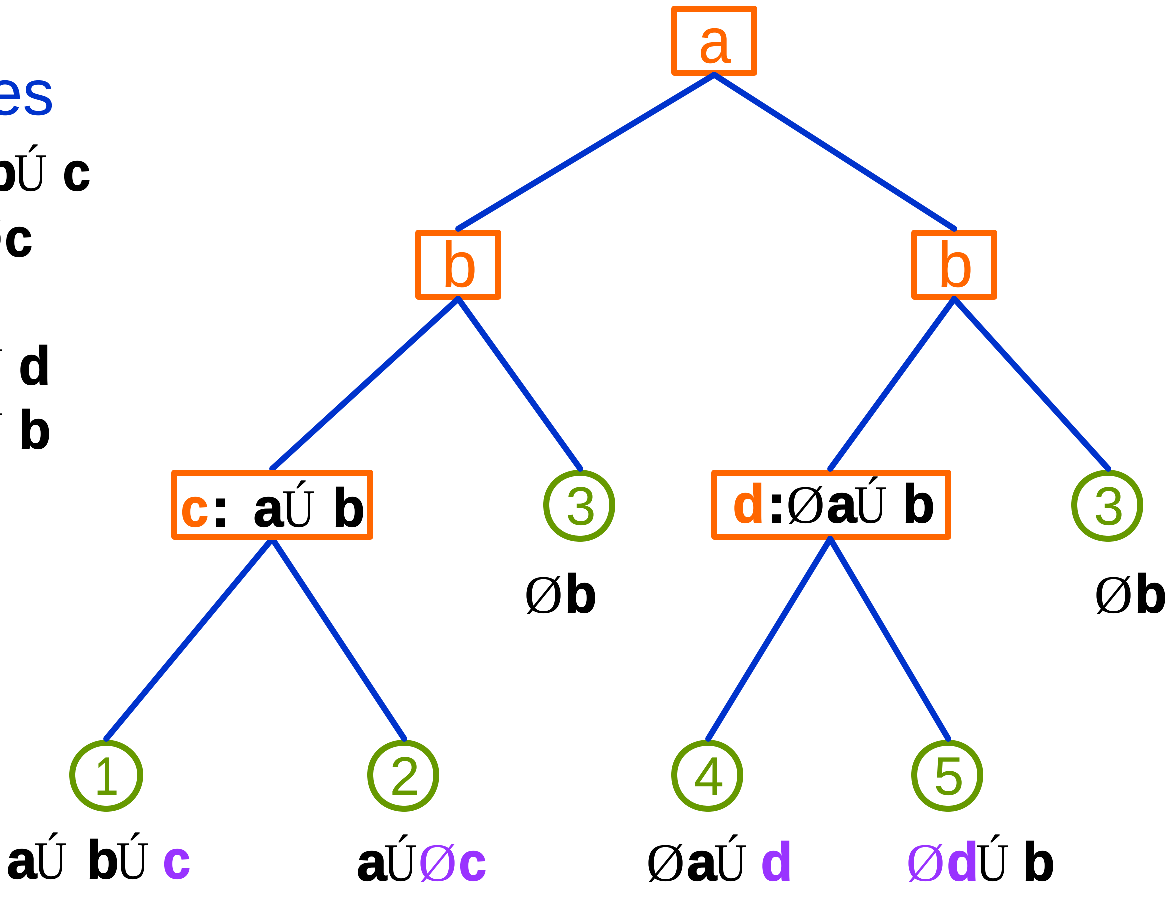
1. **aÚ bÚ c**
2. **aÚØc**
3. **Øb**
4. **ØaÚ d**
5. **ØdÚ b**



