Resolution

History

Resolution is a proof system for DNF formulas or a refutational Systems for CNF formulas

It was introduced by Blake in 1937 and then became important by a work Davis-Putnam and Robinson in the 60s in the field of automated theorem Proving.

In the last 20 years it was subject of deep investigations in the field of Proof Complexity. There are hundreds of papers with subject Resolution.

At present it is still matter of strong investigations

Plan

- Definition of the Resolution system
- Soundness and Completeness
- Examples
- Restrictions and Refinements of Resolution
- Complexity measures for Resolution
- Interpolation for Resolution
- Search Problems and Resolution.
- DPLL algorithm and Treelike Resolution

Definitions

Resolution rule

Clauses are disjunctions of literals (x1∨x2∨¬x3).

CNF ar conjunctions of clauses

Resolution Rule

$$\frac{C \vee x_i \qquad D \vee \neg x_i}{C \vee D}$$

We can assume that both x_i and $\neg x_i$ do not occur in C and D.

x_i is the resolved variables

Assignments

An assignment satisfies a clause if satisfies at least one of its literals

Property [Exercise 1]

Resolution rule is sound: if an assignment satisfies the premises of the rule then it satisfies the conclusion

Resolution refutations

Let F be a CNF, F =C1,...,Cm.

A Resolution proof P of a clauses C from F $F \xrightarrow[P]{Res} C$

is a sequence of

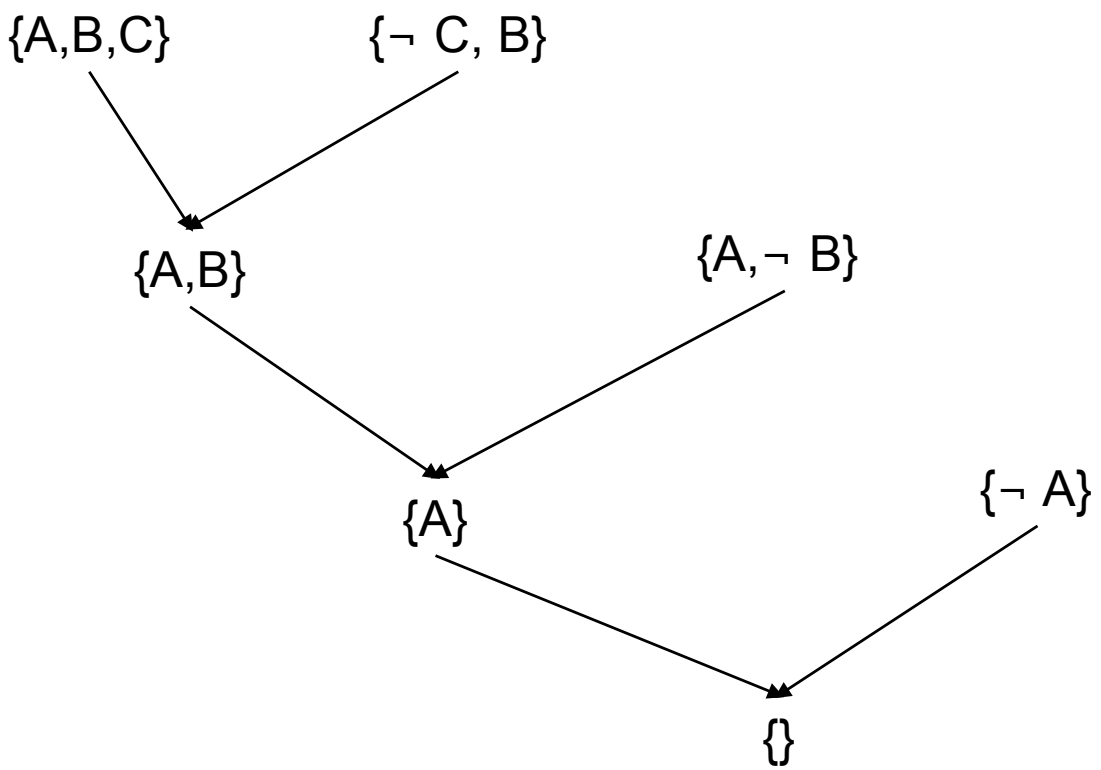
Clauses D1,.....,Dl s.t.

1. Dl =C
2. Di is either one of the Ci`s or is inferred by Resolution rule from two previous clauses Dj and Dk, j,k<i in the sequence

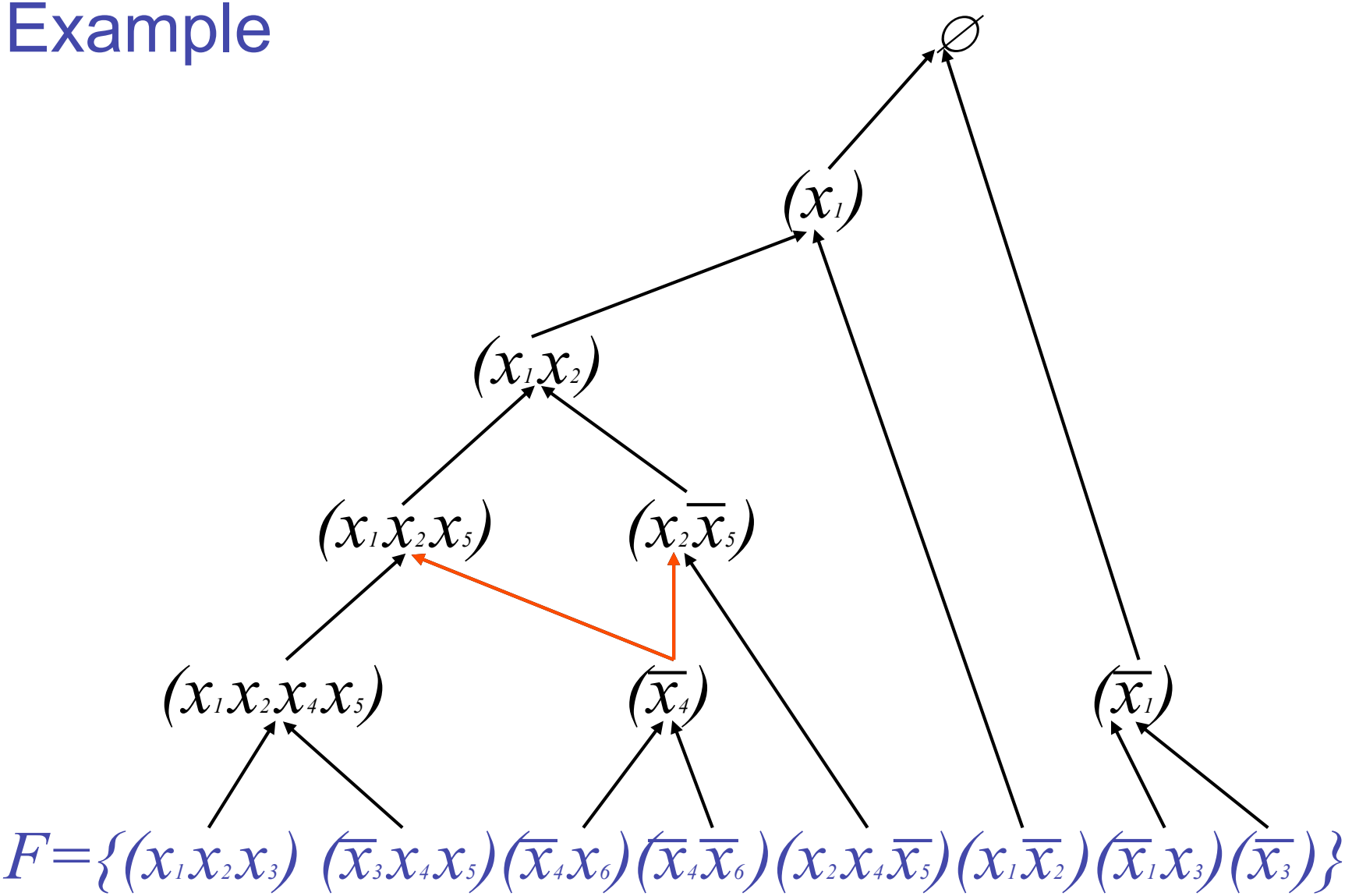
If C =[] the empty clause , we speak of Refutation of F

