# LogicS

## Lecture #3: How many Angels can Dance on the Head of a Pin?

Carlos Areces and Patrick Blackburn

{carlos.areces,patrick.blackburn}@loria.fr

INRIA Nancy Grand Est
Nancy, France

NASSLLI 2008 - Bloomington - USA

# The Story so Far

## The Story so Far

- We've talk about Logics . . .
- . . . and we've seen an example: PL

# The Story so Far

- We've talk about Logics ...
- ...and we've seen an example: PL
- But how interesting is PL?

## The Story so Far

- We've talk about Logics ...
- ... and we've seen an example: PL
- But how interesting is PL?
  - As a language it is pretty simple: $\vee$, $\neg$, $p$, $q$, ...

# The Story so Far

- We've talk about Logics ...
- ...and we've seen an example: PL
- But how interesting is PL?
  - As a language it is pretty simple: $\vee$, $\neg$, $p$, $q$, ...
  - It's relational models are boring: just a single, labelled point.

# The Story so Far

- We've talk about Logics . . .
- . . . and we've seen an example: PL
- But how interesting is PL?
  - As a language it is pretty simple: $\vee$, $\neg$, $p$, $q$, . . .
  - It's relational models are boring: just a single, labelled point.
- How can we measure the interestingness of a Logic?

# The Story so Far

- We've talk about Logics . . .
- . . . and we've seen an example: PL
- But how interesting is PL?
  - As a language it is pretty simple: $\vee$, $\neg$, $p$, $q$, . . .
  - It's relational models are boring: just a single, labelled point.
- How can we measure the interestingness of a Logic?

Any ideas?

## The Story so Far

- We've talk about Logics ...
- ...and we've seen an example: PL
- But how interesting is PL?
  - As a language it is pretty simple: $\vee$, $\neg$, $p$, $q$, ...
  - It's relational models are boring: just a single, labelled point.
- How can we measure the interestingness of a Logic?

  Any ideas?
- One way to do it:
  1) What can we encode in the language?

# The Story so Far

- We've talk about Logics . . .
- . . . and we've seen an example: PL
- But how interesting is PL?
  - As a language it is pretty simple: $\vee$, $\neg$, $p$, $q$, . . .
  - It's relational models are boring: just a single, labelled point.
- How can we measure the interestingness of a Logic?

  Any ideas?
- One way to do it:
  1) What can we encode in the language?
  2) How much does it cost?

# What do we do Today?

# What do we do Today?

How Many Angels can Dance on the Head of a Pin?

# What do we do Today?

How Many Angels can Dance on the Head of a Pin?

Or, how much can you encode in a 1-point structure?

## What do we do Today?

How Many Angels can Dance on the Head of a Pin?

Or, how much can you encode in a 1-point structure?

- We will see that we can code quite complex problems in PL.

# What do we do Today?

How Many Angels can Dance on the Head of a Pin?

Or, how much can you encode in a 1-point structure?

- We will see that we can code quite complex problems in PL.
- In particular, we will show that we can code the Graph Coloring problem.

# What do we do Today?

How Many Angels can Dance on the Head of a Pin?

Or, how much can you encode in a 1-point structure?

- We will see that we can code quite complex problems in PL.
- In particular, we will show that we can code the Graph Coloring problem.
- Then, we will introduce an efficient algorithm for deciding satisfiability of PL-SAT: the Davis Putnam algorithm.

# More than Diplomacy

## More than Diplomacy

- We saw yesterday a simple use of propositional logic in the "Diplomatic Problem".

# More than Diplomacy

- We saw yesterday a simple use of propositional logic in the "Diplomatic Problem".
- But the expressive power of PL is enough for doing many more interesting things:
  - graph coloring

# More than Diplomacy

- We saw yesterday a simple use of propositional logic in the "Diplomatic Problem".
- But the expressive power of PL is enough for doing many more interesting things:
  - graph coloring
  - constraint satisfaction problems (e.g., Sudoku)

## More than Diplomacy

- We saw yesterday a simple use of propositional logic in the "Diplomatic Problem".
- But the expressive power of PL is enough for doing many more interesting things:
  - graph coloring
  - constraint satisfaction problems (e.g., Sudoku)
  - hardware verification

# More than Diplomacy

- We saw yesterday a simple use of propositional logic in the "Diplomatic Problem".
- But the expressive power of PL is enough for doing many more interesting things:
  - graph coloring
  - constraint satisfaction problems (e.g., Sudoku)
  - hardware verification
  - planning (e.g., graphplan).

# More than Diplomacy

- We saw yesterday a simple use of propositional logic in the "Diplomatic Problem".
- But the expressive power of PL is enough for doing many more interesting things:
  - graph coloring
  - constraint satisfaction problems (e.g., Sudoku)
  - hardware verification
  - planning (e.g., graphplan).
- Note that these problems have real world applications!

