5CCS2FC2: Foundations of Computing II

Introduction to SAT Solving

Week 5

Dr Christopher Hampson

Department of Informatics

King's College London

A (Naïve) Branching SAT Algorithm

A (Naïve) Branching Divide-and-Conquer Agorithm

- The algorithm takes two inputs

> **Input 1)** A set of clauses C
>
> **Input 2)** A patial assignment $U \subseteq \{$ literals from $C \}$

- We **branch** on any variable P appearing in C:
  - First choose to make $P := \textbf{False}$ by adding $\neg P$ to U
  - Next choose to make $P := \textbf{True}$ by adding P to U

- **A (Naïve) Branching Divide-and-Conquer Agorithm**
  - We **eliminate** any satisfied clauses from C and **simplify** and remaining clauses.
  - The remaining set of clauses is **self-similar** to the original set of clauses but with *one fewer variable*.
  - Hence we can continue by **recursively** calling this algorithm until either:
    - all clauses are **satisfied** and $C = \emptyset$.
    - a **conflict** is detected in the partial assignment U

A (Naïve) Branching SAT Algorithm

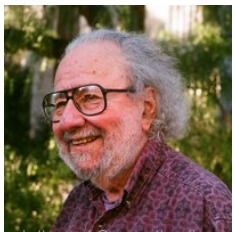
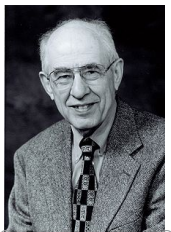
BRANCH-SAT (C, U):

1. **If** U contains a conflict : **Return** **False**
2. **If** $C = \emptyset$: **Return** **True**
3. Simplify the clauses in C as much as possible
4. Select a variable P appearing in C
5. **If** BRANCH-SAT $(C, U \cup \{\neg P\}) =$ **True** : **Return** **True**
6. **If** BRANCH-SAT $(C, U \cup \{P\}) =$ **True** : **Return** **True**
7. **Return** **False**

The DPLL Algorithm

The DPLL Algorithm

- **The Davis-Putnam-Logemann-Loveland (DPLL) algorithm**
  - Proposed in 1962 by M.Davis, G.Logemann, and D.Loveland
  - Previous work in 1960 by M.Davis and H.Putnam.
  - A **divide-and-conquer** backtrack-search type algorithm,
  - Employs **pure literal elimination** and **unit propagation**

- **Pure Literal Elimination**
  - A **pure literal** is any literal that occurs only positive or negatively in all clauses

$$(P \vee Q \vee \neg S) \quad , \quad (P \vee \neg Q) \quad , \quad (Q \vee R) \quad , \quad (Q \vee \neg R \vee \neg S)$$

  - We can always add these literals to our assignment without risk of a conflict

$$P := \text{True} \qquad \text{and} \qquad S := \text{False}$$

- **Pure Literal Elimination**
  - We can **simplify** to **eliminate** any clause containing these pure literals.

$$(P \vee Q \vee \neg S) \quad , \quad (P \vee \neg Q \vee R) \quad , \quad (Q \vee R) \quad , \quad (Q \vee \neg R \vee \neg S)$$

Pure Literal Elimination assigns literals that **SHOULD** be assigned

- **Unit Propagation**
  - A **unit clause** is any clause that contains only a single literal

$$\neg Q \qquad (P \vee Q) \qquad (\neg P \vee \neg R \vee S) \qquad (\neg P \vee Q \vee \neg S)$$
$$(\neg P \vee \neg Q) \qquad R \qquad (Q \vee R \vee S)$$

  - We have **no choice** in the assignment of unit clauses

$$Q := \textbf{False} \quad \text{and} \quad R := \textbf{True}$$

- **Unit Propagation**
  - We can **eliminate** any clause containing a unit clause literal.

$$\cancel{\neg Q} \qquad (P \vee Q) \qquad (\neg P \vee \neg R \vee S) \qquad (\neg P \vee Q \vee \neg S)$$

$$\cancel{(\neg P \vee \neg Q)} \qquad \cancel{R} \qquad \cancel{(Q \vee R \vee S)}$$

- **Unit Propagation**
  - And **simplify** any clause containing the negation of the unit clause literal.

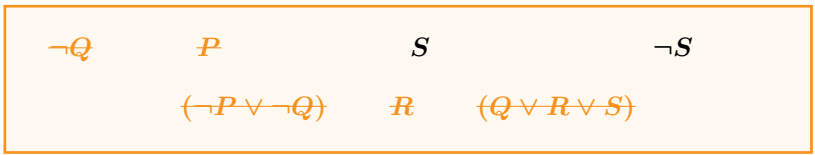
$$\cancel{\neg Q} \qquad P \qquad (\neg P \vee S) \qquad (\neg P \vee \neg S)$$

$$\cancel{(\neg P \vee \neg Q)} \qquad \cancel{R} \qquad \cancel{(Q \vee R \vee S)}$$

  - These assignments **propagate** leading to further unit clauses

$$P := \text{True}$$

- **Unit Propagation**
  - We can then repeat by **eliminating** and **simplifying** further



Unit Propagation assigns literals that **MUST** be assigned

- **The DPLL ALgorithm**
  - The algorithm takes two inputs

> **Input 1)** A set of clauses C
>
> **Input 2)** A patial assignment $U \subseteq \{$ literals from $C \}$

  - We simplify C and apply the following rules:
    - **pure literal elimination**
    - **unit propagation**

- **The DPLL ALgorithm**
  - If any clauses remain unsatisfied, we **branch** on any remaining variable P:
    - First choose to make $P := \textbf{False}$ by adding $\neg P$ to U
    - Next choose to make $P := \textbf{True}$ by adding P to U
  - We can achieve this by **recursively** calling the DPLL algorithm until either:
    - all clauses are **satisfied** and $C = \emptyset$.
    - a **conflict** is detected in the partial assignment U

The DPLL Algorithm

DPLL $(C,\ U)$:

1. **If** U contains a conflict : **Return** **False**
2. **If** $C = \emptyset$: **Return** **True**
3. Simplify the clauses in C as much as possible
4. Apply PURE-LITERAL-ELIMINATION to C
5. Apply UNIT-PROPAGATION to C
6. Select a variable P appearing in C
7. **If** DPLL $(C,\ U \cup \{\neg P\}) =$ **True** : **Return** **True**
8. **If** DPLL $(C,\ U \cup \{P\}) =$ **True** : **Return** **True**
9. **Return** **False**

End of Slides!