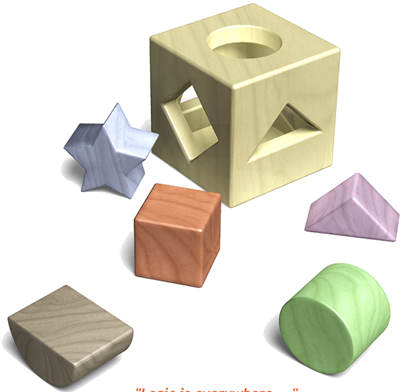
# SAT Solving – Implementation

**Steffen Hölldobler and Norbert Manthey**
International Center for Computational Logic
Technische Universität Dresden
Germany

- **Data structures**

- **Algorithms**



*"Logic is everywhere ..."*

# Warm Up

- **How is an array stored in memory?**

# Warm Up

- **How is an array stored in memory?**

- **How is a $n \cdot m$ matrix stored in memory?**

# Warm Up

- **How is an array stored in memory?**
- **How is a $n \cdot m$ matrix stored in memory?**
- **How is an array stored in Java?**

# Revision

- **Used Data Types**

# Clauses and Conjunctive Normal Forms

- **Definition**
  - A clause is a generalized disjunction $[L_1, \ldots, L_n]$, $n \geq 0$, where every $L_i$, $1 \leq i \leq n$, is a literal

# Clauses and Conjunctive Normal Forms

- **Definition**
  - A **clause** is a generalized disjunction $[L_1, \ldots, L_n]$, $n \geq 0$, where every $L_i$, $1 \leq i \leq n$, is a literal

- **Definition**
  - A formula is in **conjunctive normal form** (**clause form**, **CNF**) **iff** it is of the form $\langle C_1, \ldots, C_m \rangle$, $m \geq 0$, and every $C_j$, $1 \leq j \leq m$, is a clause

- **Agreement**
  - An interpretation for a formula $F$ can be represented as sequence of literals.

# Used Data Structures

- **(Multi-)Sets for clauses and the formula**

- **Sequences for the interpretation**

# Used Algorithms

- **Unit propagation**
- **Clause learning**

# Implementing Interpretations

# How to Implement an Interpretation

- **Given: the input formula $F$**
  - **with the variables $n = |\mathcal{R}_F|$**

- **For a clause, an interpretation $J$ is usually used**
  - **to test $J \models C$ for some clause, or**
  - **to compute $C|_J$.**

- **How to implement an interpretation?**

# Implement an Interpretation as Sequence

`space saving`

**array** S

```
contains (integer a)
  for i in S
    if a = i then return true
  return false

insert (integer a)
  if not contains(a) then
    append a to S

erase (integer a)
  if contains(a) then
    remove a from
```

# Implement an Interpretation as Sequence

`space saving`

**array** S

*contains* (integer ***a***)

  **for *i* in *S***
    **if *a* = *i* then return true**
  **return false**

*insert* (integer ***a***)

  **if not** contains(***a***) **then**
    append ***a*** to ***S***

*erase* (integer ***a***)

  **if** contains(***a***) **then**
    remove ***a*** from

`time saving`

**array** S
**array** T with ***n*** elements

*contains* (integer ***a***)

  **return** T[***a***]

*insert* (integer ***a***)

  **if not** contains(***a***) **then**
    append ***a*** to ***S***
    ***T*[*a*] = true**

*erase* (integer ***a***)

  **if** contains(***a***) **then**
    remove ***a*** from ***S***
    ***T*[*a*] = false**

# Implement an Interpretation as Sequence

- **What is the complexity of erasing an element from a sequence?**

# Implement an Interpretation as Sequence

- **What is the complexity of erasing an element from a sequence?**

- **How about in case of an interpretation?**
  - **More particularly in the case of the CDCL algorithm?**

# Decision Levels and Reasons

- **The *decision level* denotes how many decision literals have already been added to the interpretation $J$ once the literal $x$ has been added.**

# Decision Levels and Reasons

- **The *decision level* denotes how many decision literals have already been added to the interpretation $J$ once the literal $x$ has been added.**
- **Definition**
  - **The decision level of a literal $x$ with respect to an interpretation $J$ is the number of decision literals that have been added to this interpretation once the literal $x$ has been added: $|\{y \mid \dot{y} \in (J'x) \text{ where } J = J'xJ''\}|$.**