Examples

F = {A,B,C},{¬ C, B} {A,¬ B} {¬ A}



Example

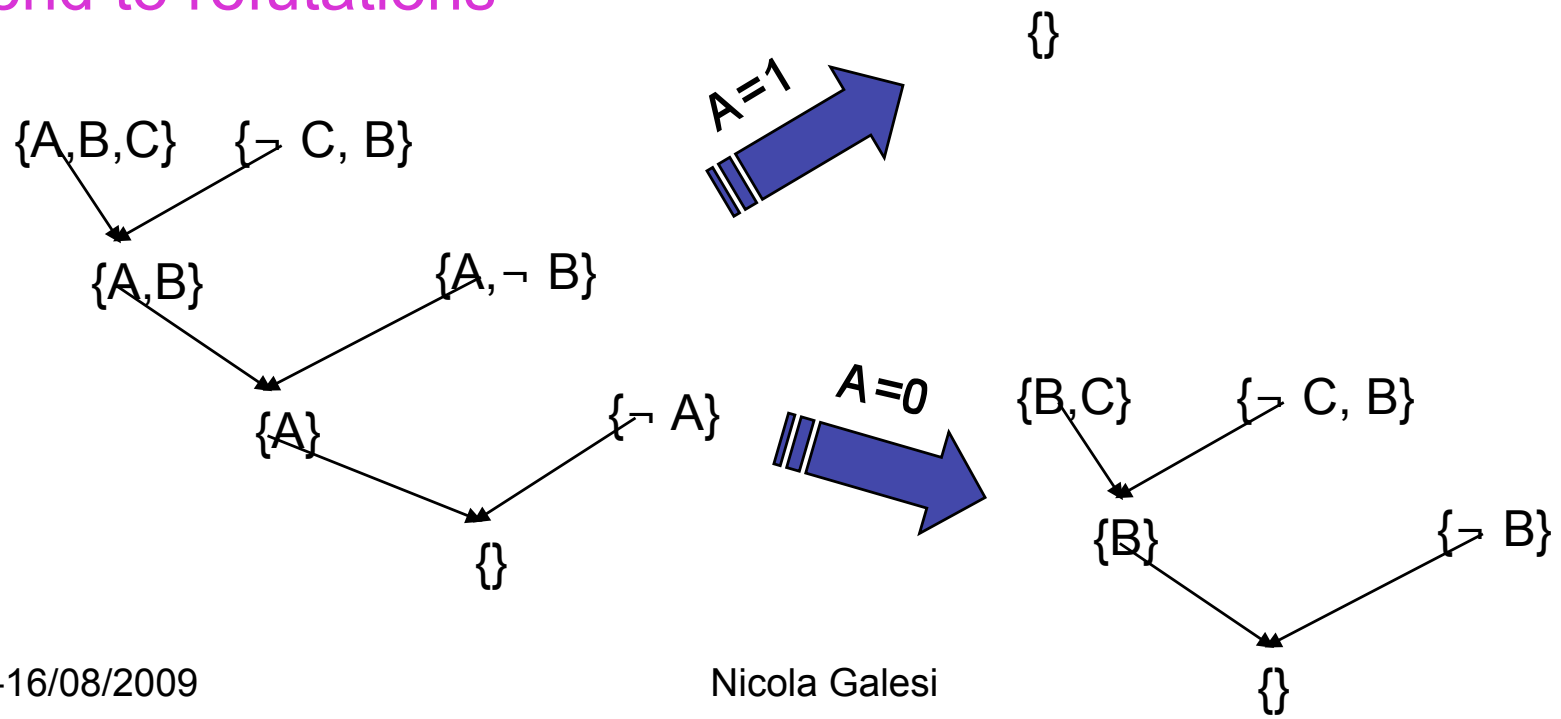

∅
(x_1)
(x_1x_2)
$(x_1x_2x_5)$
$(x_2\overline{x}_5)$
$(x_1x_2x_4x_5)$
$(\overline{x}_4)$
$(\overline{x}_1)$
$F=\{(x_1x_2x_3)\ (\overline{x}_3x_4x_5)(\overline{x}_4x_6)(\overline{x}_4\overline{x}_6)(x_2x_4\overline{x}_5)(x_1\overline{x}_2)(\overline{x}_1x_3)(\overline{x}_3)\}$

Assignments and refutations

Let C be a clause and α a partial assignment to variables of C. C[α] acts as follows:

- if some variables of C is set to 1 then C =1
- All variables set to 0 are deleted from C

Extend to refutations



A method for UNSAT

Resolution is a method to prove unsatisfiability of formulas in CNF. If a proof of F in Resolution ends with the [] then F is UNSAT.

Soundness

If $F \xrightarrow[P]{Res} []$ then F is UNSAT.

Assume by the contrary that F is SAT. Then there exists an assignment which satisfies the whole proof, in particular [].

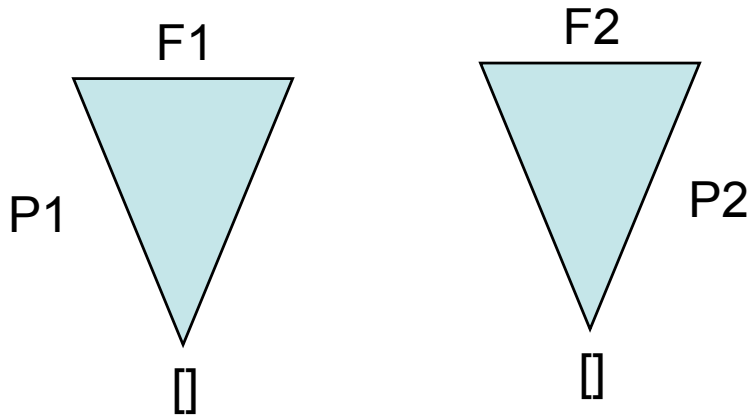
A method for UNSAT

Completeness. Induction on n = # of variables of F

n=0. Then F =[]. Ok

n→ n+1. Choose a var x in F

F1=F[x=1] and F2=F[x=0]. F1 and F2 are UNSAT , then by HI



A method for UNSAT

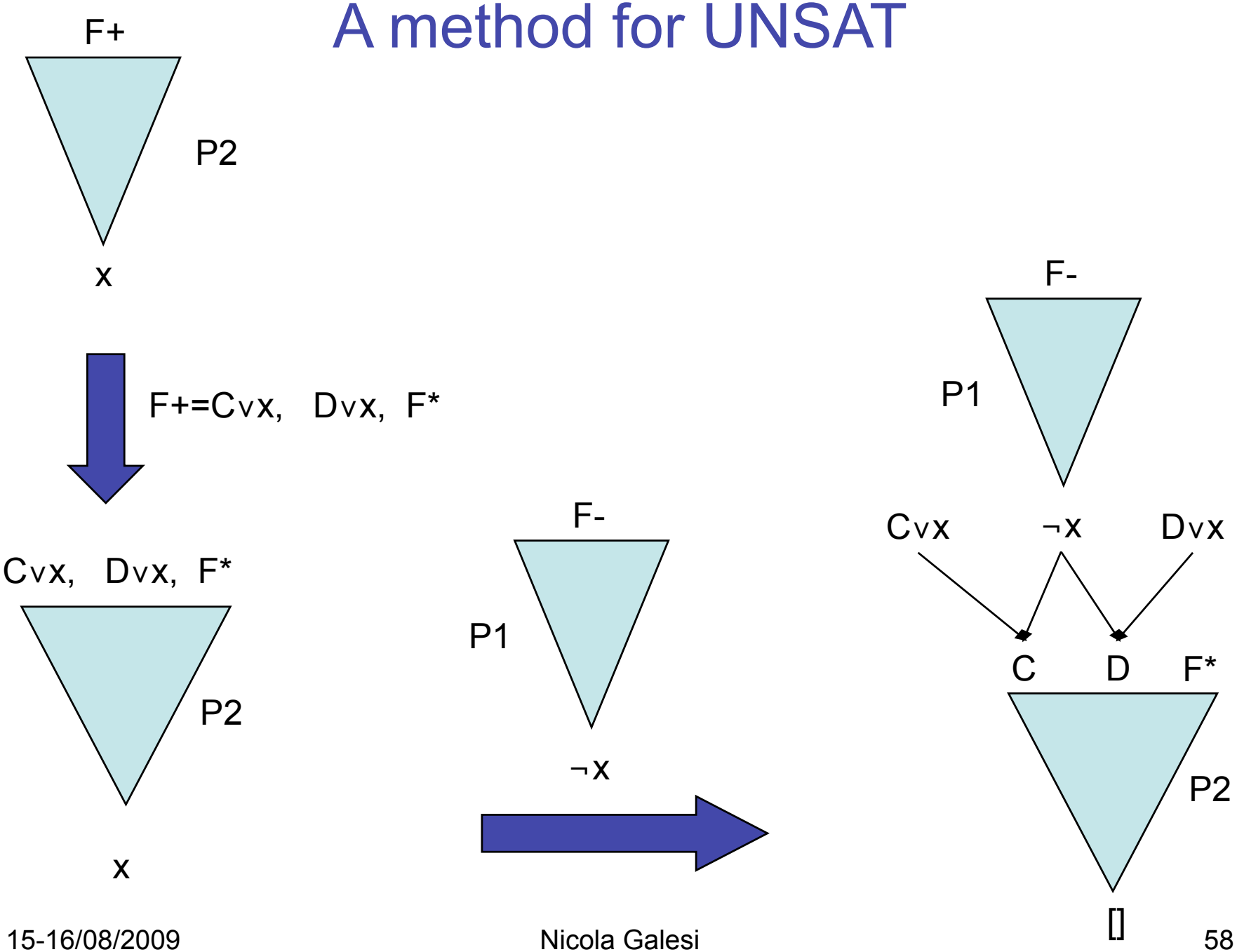
F+ obtained from F as follows

- keep all clauses containing x
- Delete all clauses containing ¬x
- Keep all other clauses

F- obtained from F as follows

- keep all clauses containing ¬x
- Delete all the clauses containing x
- Keep all other clauses

A method for UNSAT



Resolution as an algorithm

Assume S is a set of clauses.

Res(S)= S ∪ {C | C is obtained by Resolution from A, B ∈ S}

Define

$$\mathrm{Res}^0(S) = S$$

$$\mathrm{Res}^{n+1}(S) = \mathrm{Res}(\mathrm{Res}^n(S))$$

$$\mathrm{Res}^*(S) = \bigcup_{n \geq 0} \mathrm{Res}^n(S)$$

Thm. S is a set of clauses is UNSAT iff [] ∈ Res*(S).

