Practical SAT Solving
Lecture 1
Carsten Sinz, Tomáš Balyo | April 24, 2017
Events

- Lectures
  - Room 236
  - Every Monday at 14:00
  - Please enroll to the lecture on http://campus.studium.kit.edu

- Exercises
  - Room Atis-SR 010
  - Every second Thursday (starting 27.4.) at 14:00

- Exams
  - 31.7. and 9.10.
  - Oral examination
  - Bonus points for homework improve the grade
Homeworks and Lecture Notes

- You get homework points for doing homework and showing up with them on the exercises.
- You will have the possibility to collect at least 120 points for homeworks during the semester (plus a lot more bonus points).
- A volunteer can write up notes from the lecture for a reward of 25 points. Can be a group, points are split.
- You must collect at least 60 points to pass the exercises and be allowed to participate in the oral exam.
- Extra points improve your grade:
  - ≥ 70 points improves grade by 0.3
  - ≥ 90 points improves grade by 0.7
  - ≥ 110 points improves grade by 1.0
Contact

- Carsten Sinz
  - email: carsten.sinz@kit.edu
  - room: 028

- Tomas Balyo
  - email: tomas.balyo@kit.edu
  - room: 210

- Homepage
  - url: http://baldur.iti.kit.edu/sat/
  - Contains all the slides and homework assignments
Goals of this lecture

- How do SAT solvers work
  - Algorithms
- How to make a SAT solver
  - Implementation techniques
- How to use a SAT solver efficiently
  - How to encode stuff into CNF
Propositional Logic

- A Boolean variable \( (x) \) is a variable with two possible values: True and False.
- A literal is a Boolean variable \( x \) (positive literal) or its negation \( \overline{x} \) (negative literal).
- A clause is a disjunction (or \( = \lor \)) of literals.
- A CNF\(^1\) formula is a conjunction (and \( = \land \)) of clauses.

Example

\[
F = (x_1 \lor x_2) \land (x_1 \lor x_2 \lor x_3) \land (x_1)
\]

\( \text{Vars}(F) = \{ x_1, x_2, x_3 \} \)

\( \text{Lits}(F) = \{ x_1, \overline{x_1}, x_2, \overline{x_2}, x_3 \} \)

\( \text{Cls}(F) = \{ (x_1 \lor x_2), (x_1 \lor x_2 \lor x_3), (x_1) \} \)

\(^1\)Conjunctive Normal Form
A *truth assignment* $\phi$ assigns a truth value (True or False) to each Boolean variable $x$, i.e., $\phi(x) = True$ or $\phi(x) = False$.

We say that $\phi$ satisfies

- a positive literal $x$ if $\phi(x) = True$
- a negative literal $\overline{x}$ if $\phi(x) = False$
- a clause if it satisfies at least one of its literals
- a CNF formula if it satisfies all of its clauses

If $\phi$ satisfies a CNF $F$ then we call $\phi$ a *satisfying assignment* of $F$.

A formula $F$ is *satisfiable* if there is $\phi$ that satisfies $F$.

The *Satisfiability Problem* is to determine whether a given formula is satisfiable. If so, we would also like to see a satisfying assignment.
Satisfiability

- A *truth assignment* $\phi$ assigns a truth value (True or False) to each Boolean variable $x$, i.e., $\phi(x) = True$ or $\phi(x) = False$.

- We say that $\phi$ satisfies
  - a positive literal $x$ if $\phi(x) = True$
  - a negative literal $\overline{x}$ if $\phi(x) = False$
  - a clause if it satisfies at least one of its literals
  - a CNF formula if it satisfies all of its clauses

- If $\phi$ satisfies a CNF $F$ then we call $\phi$ a *satisfying assignment* of $F$.

- A formula $F$ is *satisfiable* if there is $\phi$ that satisfies $F$.