# Graph Coloring

## Graph Coloring

- The Problem: Given a graph $G = \langle N, E \rangle$ where $N$ is a set of nodes and $E$ a set of edges, and a fixed number $k$ of colors.

# Graph Coloring

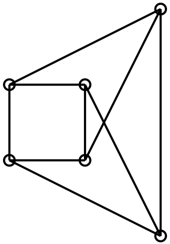
- The Problem: Given a graph $G = \langle N, E \rangle$ where $N$ is a set of nodes and $E$ a set of edges, and a fixed number $k$ of colors. Decide if we can assing colors to nodes in $N$ s.t.
  - All nodes are colored with one of the $k$ colors.

## Graph Coloring

- The Problem: Given a graph $G = \langle N, E \rangle$ where $N$ is a set of nodes and $E$ a set of edges, and a fixed number $k$ of colors. Decide if we can assing colors to nodes in $N$ s.t.
  - All nodes are colored with one of the $k$ colors.
  - For every edge $(i, j) \in N$, the color of $i$ is different from the color of $j$.

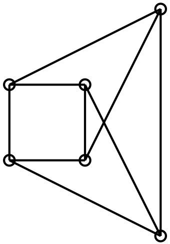
# Graph Coloring

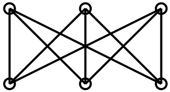
- The Problem: Given a graph $G = \langle N, E \rangle$ where $N$ is a set of nodes and $E$ a set of edges, and a fixed number $k$ of colors. Decide if we can assing colors to nodes in $N$ s.t.
  - All nodes are colored with one of the $k$ colors.
  - For every edge $(i, j) \in N$, the color of $i$ is different from the color of $j$.
- An Example:



# Graph Coloring

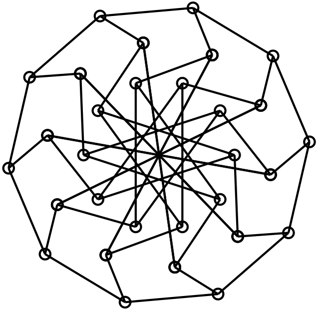
- The Problem: Given a graph $G = \langle N, E \rangle$ where $N$ is a set of nodes and $E$ a set of edges, and a fixed number $k$ of colors. Decide if we can assing colors to nodes in $N$ s.t.
  - All nodes are colored with one of the $k$ colors.
  - For every edge $(i, j) \in N$, the color of $i$ is different from the color of $j$.
- An Example:





# Graph Coloring

- The Problem: Given a graph $G = \langle N, E \rangle$ where $N$ is a set of nodes and $E$ a set of edges, and a fixed number $k$ of colors. Decide if we can assing colors to nodes in $N$ s.t.
  - All nodes are colored with one of the $k$ colors.
  - For every edge $(i, j) \in N$, the color of $i$ is different from the color of $j$.
- An Example:



# Graph Coloring: The Nitty-Gritty Details

# Graph Coloring: The Nitty-Gritty Details

- We will use $n \times k$ propositional symbols that we write $p_{ij}$ ($n$ is the number of nodes in $N$, $k$ the number of colors)

## Graph Coloring: The Nitty-Gritty Details

- We will use $n \times k$ propositional symbols that we write $p_{ij}$ ($n$ is the number of nodes in $N$, $k$ the number of colors)
- We will read $p_{ij}$ as node $i$ has color $j$

# Graph Coloring: The Nitty-Gritty Details

- We will use $n \times k$ propositional symbols that we write $p_{ij}$ ($n$ is the number of nodes in $N$, $k$ the number of colors)
- We will read $p_{ij}$ as node $i$ has color $j$
- We have to say that
  1. Each node has (at least) one color.
  2. Each node has no more than one color.
  3. Related nodes have different colors.

# Graph Coloring: The Nitty-Gritty Details

- We will use $n \times k$ propositional symbols that we write $p_{ij}$ ($n$ is the number of nodes in $N$, $k$ the number of colors)
- We will read $p_{ij}$ as node $i$ has color $j$
- We have to say that
  1. Each node has (at least) one color.
  2. Each node has no more than one color.
  3. Related nodes have different colors.

1. Each node has one color: $p_{i1} \vee \ldots \vee p_{ik}$, for $1 \leq i \leq n$

## Graph Coloring: The Nitty-Gritty Details

- We will use $n \times k$ propositional symbols that we write $p_{ij}$ ($n$ is the number of nodes in $N$, $k$ the number of colors)
- We will read $p_{ij}$ as node $i$ has color $j$
- We have to say that
  1. Each node has (at least) one color.
  2. Each node has no more than one color.
  3. Related nodes have different colors.