[Exercise 2]

Resolution as an algorithm

An algorithm to test if a formula A is a TAUT

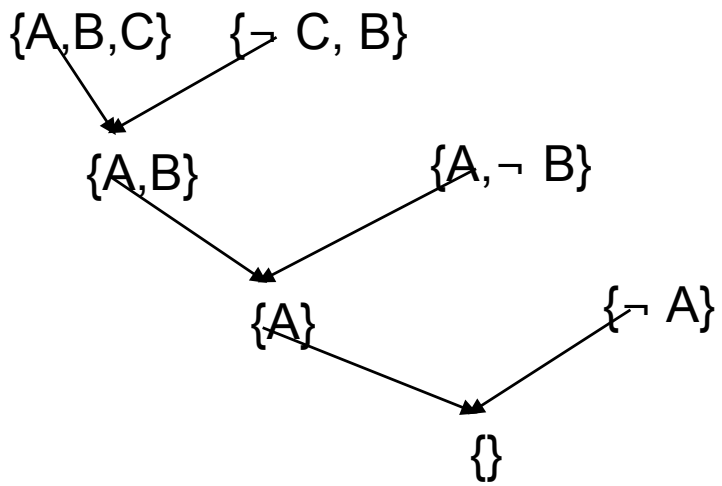
1. Take ¬A
2. Trasform ¬A in CNF formula S
3. Repeat
4. F=S
5. S= Res(S)
6. While ([] ∉ S or F=S)
7. Output([]∈S)

Refinements of Resolution

Treelike Resolution (TLR)

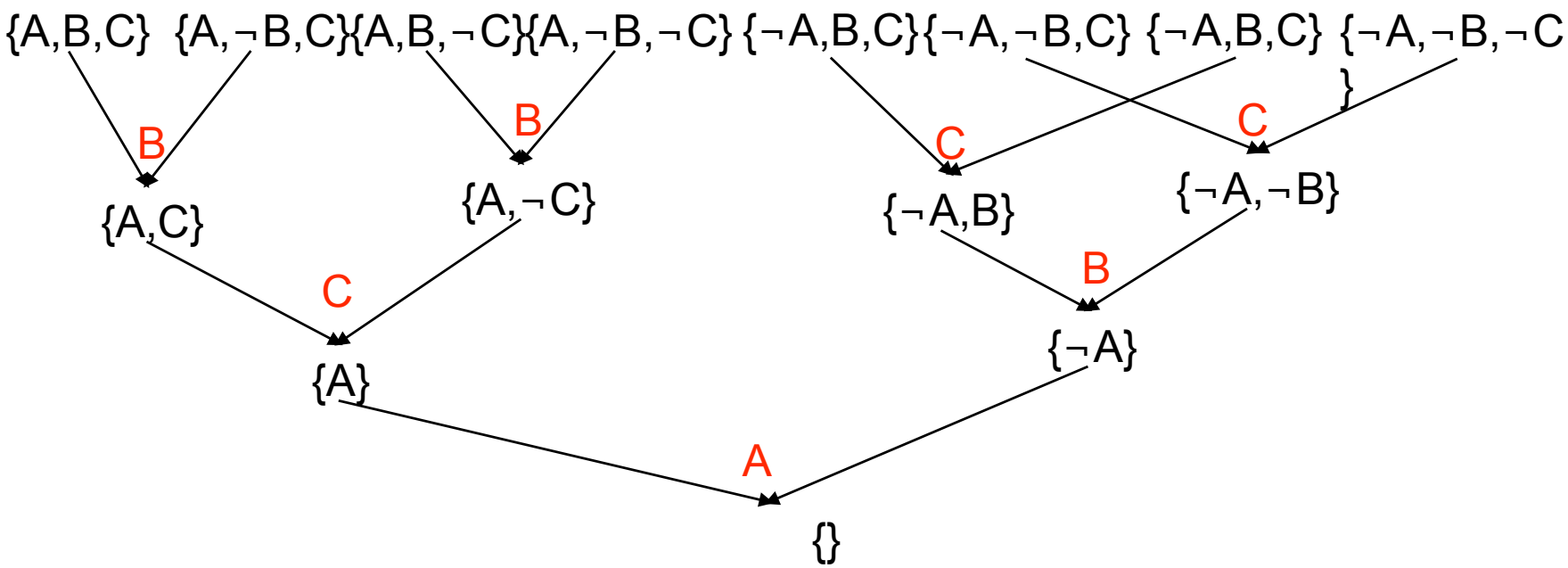
A Resolution refutation P is treelike if each clause in the proof is used at most once as a premise in a resolution rule

Said otherwise: a refutation is treelike if the proof graph is a tree.



Regular Resolution

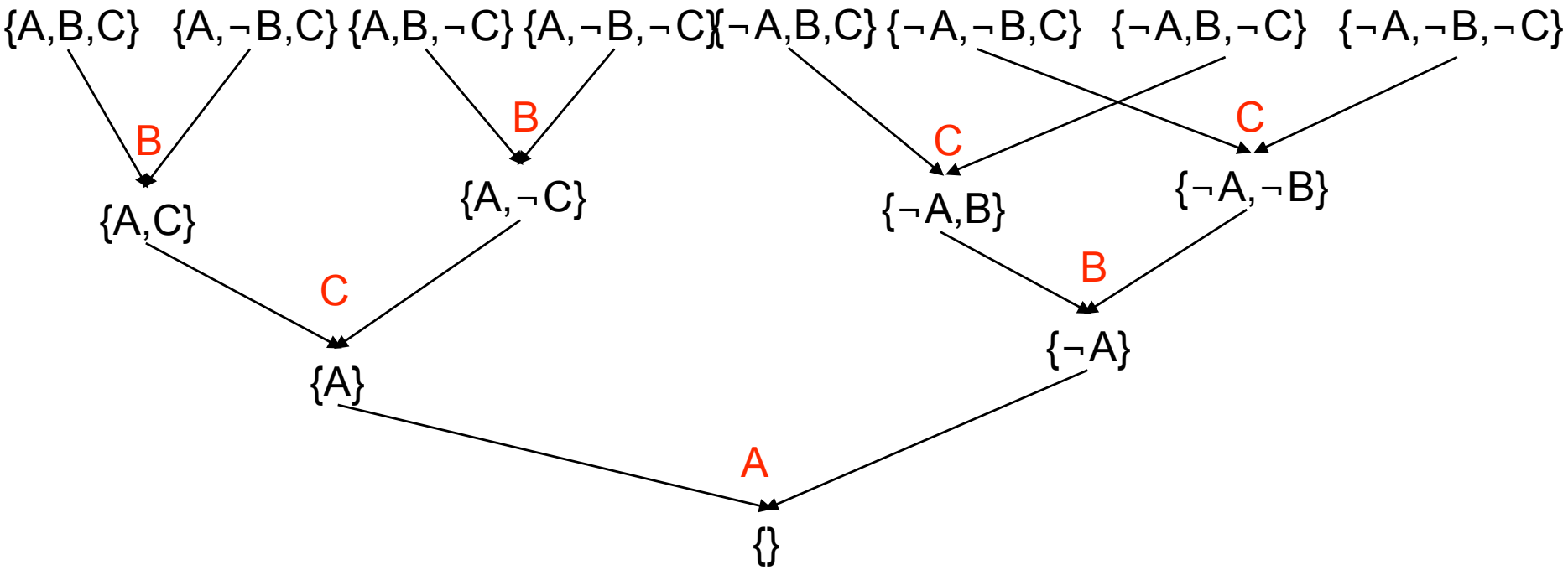
A Resolution refutation P is regular if along all paths from the empty clause to a leaf, each variable is resolved at most once.



Regularity important on daglike proofs

Ordered Resolution

A Resolution refutation P of is Ordered if ther is an elimianation order of the variables which is respected along all Paths.

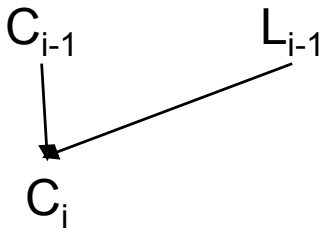


Ordered is a case of Regular [Trasform the proof in the example in ordered].

Ordered Resolution important on daglike proofs

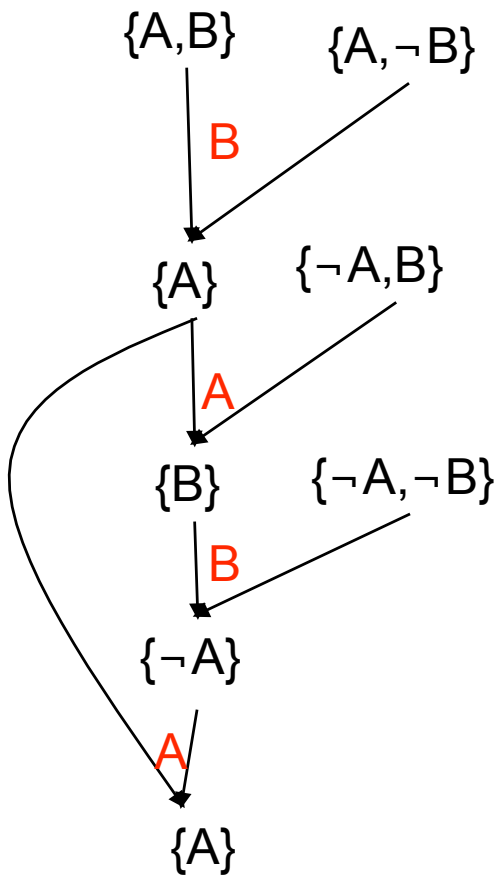
Linear Resolution

A Linear Resolution refutation of a formula F=F1,...,Fr is a sequence C1,...,Cm. s.t. C1 $\in$ F, Cm=[] and each step is of the form



Where L_{i-1} is either a clause of F or is C_j for j<i

Example



Complexity measures For Resolution

Size

Let P be a refutation C1,C2,....,Cm=[] of a CNF F = F1,...,Fr .

The size of P is m, i.e. the number of clauses in the proof or equivalently the number of nodes in the proof graph.