- The *Satisfiability Problem* is to determine whether a given formula is satisfiable. If so, we would also like to see a satisfying assignment.
Satisfiability - Examples

Satisfiable Formulas

\((x_1)\)
\((x_2 \lor x_8 \lor x_3)\)
\((\overline{x_1} \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor x_3) \land (x_1)\)

Unsatisfiable Formulas

\((x_1) \land (\overline{x_1})\)
\((x_1) \land (\overline{x_1}) \land (x_2 \lor x_8 \lor \overline{x_3})\)
\((x_1 \lor x_2) \land (\overline{x_1} \lor x_2) \land (x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor \overline{x_2})\)
Satisfiability - A Practical Example

Scheduling a meeting consider the following constraints

- Adam can only meet on Monday or Wednesday
- Bridget cannot meet on Wednesday
- Charles cannot meet on Friday
- Darren can only meet on Thursday or Friday

Expressed as SAT

\[ F = (x_1 \lor x_3) \land (\overline{x_3}) \land (\overline{x_5}) \land (x_4 \lor x_5) \land \text{AtMostOne}(x_1, x_2, x_3, x_4, x_5) \]
Satisfiability - A Practical Example

Scheduling a meeting consider the following constraints

- Adam can only meet on Monday or Wednesday
- Bridget cannot meet on Wednesday
- Charles cannot meet on Friday
- Darren can only meet on Thursday or Friday

Expressed as SAT

\[
F = (x_1 \lor x_3) \land (\overline{x_3}) \land (\overline{x_5}) \land (x_4 \lor x_5) \\
\land (x_1 \lor \overline{x_2}) \land (x_1 \lor x_3) \land (\overline{x_1} \lor \overline{x_4}) \land (\overline{x_1} \lor \overline{x_5}) \\
\land (\overline{x_2} \lor x_3) \land (\overline{x_2} \lor x_4) \land (x_2 \lor x_5) \\
\land (\overline{x_3} \lor \overline{x_4}) \land (\overline{x_3} \lor \overline{x_5}) \\
\land (\overline{x_4} \lor \overline{x_5})
\]
Satisfiability - A Practical Example

Scheduling a meeting consider the following constraints

- Adam can only meet on Monday or Wednesday
- Bridget cannot meet on Wednesday
- Charles cannot meet on Friday
- Darren can only meet on Thursday or Friday

Expressed as SAT

\[ F = (x_1 \lor x_3) \land (\overline{x_3}) \land (\overline{x_5}) \land (x_4 \lor x_5) \land \text{AtMostOne}(x_1, x_2, x_3, x_4, x_5) \]

Solution: Unsatisfiable, i.e., it is impossible to schedule a meeting with these constraints
Satisfiability - Hardness

Satisfiability is NP-Complete [1], proof idea:

- SAT is in NP – easy, checking a solution can be done in linear time
- SAT in NP-hard – encode the run of a non-deterministic Turing machine on an input to a CNF formula.

Consequences:

- We don’t have a polynomial algorithm for SAT (yet) :( 
- If $P \neq NP$ then we won’t have a polynomial algorithm :'( 
- All the known complete algorithms have exponential runtime in the worst case.

Hardness, try it yourself: http://www.cs.utexas.edu/~marijn/game/
Satisfiability - History

- 1960 The first SAT solving algorithm DP (Davis, Putnam) [2]
- 1971 SAT was the first NP-Complete problem [1]
- 1992 The First International SAT Competition, followed by 1993, 1996, since 2002 every year
- 1996 Conflict Driven Clause Learning [5]
- 1996 The First International SAT Conference (Workshop), followed by 1998, since 2000 every year

Early 90’s: 100 variables, 200 clauses, Today: 1,000,000 variables and 5,000,000 clauses.
Applications of SAT solving

- **Hardware Model Checking**
  - All major hardware companies (Intel, ...) use SAT solver to verify their chip designs

- **Software Verification**
  - SAT solver based SMT solvers are used to verify Microsoft software products
  - Embedded software in Cars, Airplanes, Refrigerators, ...
  - Unix utilities

- **Automated Planning and Scheduling in Artificial Intelligence**
  - Still one of the best approaches for optimal planning

- **Number Theoretic Problems (Pythagorean Triples)**

- **Solving other NP-hard problems (coloring, clique, ...)**
Zahlenrätsel

Der längste Mathe-Beweis der Welt

Pythagorean Triples

Problem Definition
Is it possible to assign to each integer $1, 2, \ldots, n$ one of two colors such that if $a^2 + b^2 = c^2$ then $a, b$ and $c$ do not all have the same color.