1. Each node has one color: $p_{i1} \vee \ldots \vee p_{ik}$, for $1 \leq i \leq n$
2. Each node has no more than one color: $\neg p_{il} \vee \neg p_{im}$, for $1 \leq i \leq n$, and $1 \leq l < m \leq k$

## Graph Coloring: The Nitty-Gritty Details

- We will use $n \times k$ propositional symbols that we write $p_{ij}$ ($n$ is the number of nodes in $N$, $k$ the number of colors)
- We will read $p_{ij}$ as node $i$ has color $j$
- We have to say that
  1. Each node has (at least) one color.
  2. Each node has no more than one color.
  3. Related nodes have different colors.

1. Each node has one color: $p_{i1} \vee \ldots \vee p_{ik}$,
   for $1 \leq i \leq n$
2. Each node has no more than one color: $\neg p_{il} \vee \neg p_{im}$,
   for $1 \leq i \leq n$, and $1 \leq l < m \leq k$
3. Neighboring nodes have different colors. $\neg p_{il} \vee \neg p_{jl}$,
   for $i$ and $j$ neighboring nodes, and $1 \leq l \leq k$

# Graph Coloring: Complexity

## Graph Coloring: Complexity

- Using the encoding in the previos page we efectively obtain for each graph $G$ and $k$ color, a formula $\varphi_{G,k}$ in $PL$ such that

  every model of $\varphi_{G,k}$ tell us a way of painting $G$ with $k$ colors

## Graph Coloring: Complexity

- Using the encoding in the previos page we efectively obtain for each graph $G$ and $k$ color, a formula $\varphi_{G,k}$ in $PL$ such that

  every model of $\varphi_{G,k}$ tell us a way of painting $G$ with $k$ colors

- If $\mathcal{M}$ is a model of $\varphi_{G,k}$ in which $p_{ij}$ is true, then paint node $i$ in $G$ with color $j$.

## Graph Coloring: Complexity

- Using the encoding in the previos page we efectively obtain for each graph $G$ and $k$ color, a formula $\varphi_{G,k}$ in $PL$ such that

  every model of $\varphi_{G,k}$ tell us a way of painting $G$ with $k$ colors

- If $\mathcal{M}$ is a model of $\varphi_{G,k}$ in which $p_{ij}$ is true, then paint node $i$ in $G$ with color $j$.
- What have we done?!!!
  - Perhaps you know that graph coloring is a difficult algorithmic problem.

## Graph Coloring: Complexity

- Using the encoding in the previos page we efectively obtain for each graph $G$ and $k$ color, a formula $\varphi_{G,k}$ in $PL$ such that

  every model of $\varphi_{G,k}$ tell us a way of painting $G$ with $k$ colors

- If $\mathcal{M}$ is a model of $\varphi_{G,k}$ in which $p_{ij}$ is true, then paint node $i$ in $G$ with color $j$.
- What have we done?!!!
  - Perhaps you know that graph coloring is a difficult algorithmic problem.
  - It is actually what is called an NP-hard problem (i.e., one of the hardest problems in the class of non-deterministic polynomial problems).

# Graph Coloring: Complexity

- Using the encoding in the previos page we efectively obtain for each graph $G$ and $k$ color, a formula $\varphi_{G,k}$ in $PL$ such that

  every model of $\varphi_{G,k}$ tell us a way of painting $G$ with $k$ colors

- If $\mathcal{M}$ is a model of $\varphi_{G,k}$ in which $p_{ij}$ is true, then paint node $i$ in $G$ with color $j$.
- What have we done?!!!
  - Perhaps you know that graph coloring is a difficult algorithmic problem.
  - It is actually what is called an NP-hard problem (i.e., one of the hardest problems in the class of non-deterministic polynomial problems).
  - Assuming that, we just proved that PL-SAT is also NP-hard.

# Decision Methods for PL

## Decision Methods for PL

- The most traditional methods for solving the SAT problem for propositional logics (PL-SAT) behave as follows. For any input formula $\varphi$

## Decision Methods for PL

- The most traditional methods for solving the SAT problem for propositional logics (PL-SAT) behave as follows. For any input formula $\varphi$
  - They always answer SATISFIABLE or UNSATISFIABLE.
  - After a finite amount of time.

# Decision Methods for PL

- The most traditional methods for solving the SAT problem for propositional logics (PL-SAT) behave as follows. For any input formula $\varphi$
  - They always answer SATISFIABLE or UNSATISFIABLE.
  - After a finite amount of time.
  - Always correctly.