Tree Resolution

Clauses

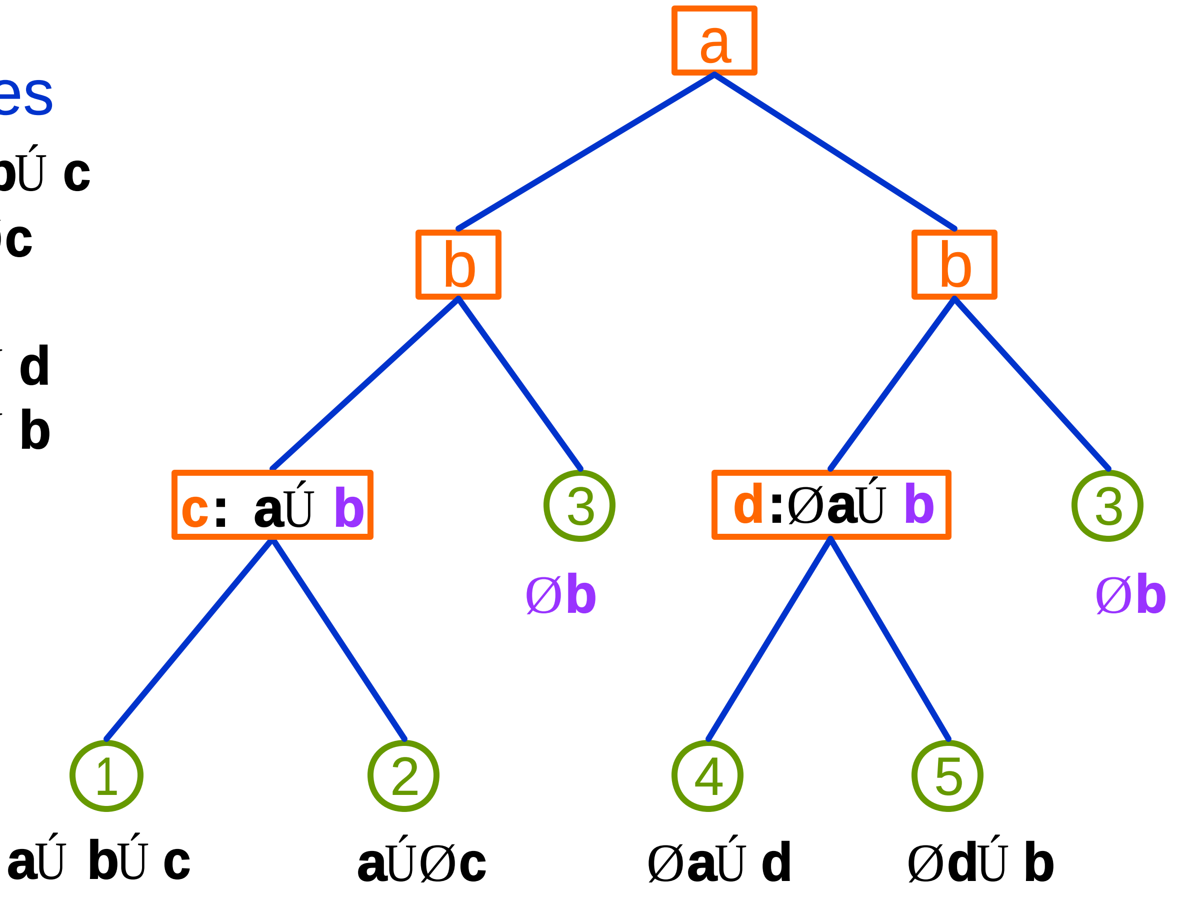
1. **aÚ bÚ c**
2. **aÚØc**
3. **Øb**
4. **ØaÚ d**
5. **ØaÚ b**