- Solution: Nope
- for $n = 7825$ it is not possible
- proof obtained by a SAT solver has 200 Terrabytes – the largest Math proof ever

How to encode this?
- for each integer $i$ we have a Boolean variable $x_i$, $x_i = 1$ if color of $i$ is 1, $x_i = 0$ otherwise.
- for each $a, b, c$ such that $a^2 + b^2 = c^2$ we have two clauses: $(x_a \lor x_b \lor x_c)$ and $(\overline{x_a} \lor \overline{x_b} \lor \overline{x_c})$
Pythagorean Triples

Problem Definition

Is it possible to assign to each integer 1, 2, . . . , n one of two colors such that if \( a^2 + b^2 = c^2 \) then a, b and c do not all have the same color.

- Solution: Nope
- for \( n = 7825 \) it is not possible
- proof obtained by a SAT solver has 200 Terrabytes – the largest Math proof ever

How to encode this?

- for each integer \( i \) we have a Boolean variable \( x_i \), \( x_i = 1 \) if color of \( i \) is 1, \( x_i = 0 \) otherwise.
- for each \( a, b, c \) such that \( a^2 + b^2 = c^2 \) we have two clauses:
  \((x_a \lor x_b \lor x_c)\) and \((\overline{x_a} \lor \overline{x_b} \lor \overline{x_c})\)
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2$, $x_5$, $x_8$ are equally spaced.
- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?
- Encode what’s forbidden: $x_2 x_5 x_8 \neq 111$ is the same as $(\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_8)$.
- Writing, e.g., $\bar{25} \bar{8}$ for the clause $(\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_8)$, we arrive at 32 clauses for the 9 digit sequence:
  123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159, 
  $\bar{123}$, $\bar{234}$, \ldots, $\bar{789}$, $\bar{135}$, $\bar{246}$, \ldots, $\bar{579}$, $\bar{147}$, $\bar{258}$, $\bar{369}$, $\bar{159}$. 
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2$, $x_5$, $x_8$ are equally spaced.

- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.

- Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?

- Encode what’s forbidden: $x_2 x_5 x_8 \neq 111$ is the same as $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$.

- Writing, e.g., $\overline{258}$ for the clause $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$, we arrive at 32 clauses for the 9 digit sequence:
  123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159,
  $\overline{123}, \overline{234}, \ldots, \overline{789}, \overline{135}, \overline{246}, \ldots, \overline{579}, \overline{147}, \overline{258}, \overline{369}, \overline{159}$.
Arithmetic Progressions

Find a binary sequence \( x_1, \ldots, x_8 \) that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at \( x_2, x_5, x_8 \) are equally spaced.
- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.

Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?

- Encode what’s forbidden: \( x_2 x_5 x_8 \neq 111 \) is the same as \( (\overline{x_2} \lor \overline{x_5} \lor \overline{x_8}) \).
- Writing, e.g., \( \overline{258} \) for the clause \( (\overline{x_2} \lor \overline{x_5} \lor \overline{x_8}) \), we arrive at 32 clauses for the 9 digit sequence:
  123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159,
  \( \overline{123}, \overline{234}, \ldots, \overline{789}, \overline{135}, \overline{246}, \ldots, \overline{579}, \overline{147}, \overline{258}, \overline{369}, \overline{159} \).
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2, x_5, x_8$ are equally spaced.
- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?
  - Encode what's forbidden: $x_2x_5x_8 \neq 111$ is the same as $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$.
  - Writing, e.g., $\overline{258}$ for the clause $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$, we arrive at 32 clauses for the 9 digit sequence:
    123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159, \overline{123}, \overline{234}, \ldots, \overline{789}, \overline{135}, \overline{246}, \ldots, \overline{579}, \overline{147}, \overline{258}, \overline{369}, \overline{159}.$
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2, x_5, x_8$ are equally spaced.
- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?
- Encode what’s forbidden: $x_2 x_5 x_8 \neq 111$ is the same as $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$.
- Writing, e.g., $\overline{258}$ for the clause $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$, we arrive at 32 clauses for the 9 digit sequence:
  123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159,
  $\overline{123}, \overline{234}, \ldots, \overline{789}, \overline{135}, \overline{246}, \ldots, \overline{579}, \overline{147}, \overline{258}, \overline{369}, \overline{159}$. 
Arithmetic Progressions