# Decision Levels and Reasons

- **The *decision level* denotes how many decision literals have already been added to the interpretation $J$ once the literal $x$ has been added.**
- **Definition**
  - **The decision level of a literal $x$ with respect to an interpretation $J$ is the number of decision literals that have been added to this interpretation once the literal $x$ has been added: $|\{y \mid \dot{y} \in (J'x) \text{ where } J = J'xJ''\}|$.**
  - $\texttt{decision\_level}(J, x) = |\{y \mid \dot{y} \in (J'x) \text{ and } J = J'xJ''\}|$
- **or**
  - $\texttt{decision\_level}(J, v) = |\{y \mid \dot{y} \in (J'x) \text{ and } J = J'xJ'' \text{ and } \mathrm{var}(x) = v\}|$.

# Decision Levels and Reasons

- The *decision level* denotes how many decision literals have already been added to the interpretation $J$ once the literal $x$ has been added.
- Definition
  - The decision level of a literal $x$ with respect to an interpretation $J$ is the number of decision literals that have been added to this interpretation once the literal $x$ has been added: $|\{y \mid \dot{y} \in (J'x) \text{ where } J = J'xJ''\}|$.
  - $\texttt{decision\_level}(J, x) = |\{y \mid \dot{y} \in (J'x) \text{ and } J = J'xJ''\}|$
- or
  - $\texttt{decision\_level}(J, v) = |\{y \mid \dot{y} \in (J'x) \text{ and } J = J'xJ'' \text{ and } \text{var}(x) = v\}|$.
- Definition
  - A clause $C$ is called a reason clause of a literal $x$ with respect to an interpretation $J$ if there is an interpretation $J'$ with $J = J'J''$ and the reduct $C|_{J'}$ with respect to the interpretation $J'$ is the unit clause $C|_{J'} = (x)$.
- For convenience we introduce a function that maps to the (set of) reason(s): $\texttt{reason}(F, J, x)$.

# Conflict Literal

- **Given a conflict clause $C$ with respect to an interpretation $J$, the conflict literal $x \in C$ is the literal of the clause $C$ whose complement $\neg x$ has the rightmost position in the sequence representation of the interpretation.**

# Conflict Literal

- **Given a conflict clause $C$ with respect to an interpretation $J$, the conflict literal $x \in C$ is the literal of the clause $C$ whose complement $\neg x$ has the rightmost position in the sequence representation of the interpretation.**
- **Definition**
  - **Given a conflict clause $C$, a literal $x \in C$ and an interpretation $J$ with $J = J'\neg xJ''$, then $x$ is the conflict literal of $C$, if $(C \setminus \{x\}) \cap (J'') = \emptyset$.**

# Conflict Literal

- Given a conflict clause $C$ with respect to an interpretation $J$, the conflict literal $x \in C$ is the literal of the clause $C$ whose complement $\neg x$ has the rightmost position in the sequence representation of the interpretation.
- **Definition**
  - Given a conflict clause $C$, a literal $x \in C$ and an interpretation $J$ with $J = J' \neg x J''$, then $x$ is the conflict literal of $C$, if $(C \setminus \{x\}) \cap (J'') = \emptyset$.

- **Conflict Level**
  - The conflict level of a conflict clause $C$ with respect to an interpretation $J$ is the highest decision level of all the literals $x$ that occur in the clause.

# Implementing Unit Propagation

# Pseudo Code for Unit Propagation

***UP*** (CNF formula $\boldsymbol{F}$, interpretation $\boldsymbol{J}$)

**Input:** A formula $\boldsymbol{F}$ in CNF, an interpretation $\boldsymbol{J}$
**Output:** An extended interpretation $\boldsymbol{J}$

```
P := ()                      // start with empty interpretation
while (x) ∈ F|_JP do         // unit rule
    P := Px                  // extend propagated literals
return (JP)
```

- **C source code is still different**

# Pseudo Code for Unit Propagation

---

***UP*** (CNF formula $F$, interpretation $J$)

**Input:** A formula $F$ in CNF, an interpretation $J$
**Output:** An extended interpretation $J$

```
P := ()                                      // start with empty interpretation
while (x) ∈ F|JP and not [] ∈ F|JP do        // unit rule
   P := Px                                   // extend propagated literals
return (JP)
```