Tree Resolution

Clauses

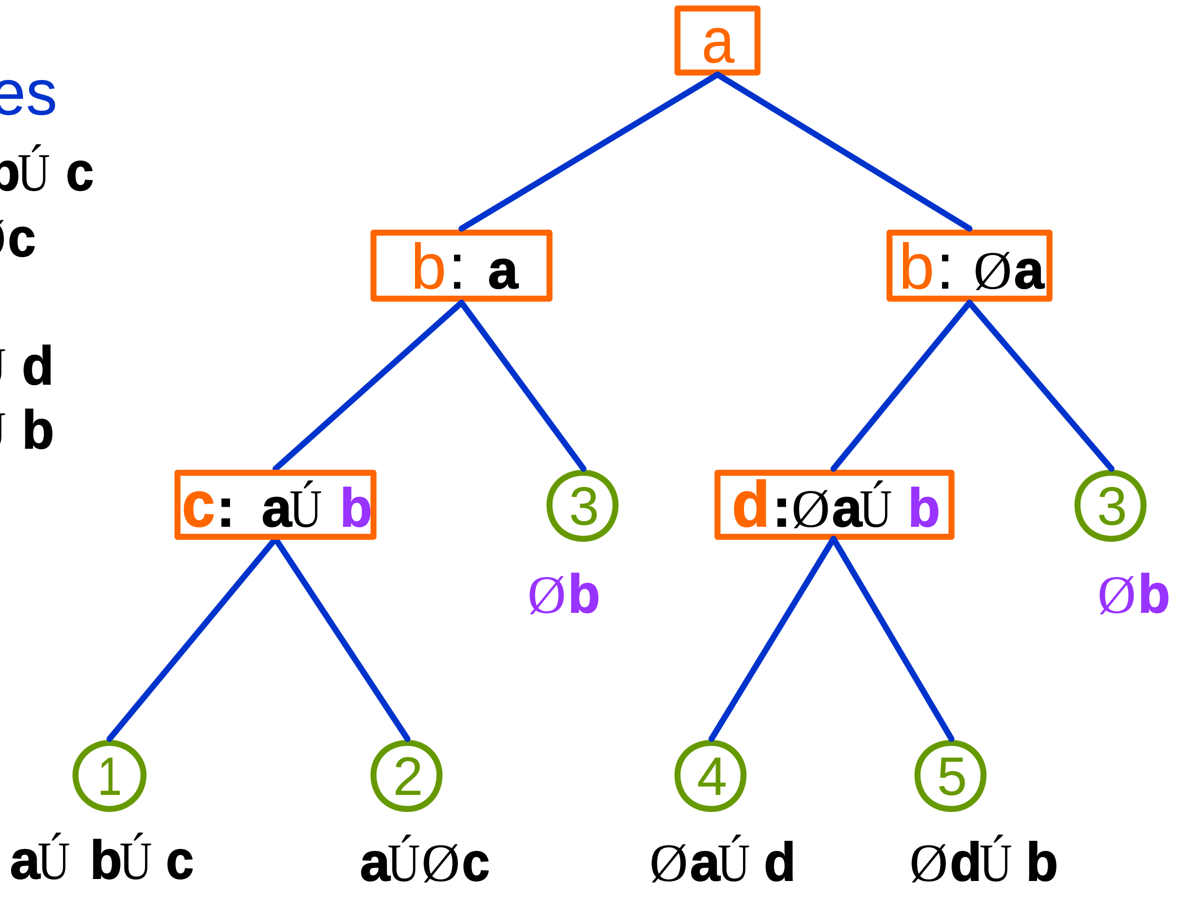
1. **aÚ bÚ c**
2. **aÚØc**
3. **Øb**
4. **ØaÚ d**
5. **ØaÚ b**