# Decision Methods for PL

- The most traditional methods for solving the SAT problem for propositional logics (PL-SAT) behave as follows. For any input formula $\varphi$
  - They always answer SATISFIABLE or UNSATISFIABLE.
  - After a finite amount of time.
  - Always correctly.
- The best known complete methods probably are
  - truth tables
  - tableaux
  - axiomatics, Gentzen calculi, natural deduction, resolution
  - Davis-Putnam

# Moving into Clausal Form

## Moving into Clausal Form

- Clausal Form: Write $\varphi$ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p\text{).}$$

# Moving into Clausal Form

- Clausal Form: Write $\varphi$ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p).$$

This just means: No conjunctions inside disjunctions
Negations only on propositional simbols

# Moving into Clausal Form

- Clausal Form: Write $\varphi$ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p\text{).}$$

This just means: No conjunctions inside disjunctions
Negations only on propositional simbols

- Using the following equivalences:

$$(\neg(p \vee q)) \quad \rightsquigarrow \quad (\neg p \wedge \neg q)$$

# Moving into Clausal Form

- Clausal Form: Write $\varphi$ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p).$$

This just means: No conjunctions inside disjunctions
Negations only on propositional simbols

- Using the following equivalences:

$$\begin{array}{lcl} (\neg(p \vee q)) & \leadsto & (\neg p \wedge \neg q) \\ (\neg(p \wedge q)) & \leadsto & (\neg p \vee \neg q) \end{array}$$

# Moving into Clausal Form

- Clausal Form: Write $\varphi$ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p).$$

This just means: No conjunctions inside disjunctions
Negations only on propositional simbols

- Using the following equivalences:

$$\begin{array}{rcl} (\neg(p \vee q)) & \leadsto & (\neg p \wedge \neg q) \\ (\neg(p \wedge q)) & \leadsto & (\neg p \vee \neg q) \\ (\neg\neg p) & \leadsto & p \end{array}$$

## Moving into Clausal Form

- Clausal Form: Write $\varphi$ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p).$$

This just means: No conjunctions inside disjunctions
Negations only on propositional simbols

- Using the following equivalences:

$$\begin{array}{rcl}
(\neg(p \vee q)) & \rightsquigarrow & (\neg p \wedge \neg q) \\
(\neg(p \wedge q)) & \rightsquigarrow & (\neg p \vee \neg q) \\
(\neg\neg p) & \rightsquigarrow & p \\
(p \vee (q \wedge r)) & \rightsquigarrow & ((p \vee q) \wedge (p \vee r))
\end{array}$$

## Moving into Clausal Form

- Clausal Form: Write $\varphi$ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p).$$

This just means: No conjunctions inside disjunctions
Negations only on propositional simbols

- Using the following equivalences:

$$\begin{array}{rcl} (\neg(p \vee q)) & \rightsquigarrow & (\neg p \wedge \neg q) \\ (\neg(p \wedge q)) & \rightsquigarrow & (\neg p \vee \neg q) \\ (\neg\neg p) & \rightsquigarrow & p \\ (p \vee (q \wedge r)) & \rightsquigarrow & ((p \vee q) \wedge (p \vee r)) \end{array}$$

The clause set associated to

$$((l_{11} \vee \ldots \vee l_{1n_1}) \wedge (l_{21} \vee \ldots \vee l_{2n_2}) \wedge \ldots \wedge (l_{k1} \vee \ldots \vee l_{kn_k})) \quad \text{is}$$

# Moving into Clausal Form

- Clausal Form: Write $\varphi$ in conjunctive normal form (CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \psi \text{ a literal (i.e., } p \text{ or } \neg p).$$

This just means: No conjunctions inside disjunctions
Negations only on propositional simbols

- Using the following equivalences:

$$\begin{array}{rcl} (\neg(p \vee q)) & \rightsquigarrow & (\neg p \wedge \neg q) \\ (\neg(p \wedge q)) & \rightsquigarrow & (\neg p \vee \neg q) \\ (\neg\neg p) & \rightsquigarrow & p \\ (p \vee (q \wedge r)) & \rightsquigarrow & ((p \vee q) \wedge (p \vee r)) \end{array}$$

The clause set associated to

$$((l_{11} \vee \ldots \vee l_{1n_1}) \wedge (l_{21} \vee \ldots \vee l_{2n_2}) \wedge \ldots \wedge (l_{k1} \vee \ldots \vee l_{kn_k})) \quad \text{is}$$
$$\{\{l_{11}, \ldots, l_{1n_1}\}, \ \{l_{21}, \ldots, l_{2n_2}\} \ , \ldots , \{l_{k1}, \ldots, l_{kn_k}\}\}$$