- **C source code is still different**

# Pseudo Code for Unit Propagation

***UP*** (CNF formula $\boldsymbol{F}$, interpretation $\boldsymbol{J}$)

**Input:** A formula $\boldsymbol{F}$ in CNF, an interpretation $\boldsymbol{J}$
**Output:** An extended interpretation $\boldsymbol{J}$

| | |
|---|---|
| $P := ()$ | // start with empty interpretation |
| **while** $(x) \in F\|_{JP}$ **and not** $[] \in F\|_{JP}$ **do** | // unit rule |
|     $P := Px$ | // extend propagated literals |
|     $reason(x) = C$ | // set reason |
| **return** $(JP)$ | |

- **C source code is still different**

# Implementing Clause Learning

# Clause Learning

- **Properties of a learned clause**

# Clause Learning

- **Properties of a learned clause**

- **Short**

# Clause Learning

- **Properties of a learned clause**
- **Short**
- **Good backjump distance**

# Clause Learning

- **Properties of a learned clause**
- **Short**
- **Good backjump distance**
- **Should trigger unit propagation**
  - **Such a clause is called asserting**

# Clause Learning Algorithm

***ConflictAnalysis*** (CNF formula $\boldsymbol{F}$, interpretation $\boldsymbol{J}$, conflict clause $\boldsymbol{C}$)

**Input:** A formula $\boldsymbol{F}$ in CNF, an interpretation $\boldsymbol{J}$, clause $\boldsymbol{C}$
**Output:** A learned clause $\boldsymbol{D}$

```
D := C                                          // start with the conflict
while some condition do                         // depending on the wanted clause
   while J = J′L and ¬L ∉ D do                  // unit rule
      J = J′                                    // remove last literal from J
   if reason(F, J, L) ≠ ∅                       // depending on condition always true
      D := D ⊗ reason(F, J, L)                  // resolve with a reason
return D
```

- **Usually, pick first reason (the one stored during UP)**

# Clause Learning Algorithm

***ConflictAnalysis*** (CNF formula $F$, interpretation $J$, conflict clause $C$)

**Input:** A formula $F$ in CNF, an interpretation $J$, clause $C$
**Output:** A learned clause $D$

$D := C$ — // start with the conflict
**while** some condition **do** — // depending on the wanted clause
  **while** $J = J'L$ **and** $\neg L \notin D$ **do** — // unit rule
    $J = J'$ — // remove last literal from $J$
  **if** $\mathtt{reason}(F, J, L) \neq \emptyset$ — // depending on condition always true
    $D := D \otimes \mathtt{reason}(F, J, L)$ — // resolve with a reason
**return** $D$

- **Usually, pick first reason (the one stored during UP)**
- **Invariant: $C$ has at least two literals of the conflict level**
- **Invariant: $D$ is always falsified, $D|_J = []$.**

# Clause Learning Algorithm

***ConflictAnalysis*** (CNF formula $\boldsymbol{F}$, interpretation $\boldsymbol{J}$, conflict clause $\boldsymbol{C}$)

**Input:** A formula $\boldsymbol{F}$ in CNF, an interpretation $\boldsymbol{J}$, clause $\boldsymbol{C}$
**Output:** A learned clause $\boldsymbol{D}$

```
D := C                                      // start with the conflict
while some condition do                     // depending on the wanted clause
   while J = J′L and ¬L ∉ D do              // unit rule
      J = J′                                // remove last literal from J
   if reason(F, J, L) ≠ ∅                   // depending on condition always true
      D := D ⊗ reason(F, J, L)              // resolve with a reason
return D
```

- **Possible abort conditions**
  - **Decision clause**:
    - **for all $L \in D$ there is no reason, $\texttt{reason}(F, J, \neg L) = \emptyset$**

# Clause Learning Algorithm

***ConflictAnalysis*** (CNF formula $F$, interpretation $J$, conflict clause $C$)

**Input:** A formula $F$ in CNF, an interpretation $J$, clause $C$
**Output:** A learned clause $D$

$D := C$ // start with the conflict
**while** some condition **do** // depending on the wanted clause
    **while** $J = J'L$ **and** $\neg L \not\in D$ **do** // unit rule
        $J = J'$ // remove last literal from $J$
    **if** $\texttt{reason}(F, J, L) \neq \emptyset$ // depending on condition always true
        $D := D \otimes \texttt{reason}(F, J, L)$ // resolve with a reason
**return** $D$

- **Possible abort conditions**
  - **Decision clause**:
    - **for all $L \in D$ there is no reason, $\texttt{reason}(F, J, \neg L) = \emptyset$**
  - **1st UIP clause (unique implication point):**
    - **exactly one literal of the highest decision level left**