Tree Resolution

Clauses

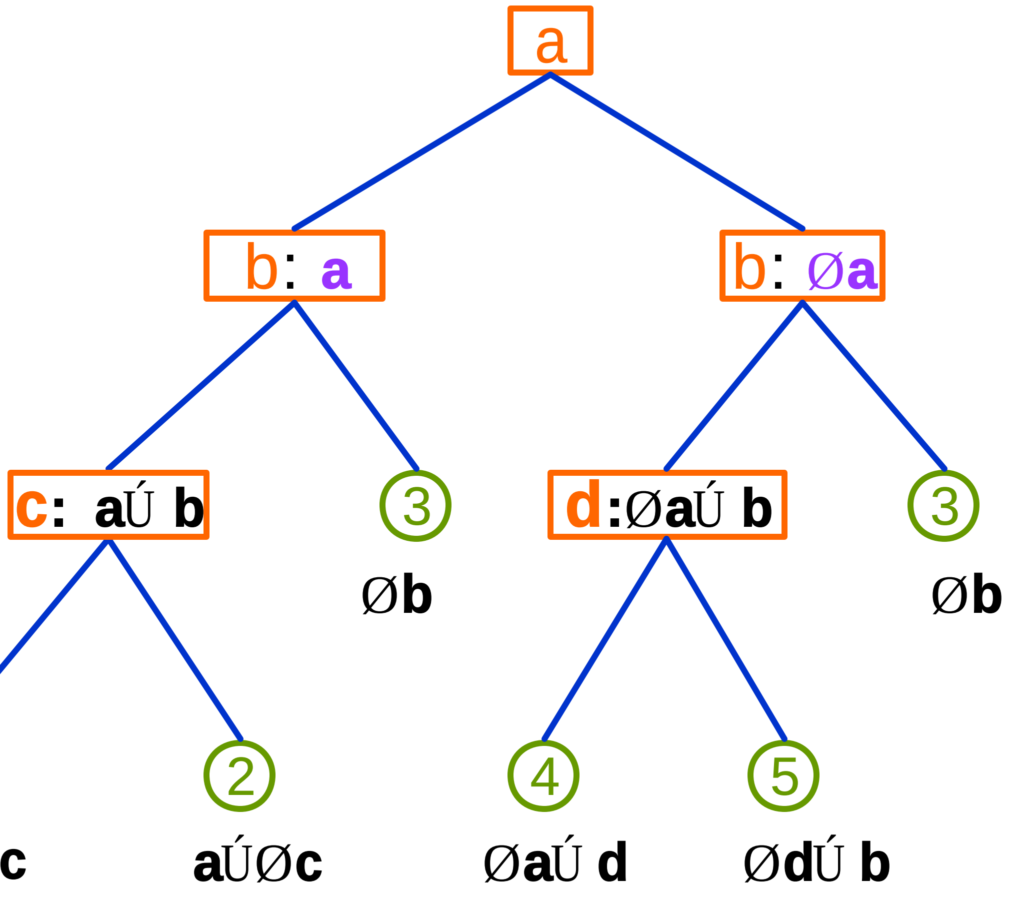
1. **a**Ú **b**Ú **c**
2. **a**ÚØ**c**
3. Ø**b**
4. Ø**a**Ú **d**
5. Ø**a**Ú **b**