# Example

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q)\underline{\vee}(q \vee (p \vee q)))$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q) \underline{\vee} (q \vee (p \vee q)))$
3. $(\underline{\neg\neg}(p \vee q) \wedge \neg(q \vee (p \vee q)))$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q) \underline{\vee} (q \vee (p \vee q)))$
3. $(\underline{\neg\neg}(p \vee q) \wedge \neg(q \vee (p \vee q)))$
4. $((p \vee q) \wedge \underline{\neg}(q \underline{\vee} (p \vee q)))$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q) \underline{\vee} (q \vee (p \vee q)))$
3. $(\underline{\neg\neg}(p \vee q) \wedge \neg(q \vee (p \vee q)))$
4. $((p \vee q) \wedge \underline{\neg}(q \underline{\vee} (p \vee q)))$
5. $((p \vee q) \wedge (\neg q \wedge \underline{\neg}(p \underline{\vee} q)))$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q) \underline{\vee} (q \vee (p \vee q)))$
3. $(\underline{\neg\neg}(p \vee q) \wedge \neg(q \vee (p \vee q)))$
4. $((p \vee q) \wedge \underline{\neg}(q \underline{\vee} (p \vee q)))$
5. $((p \vee q) \wedge (\neg q \wedge \underline{\neg}(p \underline{\vee} q)))$
6. $((p \vee q) \wedge (\neg q \wedge (\neg p \wedge \neg q)))$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q) \underline{\vee} (q \vee (p \vee q)))$
3. $(\underline{\neg\neg}(p \vee q) \wedge \neg(q \vee (p \vee q)))$
4. $((p \vee q) \wedge \underline{\neg}(q \underline{\vee} (p \vee q)))$
5. $((p \vee q) \wedge (\neg q \wedge \underline{\neg}(p \underline{\vee} q)))$
6. $((p \vee q) \wedge (\neg q \wedge (\neg p \wedge \neg q)))$
7. $\{\{p, q\}, \{\neg q\}, \{\neg p\}, \{\neg q\}\}$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q) \underline{\vee} (q \vee (p \vee q)))$
3. $(\underline{\neg\neg}(p \vee q) \wedge \neg(q \vee (p \vee q)))$
4. $((p \vee q) \wedge \underline{\neg}(q \underline{\vee} (p \vee q)))$
5. $((p \vee q) \wedge (\neg q \wedge \underline{\neg}(p \underline{\vee} q)))$
6. $((p \vee q) \wedge (\neg q \wedge (\neg p \wedge \neg q)))$
7. $\{\{p, q\}, \{\neg q\}, \{\neg p\}, \{\neg q\}\}$
8. $\{\{p, q\}, \{\neg q\}, \{\neg p\}\}$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q) \underline{\vee} (q \vee (p \vee q)))$
3. $(\underline{\neg\neg}(p \vee q) \wedge \neg(q \vee (p \vee q)))$
4. $((p \vee q) \wedge \underline{\neg}(q \underline{\vee} (p \vee q)))$
5. $((p \vee q) \wedge (\neg q \wedge \underline{\neg}(p \underline{\vee} q)))$
6. $((p \vee q) \wedge (\neg q \wedge (\neg p \wedge \neg q)))$
7. $\{\{p, q\}, \{\neg q\}, \{\neg p\}, \{\neg q\}\}$
8. $\{\{p, q\}, \{\neg q\}, \{\neg p\}\}$

The Diplomatic Problem:

$$(P \vee \neg Q) \wedge (Q \vee R) \wedge (\neg R \vee \neg P)$$

# Example

1. $\neg(\neg(p \vee q) \vee (\underline{\neg\neg} q \vee (p \vee q)))$
2. $\underline{\neg}(\neg(p \vee q) \underline{\vee} (q \vee (p \vee q)))$
3. $(\underline{\neg\neg}(p \vee q) \wedge \neg(q \vee (p \vee q)))$
4. $((p \vee q) \wedge \underline{\neg}(q \underline{\vee} (p \vee q)))$
5. $((p \vee q) \wedge (\neg q \wedge \underline{\neg}(p \underline{\vee} q)))$
6. $((p \vee q) \wedge (\neg q \wedge (\neg p \wedge \neg q)))$
7. $\{\{p, q\}, \{\neg q\}, \{\neg p\}, \{\neg q\}\}$
8. $\{\{p, q\}, \{\neg q\}, \{\neg p\}\}$

The Diplomatic Problem:

$$(P \vee \neg Q) \wedge (Q \vee R) \wedge (\neg R \vee \neg P)$$
$$\{\{P, \neg Q\}, \{Q, R\}, \{\neg R, \neg P\}\}$$

# The Davis-Putnam Algorithm

## The Davis-Putnam Algorithm

- The Davis-Putnam method is perhaps one of the most widely used algorithms for solving the SAT problem of PL

# The Davis-Putnam Algorithm

- The Davis-Putnam method is perhaps one of the most widely used algorithms for solving the SAT problem of PL
- Despite its age, it is still one of the most popular and successful complete methods

# The Davis-Putnam Algorithm

- The Davis-Putnam method is perhaps one of the most widely used algorithms for solving the SAT problem of PL
- Despite its age, it is still one of the most popular and successful complete methods

Let $\Sigma$ be the clause set associated to a formula $\varphi$

```
procedure DP(Σ)
```

## The Davis-Putnam Algorithm

- The Davis-Putnam method is perhaps one of the most widely used algorithms for solving the SAT problem of PL
- Despite its age, it is still one of the most popular and successful complete methods

Let $\Sigma$ be the clause set associated to a formula $\varphi$

```
procedure DP(Σ)
if Σ={} then return SAT                // (SAT)
```