Given a CNF formula F

Size of refuting F in X-Resolution (where X= daglike, treelike, Regular, ecc)

$S_X(F)$= min{|P| : P is a X-Resolution refutations of F}

Notice for the same F it could be that $S_{TLR}(F) >= exp(|F|^{\varepsilon})$ but $S_{DLR}(F) <= |F|^{O(1)}$

Resolution Space

Memory configuration: A set of clauses M

Refutation: $P=M_0, M_1, \ldots, M_k$

* M_0 is empty
* M_k contains the empty clause.
* M_{t+1} is obtained from M_t by:

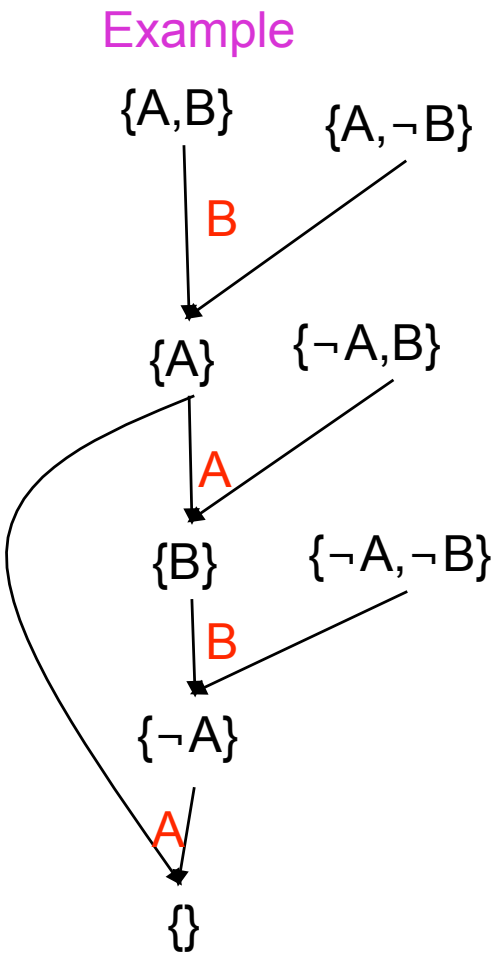
1. Axiom Download: $M_{t+1} = M_t + C \in F$.
2. Inference step: $M_{t+1} = M_t$ + some C derived by resolution from a pair of clauses in M_t.
3. Memory Erasure: M_{t+1} is a subset of M_t.

$Sp(P) = \max_{t\in[k]}\{|M_t|\}$.

$Sp_R(F) = \min\{Sp(P): P \text{ refutation of } F\}$.

Resolution Space: Example

Time		Memory		
0				
1	{A,B}			
2	{A,B}	{A,¬B}		
3	{A,B}	{A,¬B}	{A}	
4	{A,¬B}	{A}		
5	{A}			
6	{¬A,¬B}	{A}		
7	{¬A,¬B}	{A}	{B}	
8	{A}	{B}		
9	{A}	{B}	{¬A,¬B}	
10	{A}	{B}	{¬A,¬B}	{¬A}
11	{A}	{B}	{¬A}	
12	{A}	{B}	{¬A}	{}



Resolution width

C a clause. The width of C,

$$w(C) = \#\text{ literals in } C$$

F a CNF, the width of F

$$w(F) = \max\{w(C) : C \text{ a clause in } F\}$$

P a Refutation of a CNF F, the width of P

$$w(P) = \max\{w(C) : C \text{ a clause in } P\}$$

F UNSAT CNF. The width of refuting F in Resolution

$$w_R(F) = \min\{w(P) : P \text{ is a Resolution of } F\}$$

Relatioships between size and width

[BenSasson,Wigderson 99] Proved in Chapter II

Let F be a UNSAT k-CNF defined over n variables

Size-width tradeoffs for TLR

$$w_R(F) \leq \log(S_{TLR}(F)) + k$$

$$S_{TLR}(F) \geq 2^{(w_R(F)-k)}$$

Size-width tradeoffs for DLR

$$w_R(F) \leq O\left(\sqrt{n \log S_{DLR}(F)}\right) + k$$

$$S_{DLR}(F) \geq 2^{\Omega\left(\frac{(w_R(F)-k)^2}{n}\right)}$$

Relatioships between space and width

[Atserias-Dalmau 03] Proved in Chapter VI

Let F be a UNSAT k-CNF defined over n variables

Space width tradeoffs for Resolution

$$Sp_R(F) \geq w_R(F) - k + 1$$

Interpolation for Resolution

Interpolation and Complexity

Let A(**p**,**q**)→B(**p**,**r**) be a TAUT formula where **q**,**r** are sets of private varibales and **p** are commons to the two formulas.

An Interpolant C(**p**) is a formula such that

A(**p**,**q**) → C(**p**)

C(**p**) → B(**p**,**r**).

Interpolant and complexity

[Mundici 82] proved that if the formula size (circuit size) of the inteporlant is polynomial in the size of the implication then

NP ∩ co-NP ⊆ NC1/poly (resp NP ∩ co-NP ⊆P/poly)

Interpolation and Complexity

[Krajicek 94] Estimate the size of the circuit of the interpolant in terms of the length of the proof fo the implicant.

Let $A(\mathbf{p},\mathbf{q}) \wedge B(\mathbf{p},\mathbf{r})$ a UNSAT CNF

An Interpolant $C(\mathbf{p})$ is a circuit s.t.

$$C(\mathbf{a}) = \begin{cases} 0 & A(\mathbf{a},\mathbf{q}) \text{ UNSAT} \\ 1 & B(\mathbf{a},\mathbf{r}) \text{ UNSAT} \end{cases}$$

Interpolation and Complexity

Thm [Krajicek 94, Pudlak 96] proved in Chapter III

Let P be a DLR refutations of $A(\mathbf{p},\mathbf{q}) \wedge B(\mathbf{p},\mathbf{r})$. Then there exists a boolean circuit $C(\mathbf{p})$ s.t.

1. for every truth assignment **a** to the common variables p

$$C(\mathbf{a}) = \begin{cases} 0 & A(\mathbf{a},\mathbf{q}) \text{ UNSAT} \\ 1 & B(\mathbf{a},\mathbf{r}) \text{ UNSAT} \end{cases}$$

2. C is of size O(|P|) (#gates).
3. If the common variables **p** occur only positively in A and negatively in B, then C is monotone
4. If P is TLR, then C is a formula (treelike circuit)

Cor (informal)

Lower bounds on (monotone) circuit size give lower bounds on length of Resolution refutations

Search Problems And Resolution

Branching Progam

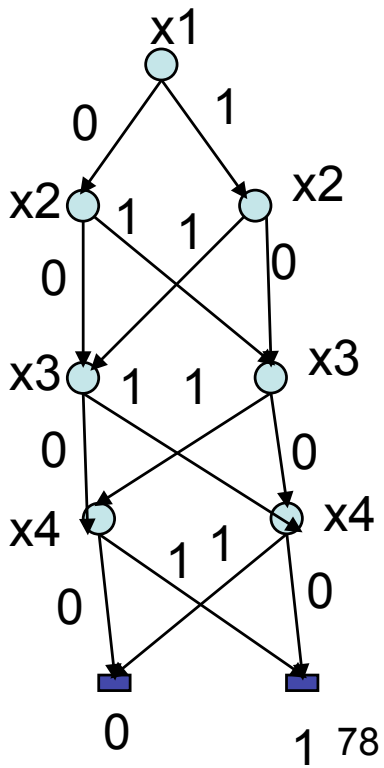
Branching Programs

A 2-regular dag with one source node and two target nodes

- Every node labelled with a variable
- The two edges leaving a node are labelled with element of a set X

A branching program computes a function $f:\{0,1\}^n \rightarrow X$ as follows:

f(a1,...,an)=x if the path starting at the root and following the edges according to **a** ends in x