Tree Resolution

Clauses

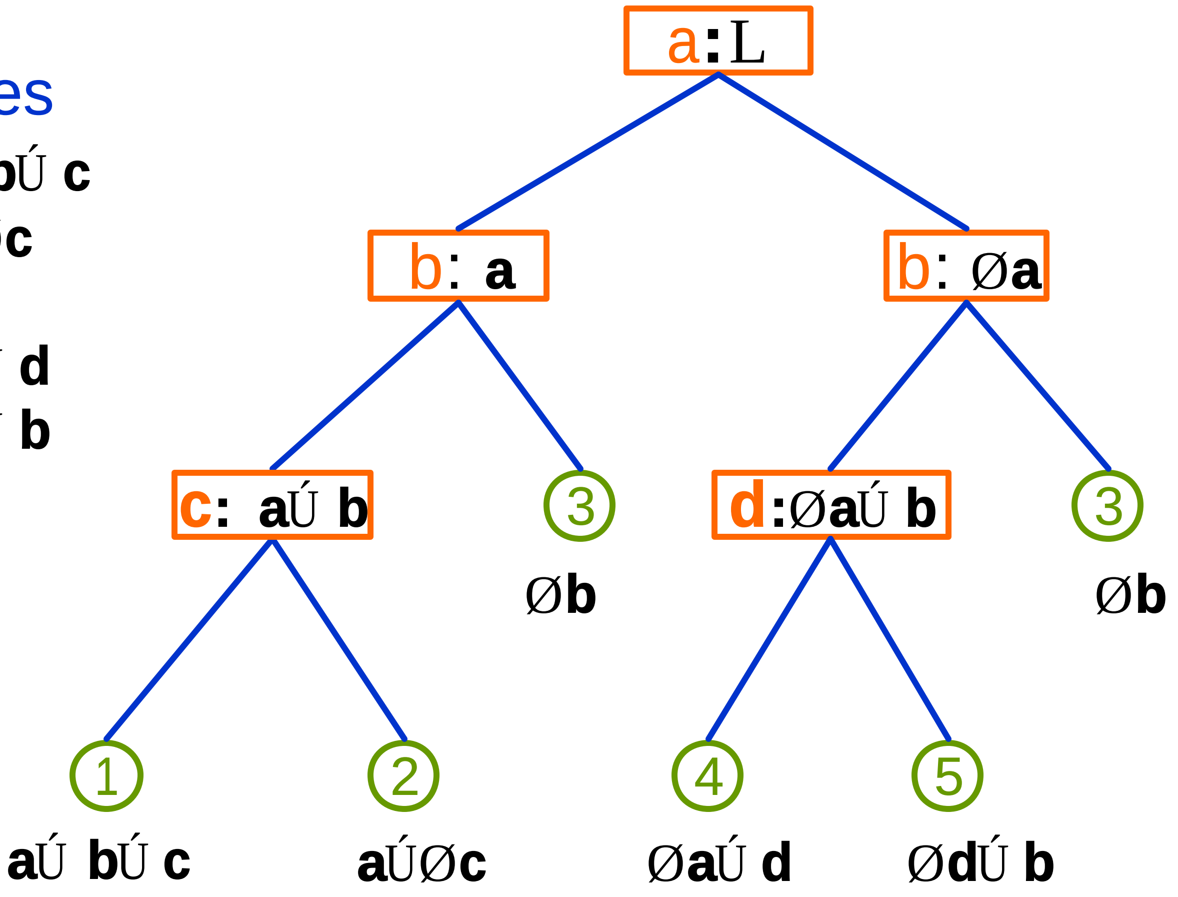
1. **aÚ bÚ c**
2. **aÚØc**
3. **Øb**
4. **ØaÚ d**
5. **ØaÚ b**



Tree Resolution Proof

Clauses

1. **aÚ bÚ c**
2. **aÚØc**
3. **Øb**
4. **ØaÚ d**
5. **ØaÚ b**



Hilbert's Nullstellensatz

- System of polynomials

  $Q_1(x_1,\dots,x_n)=0,\dots,Q_m(x_1,\dots,x_n)=0$

  over field **K** has **no** solution in any extension field of **K**

  Û

  there exist polynomials

  $P_1(x_1,\dots,x_n),\dots,P_m(x_1,\dots,x_n)$ in $K[x_1,\dots,x_n]$ s.t.

$$\sum_{i=1}^{m} P_i Q_i \equiv 1$$

Nullstellensatz proof system

- Clause $\overbrace{(x_1\text{Ú } \text{Ø}x_2 \text{ Ú } x_3)}^{C}$ becomes equation $\underbrace{(1-x_1)x_2(1-x_3)}_{Q_C}=0$

- Add equations $x_i^2 - x_i = 0$ for each variable
  - Guarantees only 0-1 solutions

- A **proof** is polynomials $P_1,\dots,P_{m+n}$ proving unsatisfiability: i.e. such that

$$\sum_{j=1}^{m} P_j Q_{C_j} + \sum_{i=1}^{n} P_{m+i}(x^2 - x) \equiv 1$$

Polynomial Calculus

- Similar to Nullstellensatz except:
  - Begin with $Q_1, \ldots, Q_{m+n}$ as before
  - Given polynomials **R** and **S** can infer
    - $a \bullet R + b \bullet S$ for any a, b in K
    - $x_i \bullet R$
  - Derive constant polynomial **1**
  - Degree = maximum degree of polynomial appearing in the proof
  - Can find proof of **degree d** in time $n^{O(d)}$ using Groebner basis-like algorithm (linear algebra)

- Special case of **$AC^0[p]$-Frege** if **$K=GF(p)$** (depth 1)

Exercise

- Show that every unsatisfiable formula has a proof of degree at most **n+1** for Nullstellensatz/Polynomial Calculus