# The Davis-Putnam Algorithm

- The Davis-Putnam method is perhaps one of the most widely used algorithms for solving the SAT problem of PL
- Despite its age, it is still one of the most popular and successful complete methods

Let $\Sigma$ be the clause set associated to a formula $\varphi$

```
procedure DP(Σ)
if Σ={} then return SAT                // (SAT)
if {} ∈ Σ then return UNSAT            // (UNSAT)
```

# The Davis-Putnam Algorithm

- The Davis-Putnam method is perhaps one of the most widely used algorithms for solving the SAT problem of PL
- Despite its age, it is still one of the most popular and successful complete methods

Let $\Sigma$ be the clause set associated to a formula $\varphi$

```
procedure DP(Σ)
if Σ={} then return SAT                   // (SAT)
if {} ∈ Σ then return UNSAT               // (UNSAT)
if Σ has unit clause {l}
   then DP(Σ[{l=true}])                   // (Unit Pr.)
```

# The Davis-Putnam Algorithm

- The Davis-Putnam method is perhaps one of the most widely used algorithms for solving the SAT problem of PL
- Despite its age, it is still one of the most popular and successful complete methods

Let $\Sigma$ be the clause set associated to a formula $\varphi$

```
procedure DP(Σ)
if Σ={} then return SAT                        // (SAT)
if {} ∈ Σ then return UNSAT                    // (UNSAT)
if Σ has unit clause {l}
   then DP(Σ[{l=true}])                        // (Unit Pr.)
Choose literal l and
   if DP(Σ[{l=true}]) return SAT
      then return SAT
      else return DP(Σ[{l=false}])             // (Split)
```

# Examples

# Examples

$$\neg(\neg(p \vee q) \vee (\neg\neg q \vee (p \vee q))) \; -\mathsf{CNF}\!\rightarrow \; \{\{p, q\}, \{\neg q\}, \{\neg p\}\}$$

# Examples

$$\neg(\neg(p \vee q) \vee (\neg\neg q \vee (p \vee q))) \ -\mathrm{CNF}\!\rightarrow \ \{\{p, q\}, \{\neg q\}, \{\neg p\}\}$$

$$\{\{P, \neg Q\}, \{Q, R\}, \{\neg R \vee \neg P\}\}$$

# DP: Performance

# DP: Performance

- The worst case complexity of the algorithm we show is $O(1,696^n)$, and a small modification moves it to $O(1,618^n)$.
- This is an improvement!...

## DP: Performance

- The worst case complexity of the algorithm we show is $O(1,696^n)$, and a small modification moves it to $O(1,618^n)$.
- This is an improvement!... Notice that, for example,
$$2^{100} = 1.267.650.000.000.000.000.000.000.000.000$$

## DP: Performance

- The worst case complexity of the algorithm we show is $O(1,696^n)$, and a small modification moves it to $O(1,618^n)$.
- This is an improvement!... Notice that, for example,

$$
\begin{array}{rcr}
2^{100} & = & 1.267.650.000.000.000.000.000.000.000.000 \\
1.696^{100} & = & 87.616.270.000.000.000.000.000
\end{array}
$$

## DP: Performance

- The worst case complexity of the algorithm we show is $O(1,696^n)$, and a small modification moves it to $O(1,618^n)$.
- This is an improvement!... Notice that, for example,

$$\begin{array}{rcr} 2^{100} & = & 1.267.650.000.000.000.000.000.000.000.000 \\ 1.696^{100} & = & 87.616.270.000.000.000.000.000 \\ 1.618^{100} & = & 790.408.700.000.000.000.000 \end{array}$$

## DP: Performance

- The worst case complexity of the algorithm we show is $O(1,696^n)$, and a small modification moves it to $O(1,618^n)$.
- This is an improvement!... Notice that, for example,

$$
\begin{array}{rcr}
2^{100} & = & 1.267.650.000.000.000.000.000.000.000.000 \\
1.696^{100} & = & 87.616.270.000.000.000.000.000 \\
1.618^{100} & = & 790.408.700.000.000.000.000
\end{array}
$$

- DP can reliably solve problems with up to 500 variables

## DP: Performance

- The worst case complexity of the algorithm we show is $O(1,696^n)$, and a small modification moves it to $O(1,618^n)$.
- This is an improvement!... Notice that, for example,

$$\begin{array}{rcr} 2^{100} & = & 1.267.650.000.000.000.000.000.000.000.000 \\ 1.696^{100} & = & 87.616.270.000.000.000.000.000 \\ 1.618^{100} & = & 790.408.700.000.000.000.000 \end{array}$$

- DP can reliably solve problems with up to 500 variables
- Sadly real world applications easily go into the thousands of variables (remember coloring: #nodes $\times$ #colors).