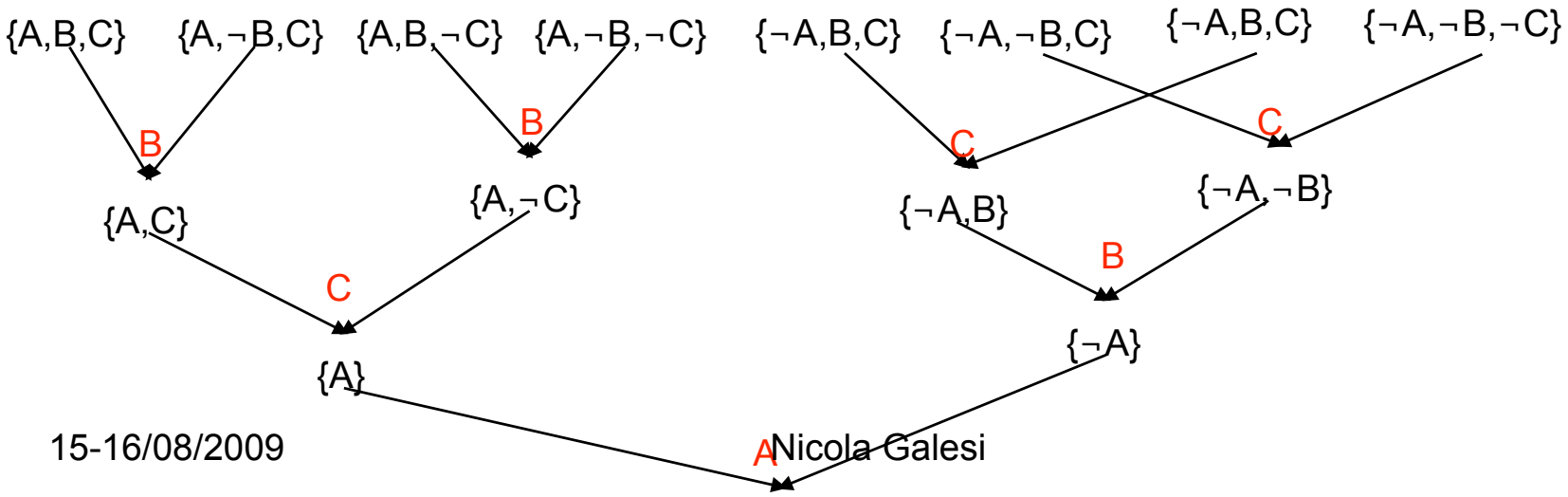
A False Clause Search Problem

Let F be a UNSAT CNF F= C1,....,Cm

Search Problem

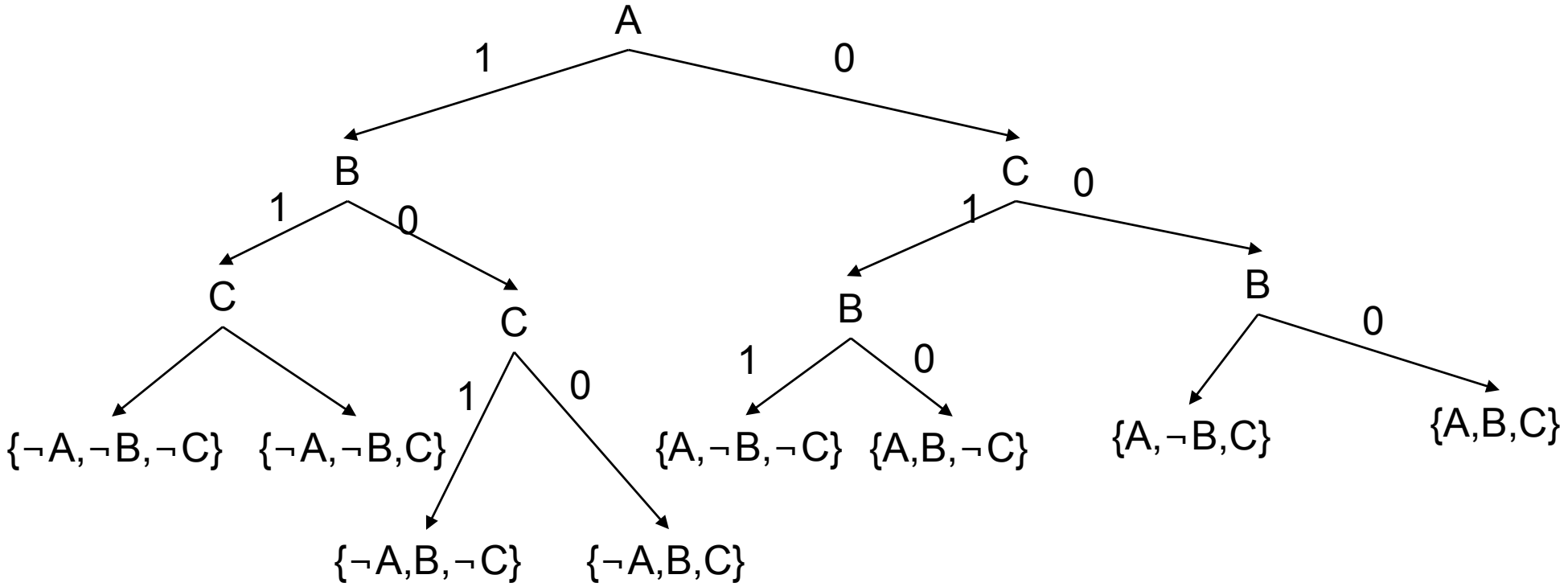
Given an assignment α to variables F find a clause C in F falsified under α

Refutation into decision trees



A False Clause Search Problem

Refutation into decision trees



Thm. A TLR refutation for F defines a decision tree that solves the FCLP for F. [Exercise 5. Make the statement and its proof precise]

Regular resolution define ordered branching programs to solve FCSP

DPLL and treelike Resolution

A SAT algorithm

Let F be a CNF. DPLL is an algorithm for the SAT of F

DPLL (F)

if F is empty

F is satisfiable and report the assignment

If F contains the empty clause

then return

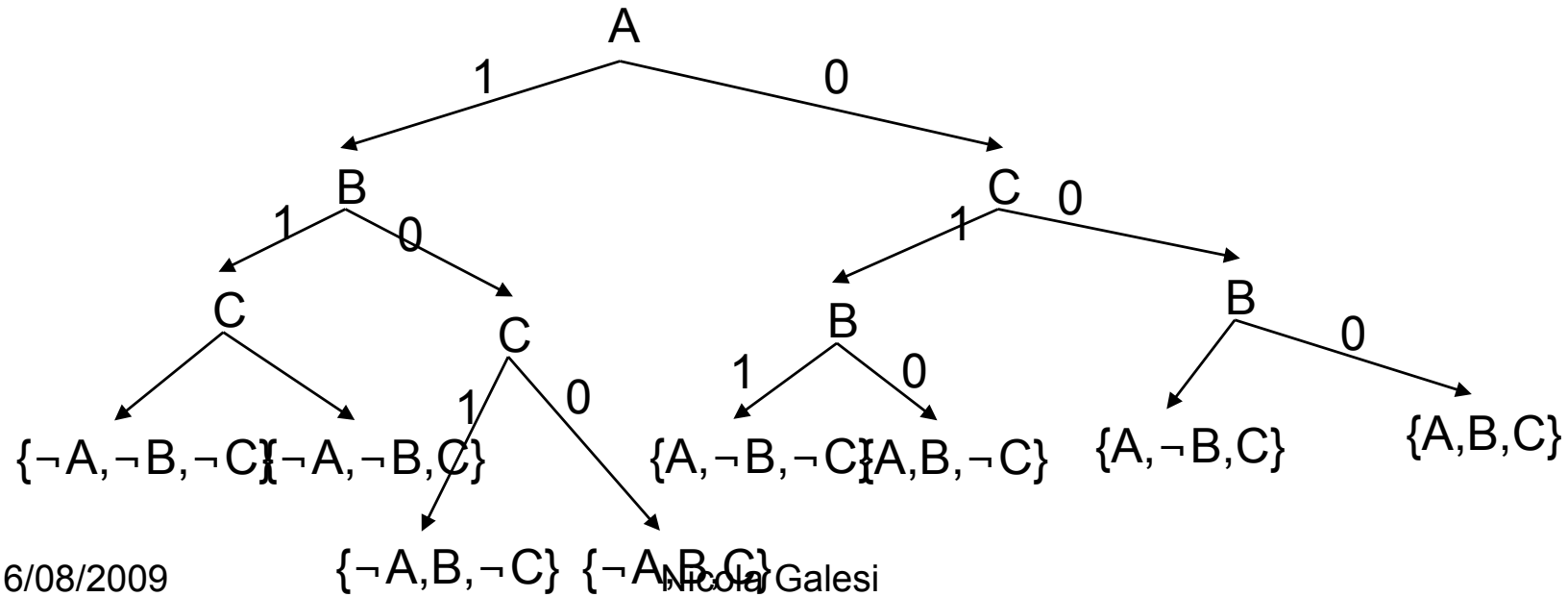
else

- choose a literal x
- DPLL(F[x=1])
    - DPLL(F[x=0])

Search tree on DPLL

Let F be a UNSAT CNF. DPLL is producing a Treelike Resolution refutation of F.

F= {A,B,C} {A,¬B,C} {A,B,¬C} {A,¬B,¬C}
{¬A,B,C} {¬A,¬B,C} {¬A,B,¬C} {¬A,¬B,¬C}



DPLL and Proof Complexity

DPLL vs TLR refutations.

DPLL is an algorithm to recover TLR refutations on UNSAT CNF formulas. This will have some consequences on Automatizabilty of TLR (Chapter III)

TLR Refutations vs DPLL performances

Since TLR is not polynomially bounded, i.e. There are families of formulas requiring exponential size TRL refutations, then DPPL performs bad on this formula.

DLR and Proof Search

Extending similar considerations to DLR and find good algorithms giving DLR refutations is matter of research

Exponential separations: Tree- vs Dag-like Resolution

Nicola Galesi
Dipartimento di Informatica
Università degli Studi di Roma "La Sapienza"

2009 August 15 – 16
NoNa Summer School
St Petersburg

History of Results

The Result.

A family of UNSAT formulae requiring exponential size TLR but admitting polynomial size DLR.

History

(1) [Goerdt 92] gave a first example of UNSAT formula with polynomial size DLR but requiring quasipolynomial TLR refutations (ad hoc modification of the PHP)

(2) [Bonet,Galesi,Esteban,Johannsen 98] Gave the first exponential separation. Use the Interpolation method together with a circuit lower bound for monotone real NC hierarchy.

(3) [Ben-Sasson, Impagliazzo,Wigderson03] Simplify and slightly improve the result for Resolution

Plan of the day

1. First we introduce a general setting to prove lower bound for tree-like Resolution: the **Prover-Delayer Game**

2. Then we introduce **Pebbling Games** on graphs and briefly discuss an important old result of [Celoni,Paul,Tarjan 77]

3. We build a family of UNSAT formulae **Peb(G)** encoding a principle related to pebbling of directed acyclic graphs G

4. We will show there are polynomial size DLR refutations for **Peb(G),** for any dag G.

5. Using the Prover Delayer game and the result of [CPT 77] we will prove that there exists a graph G s.t. Peb(G) requires exponential size proofs in TLR

Notions and Techniques

1. the **Prover-Delayer Game**

2. **Pebbling Games** and [CPT 77] result

3. Construction of **Peb(G)**

4. [BSIW03] proof

Prover Delayer Game

Prover Delayer Game

Definition.