Cutting Planes

- Introduced to relate integer and linear programming [Gomory 59, Chvatal 73]:
  - Objects are linear integer inequalities
  - Clause **(x_1Ú Øx_2 Ú x_3)** becomes inequality
    x_1+(1-x_2)+x_3 ³ 1
  - Add inequalities **x_i ³ 0** and **1-x_i ³ 0**
- Goal: derive **0 ³ 1**
- Special case of **TC^0-Frege** (depth 1)

Cutting Planes rules

- addition:

$$\frac{\begin{array}{c} a_1x_1 + \ldots \;\; + a_nx_n \;\, {}^{3} \;\, A \\ b_1x_1 + \ldots \;\; + b_nx_n \;\, {}^{3} \;\, B \end{array}}{(a_1+b_1)x_1+\ldots+(a_n+b_n)x_n \;\, {}^{3} \;\, A+B}$$

- multiplication by positive integer:

$$\frac{a_1x_1 + \ldots \;\; + a_nx_n \;\, {}^{3} \;\, A}{ca_1x_1 + \ldots + ca_nx_n \;\, {}^{3} \;\, cA}$$

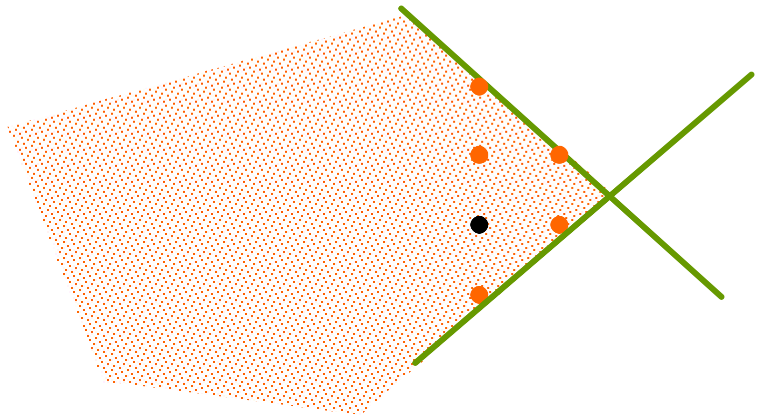
- **Division** by positive integer:

$$\frac{ca_1x + \ldots + ca_nx_n \;\, {}^{3} \;\, B}{a_1x_1 + \ldots + a_nx_n \;\, {}^{3} \;\, \text{é}B/c\text{ù}}$$

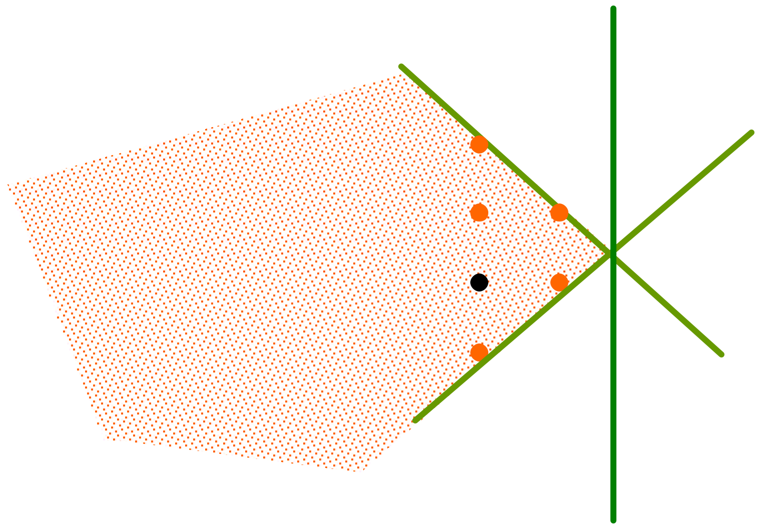
Why is it called cutting planes?

$-x-y \;{}^{3}\; -2$

$-x+y \;{}^{3}\; -1$



Why is it called cutting planes?