# DP: Performance

- The worst case complexity of the algorithm we show is $O(1,696^n)$, and a small modification moves it to $O(1,618^n)$.
- This is an improvement!... Notice that, for example,

$$
\begin{array}{rcr}
2^{100} & = & 1.267.650.000.000.000.000.000.000.000.000 \\
1.696^{100} & = & 87.616.270.000.000.000.000.000 \\
1.618^{100} & = & 790.408.700.000.000.000.000
\end{array}
$$

- DP can reliably solve problems with up to 500 variables
- Sadly real world applications easily go into the thousands of variables (remember coloring: #nodes $\times$ #colors).
- But this is worst time complexity. You might get lucky...

# You Might get Lucky

# You Might get Lucky

- Indeed, some method (called ‘incomplete methods’) rely in that you might get lucky.

# You Might get Lucky

- Indeed, some method (called 'incomplete methods') rely in that you might get lucky.
- We can't cover them in the course, but intuitively,
  - they are stochastic methods
  - that randomly generate valuations
  - and try to maximize the probability that the valuation actually satisfies the input formula.

# You Might get Lucky

- Indeed, some method (called 'incomplete methods') rely in that you might get lucky.
- We can't cover them in the course, but intuitively,
  - they are stochastic methods
  - that randomly generate valuations
  - and try to maximize the probability that the valuation actually satisfies the input formula.
- Examples of these methods are GSAT and WalkSAT.
- For example, a $k$-coloring algorithm based on GSAT was able to beat specialized coloring algorithms.

# What we Covered in this Lecture

## What we Covered in this Lecture

- Our first really computational lecture of the course.

## What we Covered in this Lecture

- Our first really computational lecture of the course.
- We discussed the balance between expressive power and complexity.
  - We can code complex problems in PL

## What we Covered in this Lecture

- Our first really computational lecture of the course.
- We discussed the balance between expressive power and complexity.
  - We can code complex problems in PL
    (but the coding can be unintuitive, long, complex)

## What we Covered in this Lecture

- Our first really computational lecture of the course.
- We discussed the balance between expressive power and complexity.
  - We can code complex problems in PL
    (but the coding can be unintuitive, long, complex)
  - We have efficient decision methods for PL

## What we Covered in this Lecture

- Our first really computational lecture of the course.
- We discussed the balance between expressive power and complexity.
  - We can code complex problems in PL
    (but the coding can be unintuitive, long, complex)
  - We have efficient decision methods for PL
    (able to cope with problems with hundreds of propositional symbols, but our codings easily get into the thousands).

# What we Covered in this Lecture

- Our first really computational lecture of the course.
- We discussed the balance between expressive power and complexity.
  - We can code complex problems in PL
    (but the coding can be unintuitive, long, complex)
  - We have efficient decision methods for PL
    (able to cope with problems with hundreds of propositional symbols, but our codings easily get into the thousands).
- Still, no matter how nicely we paint them, 1-point relational structures are booooooooooring.

# Relevant Bibliography I

# Relevant Bibliography I

**Cook's Theorem**: the satisfiability problem for propositional logic is NP-complete.

# Relevant Bibliography I

**Cook's Theorem**: the satisfiability problem for propositional logic is NP-complete.

- That is, any problem in NP can be reduced in polynomial time to PL SAT.

# Relevant Bibliography I

**Cook's Theorem**: the satisfiability problem for propositional logic is NP-complete.

- That is, any problem in NP can be reduced in polynomial time to PL SAT.
- In plain English: if we find a cheap way of solving PL SAT, we'll also have a cheap way of solving a hell of a lot more.

## Relevant Bibliography I

**Cook's Theorem**: the satisfiability problem for propositional logic is NP-complete.

- That is, any problem in NP can be reduced in polynomial time to PL SAT.
- In plain English: if we find a cheap way of solving PL SAT, we'll also have a cheap way of solving a hell of a lot more. (Coda: probably there is no cheap way. Too good to be true. But still, it has not been shown. $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$)

# Relevant Bibliography I

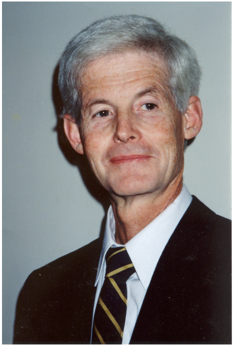
**Cook's Theorem**: the satisfiability problem for propositional logic is NP-complete.