In the PD game there are two players playing on a UNSAT k-CNF formula F

Prover: asks for variables x of F

Delayer: answers with a value for x or leaves it unset

and win 1$

Prover wins as soon as he falsifies a clause of F

Objective: Maximize the gain of Delayer

Prover Delayer Game

Example.

$$F =_{def} (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge$$
$$(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$$

I round

Prover: x1 ?
Delayer: unset
Gain: 1$
Prover x1=T

$$F[x1 = T] =_{def} (x_2 \vee x_3) \wedge (\neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)$$

II round

Prover: x2 ?
Delayer: unset
Gain: 2$
Prover x2=T

$$F[x_1 = T, x_2 = T] =_{def} (x_3) \wedge (\neg x_3)$$

III round

Prover: x3 ?
Delayer: unset
Gain: 3$
Prover x3=F → WIN

TLR proof size vs PD game

Idea.

"Good" strategies for the Delayer on a unsat CNF F, give "good" lower bounds for TLR refutations of F

Thm [Pudlak,Impagliazzo]

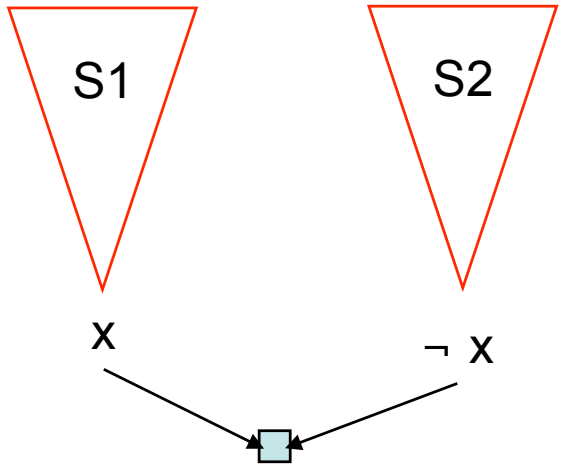
If F has TLR refutations of size S, then the Prover wins the PD game on F leaving O(log S) dollars to the Delayer

Cor

If in any PD game over F the Delayer wins at least p dollars, then the shortest TLR refutations of F are of size $2^{\Omega(p)}$.

Proof of the Theorem

Assume to have a size S TLR refutation P for F, |P| = S



Notice that both S1 and S2 cant be <= S/2. Say S1<=S/2.

For a generic round k, denote by

α_k = the assignment to variables of F built so far

p_k = the total number of dollars scored by the Delayer so far

Prover Rule

The Prover keeps the following invariant IP:

$$\left|P[\alpha_k]\right| \leq \frac{|P|}{2^{p_k}} = \frac{S}{2^{p_k}}$$

Assume the Prover is able to keep the invariant along the game.

Let f the final round. We want to calculate p_f, knowing that at the end $|P[\alpha_f]|=1$

$$1 \leq \frac{S}{2^{p_f}} \Rightarrow p_f = O(\log S)$$

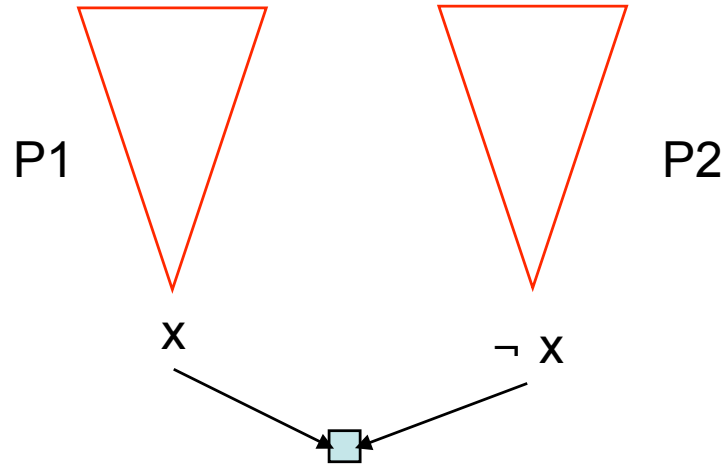
Keeping the Invariant - I

Base. At the beginning of the game:

$\alpha=\varnothing$, p=0; and the IP follows

Induction.

$P[\alpha_k] =$



Wlog $|P1| \leq \frac{|P[\alpha_k]|}{2}$

Prover chooses to present x

Keeping the Invariant- II

I Case. Delayer gives a value i ={0,1} to x.

Then $p_{k+1}=p_k$

$$\left|P[\alpha_{k+1}]\right| = \left|P[\alpha_k \cup x = i]\right| \overset{Contruction}{\leq} \left|P[\alpha_k]\right| \overset{Induction}{\leq} \frac{|P|}{2^{p_k}} \overset{p_k = p_{k+1}}{=} \frac{|P|}{2^{p_{k+1}}}$$

II Case. Delayer gets 1 $ but leaves Prover to choose. Prover gives to x the value to proceed into P1 (the smaller).

$p_{k+1}=1+p_k$, but $|P1| \leq \frac{|P[\alpha_k]|}{2}$

$$\left|P[\alpha_{k+1}]\right| = \left|P[\alpha_k \cup x = 0]\right| \leq \frac{\left|P[\alpha_k]\right|}{2} \overset{Induction}{\leq} \frac{|P|}{2*2^{p_k}} \overset{1+p_k = p_{k+1}}{=} \frac{|P|}{2^{p_{k+1}}}$$

Prover Delayer Game: Conclusions

Why PD Game ?

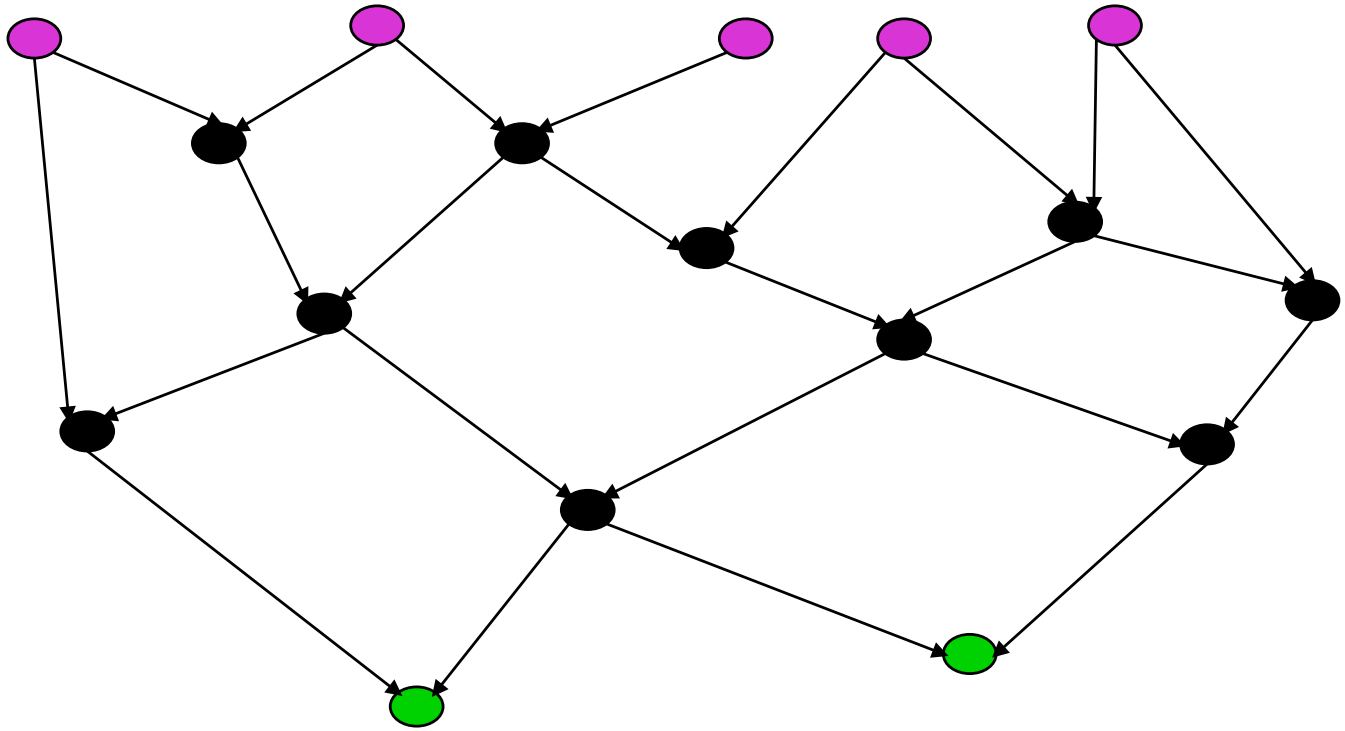
To prove an exponential lower bounds for TLR refutations of some UNSAT k-CNF F over n variables, is sufficient to find a strategy for the Delayer in the PD game over F that allow her to win $\Omega(n)$ dollars against any strategy of the Prover.

Not an easy task - after all

Since this means that to prove unsatisfiabilty of F the Prover needs to ask simultaneously almost all variables.