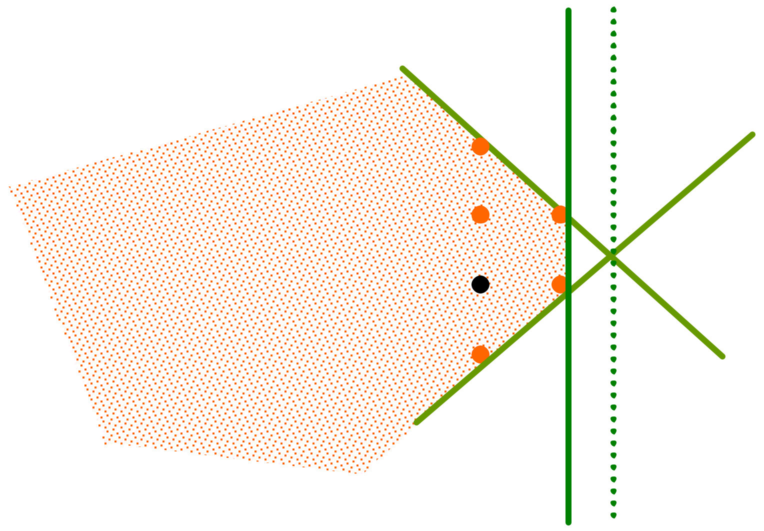


$-x-y$ 3 -2

$-x+y$ 3 -1

$-2x$ 3 -3

Why is it called cutting planes?