# Clause Learning Algorithm

***ConflictAnalysis*** (CNF formula $F$, interpretation $J$, conflict clause $C$)

**Input:** A formula $F$ in CNF, an interpretation $J$, clause $C$
**Output:** A learned clause $D$

| | |
|---|---|
| $D := C$ | // start with the conflict |
| **while** some condition **do** | // depending on the wanted clause |
|   **while** $J = J'L$ **and** $\neg L \not\in D$ **do** | // unit rule |
|     $J = J'$ | // remove last literal from $J$ |
|   **if** $\mathtt{reason}(F, J, L) \neq \emptyset$ | // depending on condition always true |
|     $D := D \otimes \mathtt{reason}(F, J, L)$ | // resolve with a reason |
| **return** $D$ | |

- **Possible abort conditions**
  - **Decision clause:**
    - **for all $L \in D$ there is no reason, $\mathtt{reason}(F, J, \neg L) = \emptyset$**
  - **1st UIP clause (unique implication point):**
    - **exactly one literal of the highest decision level left**
- **1st UIP clause is constructed faster, and usually shorter**
- **Not discussed here: clause minimization**

# Clause Learning Algorithm

***ConflictAnalysis*** (CNF formula $F$, interpretation $J$, conflict clause $C$)

**Input:** A formula $F$ in CNF, an interpretation $J$, clause $C$
**Output:** A learned clause $D$

| | |
|---|---|
| $D := C$ | // start with the conflict |
| **while** some condition **do** | // depending on the wanted clause |
|   **while** $J = J'L$ **and** $\neg L \not\in D$ **do** | // unit rule |
|     $J = J'$ | // remove last literal from $J$ |
|   **if** $\texttt{reason}(F, J, L) \neq \emptyset$ | // depending on condition always true |
|     $D := D \otimes \texttt{reason}(F, J, L)$ | // resolve with a reason |
| **return** $D$ | |

- **Can this algorithm be implemented faster?**
- **Assume we are interested in the 1st UIP clause!**
- ***D* is a set of literals**

# Clause Learning Algorithm

***ConflictAnalysis*** (CNF formula $\boldsymbol{F}$, interpretation $\boldsymbol{J}$, conflict clause $\boldsymbol{C}$)

**Input:** A formula $\boldsymbol{F}$ in CNF, an interpretation $\boldsymbol{J}$, clause $\boldsymbol{C}$
**Output:** A learned clause $\boldsymbol{D}$

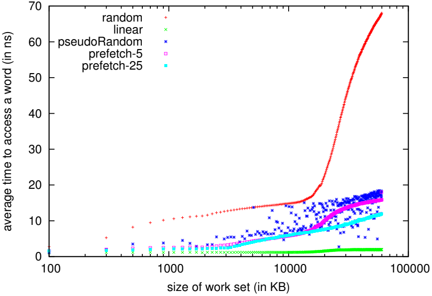
```
D := C                                   // start with the conflict
while some condition do                  // depending on the wanted clause
   while J = J′L and ¬L ∉ D do           // unit rule
      J = J′                             // remove last literal from J
   if reason(F, J, L) ≠ ∅                // depending on condition always true
      D := D ⊗ reason(F, J, L)           // resolve with a reason
return D
```

- **Can this algorithm be implemented faster?**
- **Assume we are interested in the 1st UIP clause!**
- ***D* is a set of literals**
- ***D* can be represented implicitely by an occurence array and *J***

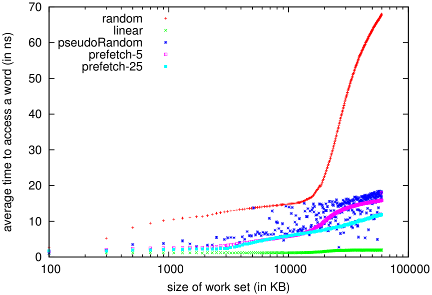
# Choosing Data Structures

# Data structures

- Things to worry about for efficiency:
  - Number of memory accesses
  - Order of memory locations to be accessed



# Data structures




- **pseudoRandom: random cache line, multiple accesses**
- **prefetching: tell the memory where the X-th next access will be**

# Data structures

- Iterate through data structures linearly, use arrays
- Reduce number of memory accesses
  - Blocking Literal in watch list
- Store data about variables together in one block
  - Assignment, reason, decision level