Pebbling Games

Pebbling Games on DAG



- Source Nodes
- Internal Nodes
- Target nodes
- Pebbles

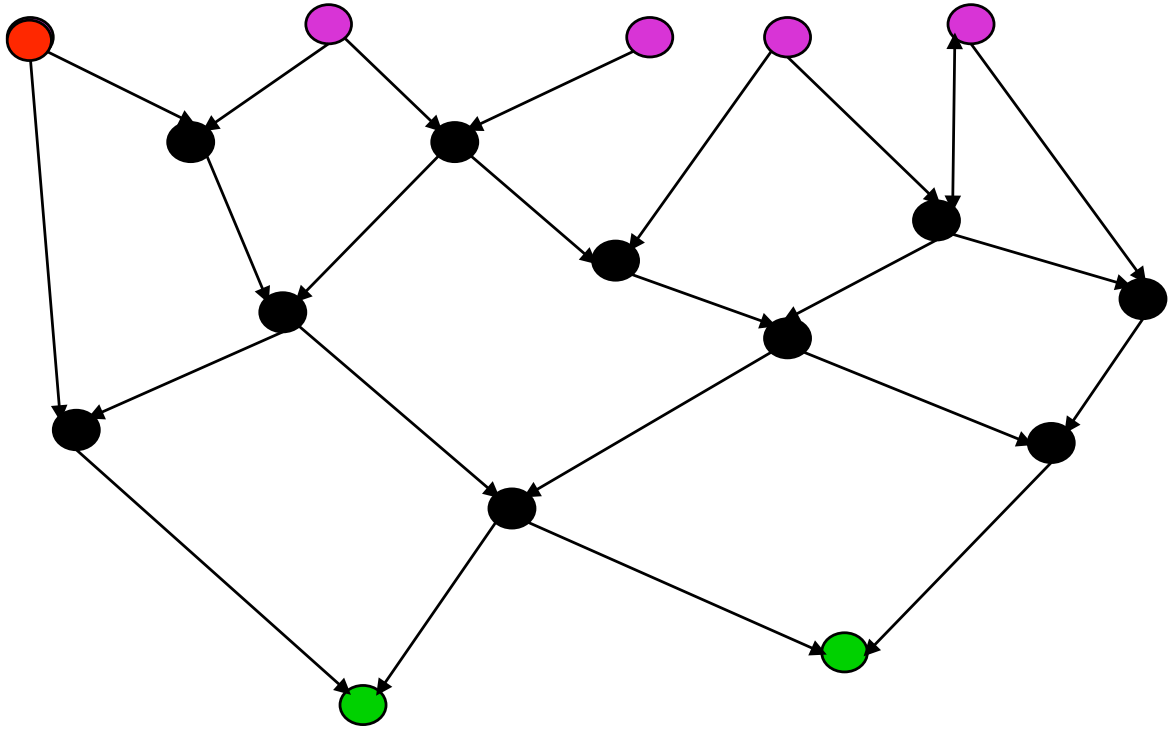
Pebbling Games: Rules

Game:

Place pebbles ● on dag's nodes according to the rules

Rule 1

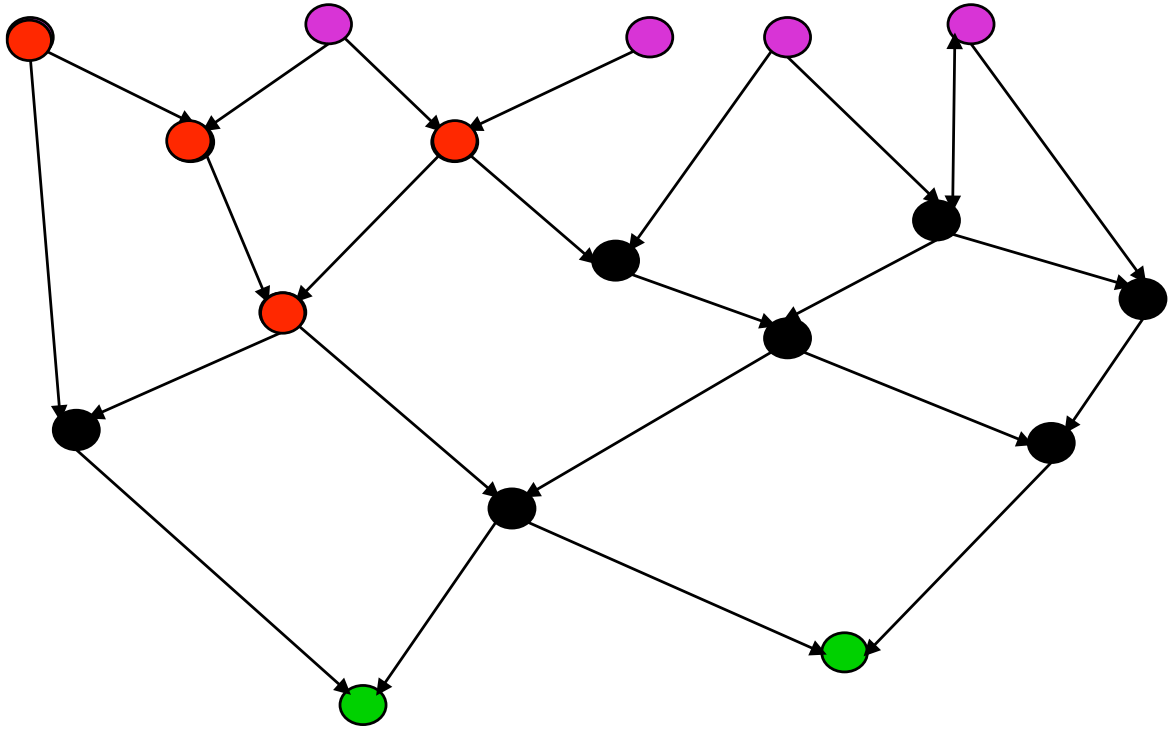
Sources nodes can be pebbled freely



Pebbling Games: Rules

Rule 2

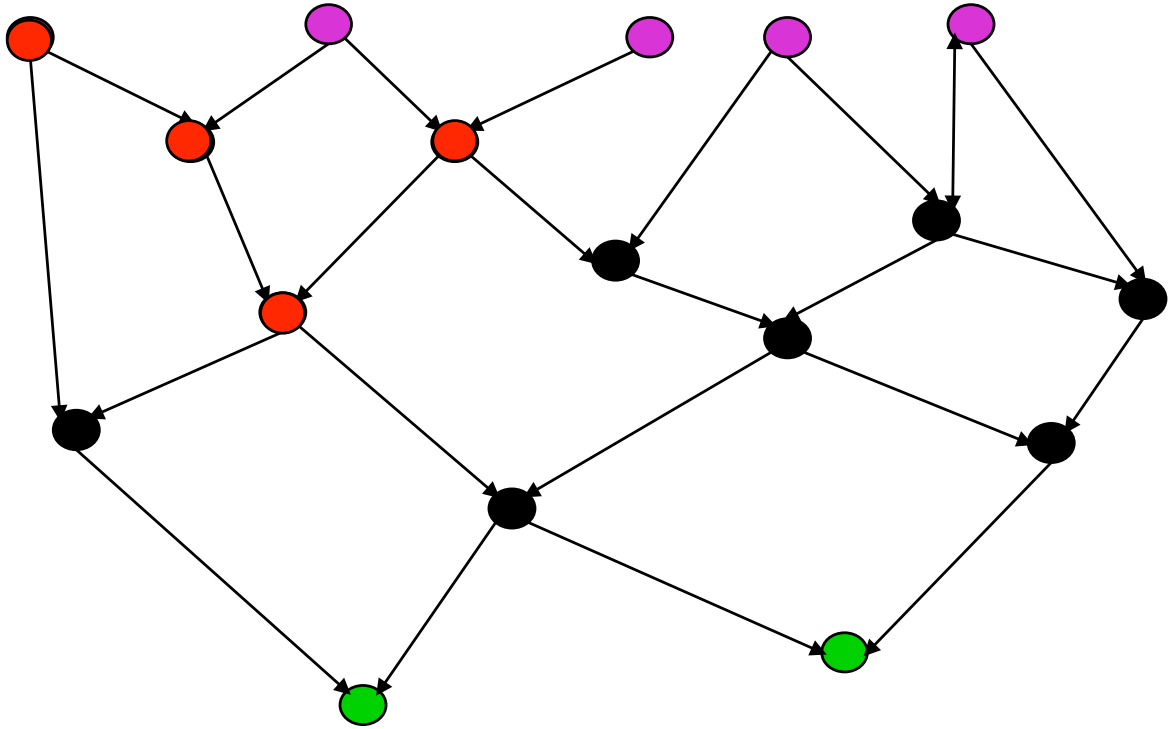
Internal nodes can pebbled only if their parents are both pebbled



Pebbling Games: Rules

Rule 3

Pebbles can be removed at any time



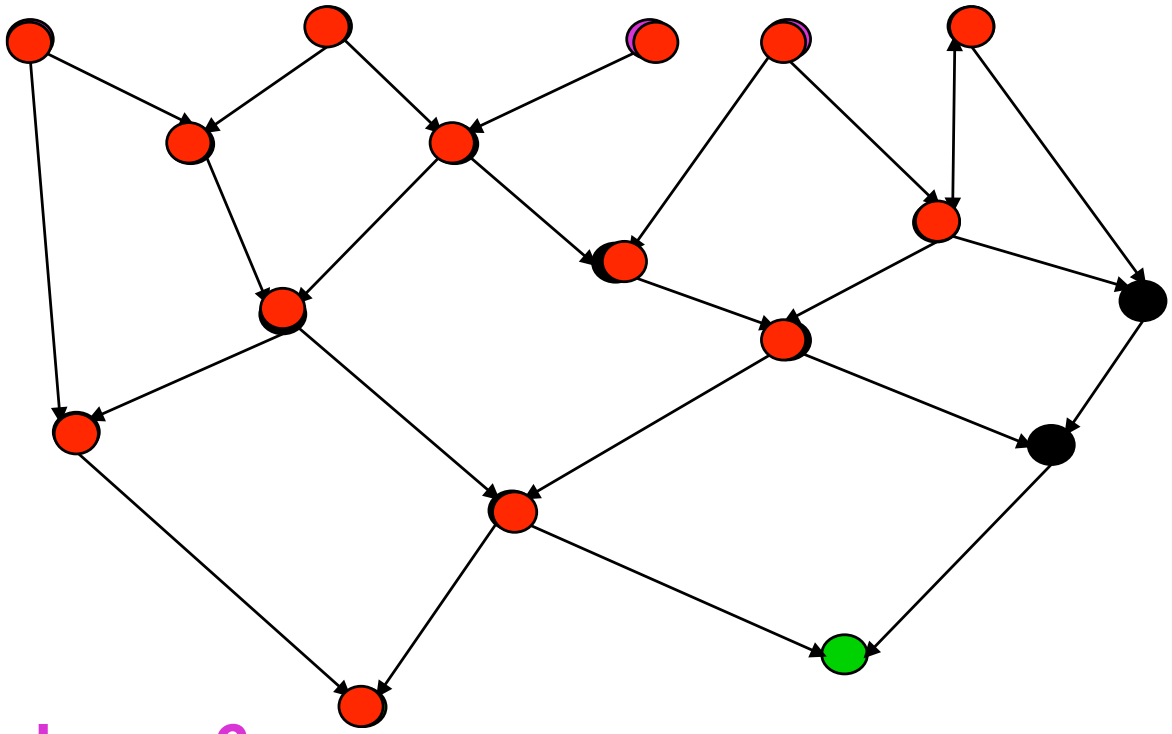
Pebbling Games: Aim & Complexity

Aim

Put a Pebble on some target node

Complexity Measure: Pebbling number

Maximal number of pebbles placed **simultaneously** on the graph.