$-x-y \;\; 3 \;\; -2$

$-x+y \;\; 3 \;\; -1$

$-2x \;\; 3 \;\; -3$

$-x \;\; 3 \;\; -1$

Cutting Planes p-simulates Resolution

Resolution

$$\frac{\textbf{(aÚbÚcÚØd)}\quad\textbf{(ØaÚbÚcÚØf)}}{\textbf{(bÚcÚØdÚØf)}}$$

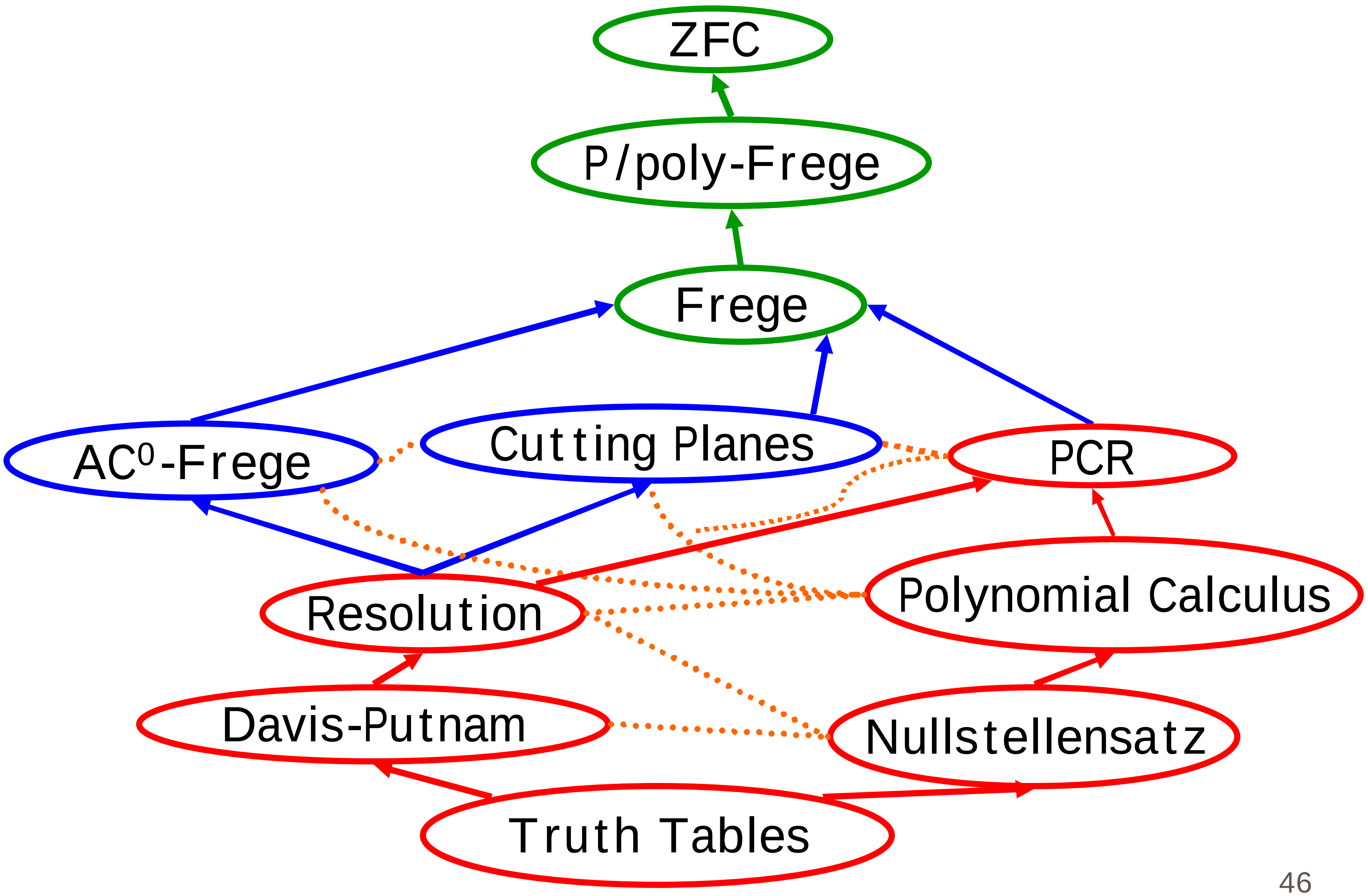
Cutting Planes

$$\begin{array}{r}
a + b + c + (1-d) \;^3\; 1 \\
(1-a) + b + c + (1-f) \;^3\; 1 \\
(1-d) \;^3\; 0 \\
(1-f) \;^3\; 0 \\
\hline
2b + 2c + 2(1-d) + 2(1-f) \;^3\; 1 \\
\hline
b + c + (1-d) + (1-f) \;^3\; 1
\end{array}$$

Addition

Division

Some Proof System Relationships



How high is the hierarchy?

- **Defn:** Proof system **U** **p-dominates** proof system **V** **iff** there is polynomial **f:N→N** s.t.
  $P.V accepts (x,P) Û $P'.|P'|£f(|P|). U accepts (x,P')
- **Defn: U** is **super** iff **U** p-dominates all other propositional proof systems, **U** is **super-duper** iff it p-simulates all such systems.
- **Thm:** [Krajicek-Pudlak 89]
  - **EXP**=**NEXP** **implies** super-duper proof systems exist
  - **NEXP=coNEXP implies** super proof systems exist

Why all these proof systems?

- **Proof systems formalize different types of reasoning**
- **Why even include the weaker systems within a given type of reasoning?**
  - many weaker proof systems have better associated proof search strategies, e.g. Davis-Putnam, Nullstellensatz, Polynomial Calculus.
- **Natural correspondence with circuit complexity classes**
  - analyze systems working upwards in proof strength to gain insight for techniques

Sources

- [Cook, Reckhow 79]
- [Urquhart 95]
- [Beame, Impagliazzo, Krajicek, Pitassi, Pudlak 94]
- [Clegg, Edmonds, Impagliazzo 96]
- [Krajicek, Pudlak 89]

Homework

- Show that every unsatisfiable formula has a proof of degree at most **n+1** for Nullstellensatz/Polynomial Calculus

- Show that resolution may be simulated by sequent calculus where we start with one sequent per clause and all cuts are on literals

- Show that every formula may be rebalanced to an equivalent one of logarithmic depth
  - First find a node in the formula that has constant fraction of the nodes in its subtree

Tableaux/Model Elimination systems

- search through sub-formulas of input formula that might be true simultaneously
- e.g. if Ø**(A** $\rightarrow$ **B)** is true **A** must be true and **B** must be false
- build a tree of possible models based on subformulas
- equivalent to sequent calculus without the cut rule
- In worst case is worse than truth tables (**n!**)