- That is, any problem in NP can be reduced in polynomial time to PL SAT.
- In plain English: if we find a cheap way of solving PL SAT, we'll also have a cheap way of solving a hell of a lot more. (Coda: probably there is no cheap way. Too good to be true. But still, it has not been shown. $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$).
- Cook's Web page: `http://www.cs.toronto.edu/~sacook/`

## Relevant Bibliography I

**Cook's Theorem**: the satisfiability problem for propositional logic is NP-complete.

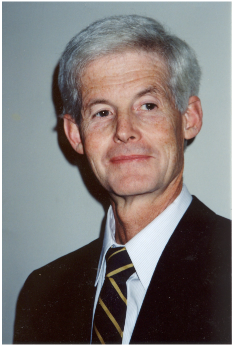
- That is, any problem in NP can be reduced in polynomial time to PL SAT.
- In plain English: if we find a cheap way of solving PL SAT, we'll also have a cheap way of solving a hell of a lot more. (Coda: probably there is no cheap way. Too good to be true. But still, it has not been shown. $P \stackrel{?}{=} NP$).
- Cook's Web page: `http://www.cs.toronto.edu/~sacook/`



## Relevant Bibliography I

**Cook's Theorem**: the satisfiability problem for propositional logic is NP-complete.

- That is, any problem in NP can be reduced in polynomial time to PL SAT.
- In plain English: if we find a cheap way of solving PL SAT, we'll also have a cheap way of solving a hell of a lot more. (Coda: probably there is no cheap way. Too good to be true. But still, it has not been shown. $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$).
- Cook's Web page: `http://www.cs.toronto.edu/~sacook/`



Cook, Stephen (1971). *The complexity of theorem proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, 151–158.

# Relevant Bibliography II

# Relevant Bibliography II

The Davis Putnam Algorithm

# Relevant Bibliography II

### The Davis Putnam Algorithm

- It was not developed by Prof. David Putnam, but by Martin Davis and Hilary Putnam.

# Relevant Bibliography II

### The Davis Putnam Algorithm

- It was not developed by Prof. David Putnam, but by Martin Davis and Hilary Putnam.
- It was then improved (with the split rule) by Martin Davis, George Logemann and Donald Loveland. The correct name is DPLL.

# Relevant Bibliography II

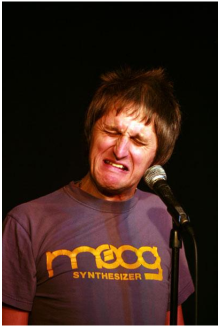
## The Davis Putnam Algorithm

- It was not developed by Prof. David Putnam, but by Martin Davis and Hilary Putnam.
- It was then improved (with the split rule) by Martin Davis, George Logemann and Donald Loveland. The correct name is DPLL.
- Davis' Web page: `http://www.cs.nyu.edu/cs/faculty/davism/`

# Relevant Bibliography II

### The Davis Putnam Algorithm

- It was not developed by Prof. David Putnam, but by Martin Davis and Hilary Putnam.
- It was then improved (with the split rule) by Martin Davis, George Logemann and Donald Loveland. The correct name is DPLL.
- Davis' Web page: `http://www.cs.nyu.edu/cs/faculty/davism/`



# Relevant Bibliography II

## The Davis Putnam Algorithm

- It was not developed by Prof. David Putnam, but by Martin Davis and Hilary Putnam.
- It was then improved (with the split rule) by Martin Davis, George Logemann and Donald Loveland. The correct name is DPLL.
- Davis' Web page: `http://www.cs.nyu.edu/cs/faculty/davism/`

# Relevant Bibliography II

## The Davis Putnam Algorithm

- It was not developed by Prof. David Putnam, but by Martin Davis and Hilary Putnam.
- It was then improved (with the split rule) by Martin Davis, George Logemann and Donald Loveland. The correct name is DPLL.
- Davis' Web page: `http://www.cs.nyu.edu/cs/faculty/davism/`


Davis, Martin; Putnam, Hillary (1960). *A Computing Procedure for Quantification Theory*. Journal of the ACM 7 (1): 201–215.

Davis, Martin; Logemann, George, and Loveland, Donald (1962). *A Machine Program for Theorem Proving*. Communications of the ACM 5 (7): 394–397.

# Interesting Links

## Interesting Links

- The SAT*Live*! page. The page for all things SAT
  http://www.satlive.org/

## Interesting Links

- The SAT*Live*! page. The page for all things SAT
  `http://www.satlive.org/`
- The international SAT Competitions web page
  `http://www.satcompetition.org/`

## Interesting Links

- The SAT*Live*! page. The page for all things SAT
  `http://www.satlive.org/`
- The international SAT Competitions web page
  `http://www.satcompetition.org/`
- Some provers
  - RSat `http://reasoning.cs.ucla.edu/rsat/`
  - MiniSat `http://minisat.se/`
  - PicoSat `http://fmv.jku.at/picosat/`

# Interesting Links

- The SAT*Live*! page. The page for all things SAT
  `http://www.satlive.org/`
- The international SAT Competitions web page
  `http://www.satcompetition.org/`
- Some provers
  - RSat `http://reasoning.cs.ucla.edu/rsat/`
  - MiniSat `http://minisat.se/`
  - PicoSat `http://fmv.jku.at/picosat/`
- The Walksat Home Page
  `http://www.cs.rochester.edu/~kautz/walksat/`

## The Next Lecture

Lecture #4

**Points & Lines™**

Bored of working always with the same old point?

UPGRADE your model! Get OTHER points and even LINES!

(Offer available for a limited time)