Pebbling Number = 6

Definitions

Pebbling number of a strategy to pebble G

Let S be strategy to pebble a dag G.

Pn(G,S)= max # of simultaneous pebbles on G following S

Pebbling number of G

Pn(G) = min { pn(G,S) | S strategy to pebble G}

Hardness Results: prove that a graph G requires high pn(G)

Original Motivations: Prove space lower bounds for Turing Machine: pebbling game models space in TM

An Important Old Result

[Celoni, Paul, Tarjan 77]

Found (constructively) a directed acyclic graph G over n nodes with in-degree <= 2 such that

$$pn(G) \geq \Omega(n / \log n)$$

Pebbling Formulas

Modelling PG as UNSAT Formulas

Let G be a dag with indegree <=2. We encode the principle that the pebbling game must terminate sucessfully pebbling a target node. Put in an UNSAT formula we say that

1. Sources nodes can be always pebbled
2. Internal nodes can be pebbled whenever its parents are
3. But no target node is ever pebbled

Modelling PG as UNSAT Formulas

G=(V,E)

$v \in V$, x_v iff "node v has been pebbled"

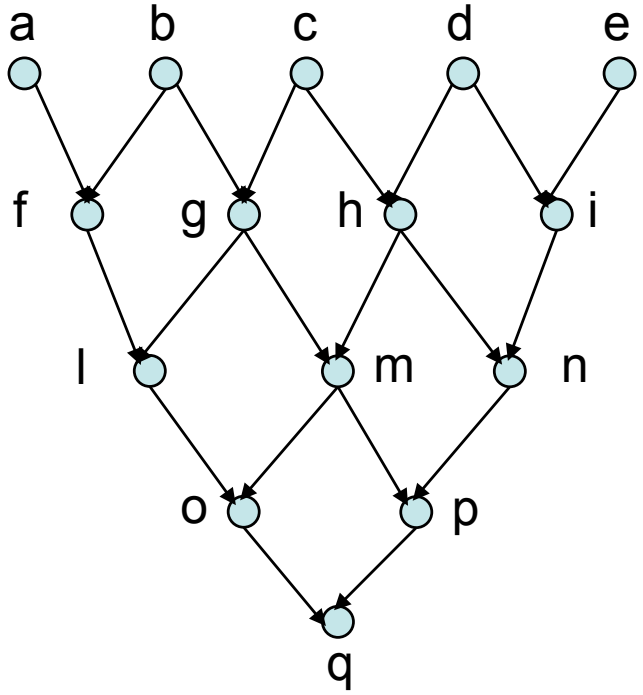
	x_v	for any source node $v \in V$
$Peb^0(G)$	$x_v \wedge x_w \rightarrow x_z$	for any $(v,z),(w,z) \in E$
	x_v	for any target $v \in V$

[Exercise 8] Prove that $Peb^0(G)$ is unsatisfiable

$Peb^0(G)$ is not Sufficient

Thm. There are polynomial size in n=|V| TLR refutations of $Peb^0(G)$.

Proof Idea. Visit the graph G passing once from any node in a depth first fashion, starting from the bottom. Apply to the following pyramidal graph and then generalize to any graph



Adding Complexity to $Peb^0(G)$

Idea. Use pebbles of two colors   and let us consider a node v pebbled if it is pebbled by one of two colours

Why.

1) Similar to the formulas used in the previous exponential separations [BEGJ98] (pyramidal)
2) From the previous proof for $Peb^0(G)$. Not possible to carry on that proofs on such a modification.

The New Principle

1. Sources nodes can be always pebbled by any of the two colours
2. Internal nodes can be pebbled of any colours whenever its parents are pebbled by any color
3. No target node is ever pebbled with any colour

Modelling 2-color PG as UNSAT Formula

G=(V,E), dag with indegree <=2.

v∈V, ther are two variables for each node

$x_{(v,R)}$ iff "node v has been pebbled RED"

$x_{(v,B)}$ iff "node v has been pebbled BLUE"

$$Peb(G) = \begin{cases} x_{(v,R)} \vee x_{(v,B)} & \forall v \in V, v \text{ source} \\ x_{(v,a)} \wedge x_{(w,b)} \rightarrow x_{(z,R)} \vee x_{(z,B)} & \forall (v,z),(w,z) \in E, \forall a,b \in \{R,B\} \\ \neg x_{(v,R)}, \neg x_{(v,B)} & \forall v \in V, v \text{ target} \end{cases}$$

[Exercise 8] Prove that for all dag G, Peb(G) is unsatisfiable and give polynomial size DLR refutations.

Lower bounds for Pebbling Formulas in TLR

Main Idea

Main Thm

Let G be a dag with indegree <=2. Delayer can always win pn(G)-3 dollars in any PD game played on peb(G).

Cor [by Thm PI96;CPT77]

Take as G the dag of [CPT]. By Main Thm and Thm of [PI] we Have that Peb(G) requires exponential size TLR refutations.

Informal proof - I

Tools.

Let G=(V,E) our dag. Let S and T the sets of sources and target nodes in G. Denote pn(G) as pn(G,S,T)

Informal Strategy Kept by the Delayer

- Delayer keeps sets of actual sources and target nodes in G

  S_i and T_i initially set resp. as S_0=S and T_0=T.

- Assume that Prover proposes the variable x(v,·) talking of node v. The Delayer will let x(v,·) unassigned only if adding v to the target T_i the pebbling number of G is decreasing

Informal proof - II

Prop1. [Invariant Property]

After round i, if the Delayer has scored p_i points, then

$$p_i >= pn(G,S,T) - pn(G,S_i,T_i)$$

Read: the Delayer scores as many points as the pebbling number of G

Prop2

After the last round f, the pebbling number of G is

$$pn(G,S_f,T_f) <= 3$$

Cor

$p_f >= pn(G)-3$. Then when G is the [CPT] graph, the theorem follows

Delayer's Strategy - I

Let $x_{(v,\cdot)}$ the variable queried by the Prover after round i.

Case 1. $v \in S_i$, the Delayer sets $x_{(v,\cdot)} = 1$

Read: Delayer must satisfy a actual source axiom, otherwise could immediately loose

Case 2. $v \in T_i$, the Delayer sets $x_{(v,\cdot)} = 0$

Read: Delayer must satisfy a actual target axiom otherwise could immediately loose

Delayer's Strategy - II

Case 3. $v \notin S_i \cup T_i$, and $pn(G, S_i, T_i) = pn(G, S_i, T_i \cup \{v\})$.

Then the Delayer sets $x_{(v,\cdot)} = 0$ and add v to T_i

Read: Since the pebbling number of G does not decrease adding v to T_i, then Delayer can safely set the variable to the value most advantegeous for her

Case 4. $v \notin S_i \cup T_i$, and $pn(G, S_i, T_i) > pn(G, S_i, T_i \cup \{v\})$.

Then the Delayer leaves $x_{(v,\cdot)}$ unset and add v to S_i.

Read: Since the pebbling number of G decreases adding v to T_i, to keep the invariant the Delayer gain 1$ but leaves the decision on the variable to the Prover. He can then safely add v to the sources, to be meant as a"pebbled" node

Keeping the Invariant - I

Reduction Lemma.

For any $v \in V$ and any set S and T.

$$pn(G,S,T) <= \max\{pn(G,S,T \cup \{v\}), pn(G,S \cup \{v\},T)+1\}$$

Proof.

To pebble T from S, first pebble $T \cup \{v\}$ from S. If this ends with a node of T pebbled we have done (in this case $pn(G,S,T) <= pn(G,S,T \cup \{v\})$. Otherwise is v to be pebbled. Then keep v pebbled and pebble T from $S \cup \{v\}$ (in this case $pn(G,S,T) <= pn(G,S \cup \{v\},T)+1$).

Keeping the Invariant - I

Prop 1.

After round i, if the Delayer has scored p_i points, then

$$p_i >= pn(G,S,T) - pn(G,S_i,T_i)$$

Proof. At the beggining p=0, $S_i=S, T_i=T$.

At any round $pn(G,S_i,T_i)$ is changing only in case 4. We loose 1 from $pn(G,S_i,T_i)$ but Delayer scores one point and since she adds v to S_i the property follows by reduction lemma

Terminating the game

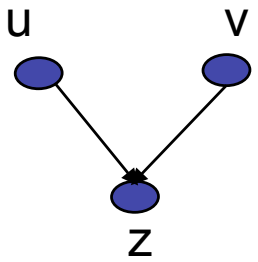
Prop2

After the last round f, the pebbling number of G is

$$pn(G,S_f,T_f) <= 3$$

Proof.

By the strategy of the Delayer neither (actual) axiom clauses nor (actual) target clauses will never be violated (Prove this ! [Exercise 9]). Then, since the Prover aways win, at the end will be violated an internal (actual) clause axiom.



Prover wins on this by using at most three pebbles