Find a binary sequence $x_1, \ldots , x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2, x_5, x_8$ are equally spaced.
- 6 Solutions: 
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?
- Encode what’s forbidden: $x_2 x_5 x_8 \neq 111$ is the same as $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$.
- Writing, e.g., $\overline{258}$ for the clause $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$, we arrive at 32 clauses for the 9 digit sequence:
  123, 234, \ldots , 789, 135, 246, \ldots , 579, 147, 258, 369, 159, 
  $\overline{123}, \overline{234}, \ldots , \overline{789}, \overline{135}, \overline{246}, \ldots , \overline{579}, \overline{147}, \overline{258}, \overline{369}, \overline{159}$.
Background: Van der Waerden Numbers

Theorem (van der Waerden)

If $n$ is sufficiently large, every sequence $x_1, \ldots, x_n$ of numbers $0 \leq x_i < r$ contains a number that occurs at least $k$ times equally spaced.

- The smallest such number is the van der Waerden number $W(r, k)$.
- We have seen that $W(2, 3) = 9$.
- For larger $r, k$ the numbers are only partially known.
- E.g., $W(2, 6) = 1132$ was shown in 2008 by Kouril and Paul [6] (with the help of a SAT solver!), but $W(2, 7)$ is yet unknown.
- $2^{2^22^k+9}$ is an upper bound for $W(r, k)$ (shown in 2001 by Gowers [7]).
Graph Coloring

- McGregor graph (of order 10): planar, 110 nodes.
- Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner’s column “Mathematical Games”)
Graph Coloring

- McGregor graph (of order 10): planar, 110 nodes.
- Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner’s column “Mathematical Games”)
Graph Coloring

- McGregor graph (of order 10): planar, 110 nodes.
- Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner’s column “Mathematical Games”)
Definition: Graph coloring

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.

- SAT encoding: use $k \cdot |V|$ Boolean variables $v_j$ for $v \in V$, $1 \leq j \leq k$, where $v_j$ is true, if node $v$ gets color $j$.
- Clauses?
  - $(v_1 \lor \cdots \lor v_k)$ for $v \in V$ (“every node gets a color”)
  - $(\overline{u_j} \lor \overline{v_j})$ for $u \rightarrow v$, $1 \leq j \leq k$ (“adjacent nodes have diff. colors”)
- What about multiple colors for a node? → At-most-one constraints
Definition: Graph coloring

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.

- SAT encoding: use $k \cdot |V|$ Boolean variables $v_j$ for $v \in V$, $1 \leq j \leq k$, where $v_j$ is true, if node $v$ gets color $j$.

- Clauses?
  
  \[(v_1 \lor \cdots \lor v_k) \text{ for } v \in V \]  
  
  \[(u_j \lor v_j) \text{ for } u \rightarrow v, 1 \leq j \leq k \]  
  
  (“every node gets a color”)
  
  (“adjacent nodes have diff. colors”)

- What about multiple colors for a node? → At-most-one constraints
Definition: Graph coloring

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.

- SAT encoding: use $k \cdot |V|$ Boolean variables $v_j$ for $v \in V$, $1 \leq j \leq k$, where $v_j$ is true, if node $v$ gets color $j$.
- Clauses?
  
  $(v_1 \lor \cdots \lor v_k)$ for $v \in V$  
  (“every node gets a color”)
  
  $(u_j \lor \overline{v_j})$ for $u \rightarrow v$, $1 \leq j \leq k$  
  (“adjacent nodes have diff. colors”)

- What about multiple colors for a node? → At-most-one constraints
Graph Coloring: Encoding in SAT

Definition: Graph coloring

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.

- SAT encoding: use $k \cdot |V|$ Boolean variables $v_j$ for $v \in V$, $1 \leq j \leq k$, where $v_j$ is true, if node $v$ gets color $j$.
- Clauses?
  
  $(v_1 \lor \cdots \lor v_k)$ for $v \in V$  
  (“every node gets a color”)
  $(u_j \lor v_j)$ for $u \rightarrow v$, $1 \leq j \leq k$  
  (“adjacent nodes have diff. colors”)
- What about multiple colors for a node? $\rightarrow$ At-most-one constraints
Graph Coloring: Encoding in SAT

**Definition: Graph coloring**

Given an undirected graph \( G = (V, E) \), a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most \( k \) colors is called a \( k \)-coloring. The Graph Coloring Problem asks whether a \( k \)-coloring for \( G \) exists.

- **SAT encoding:** use \( k \cdot |V| \) Boolean variables \( v_j \) for \( v \in V \), \( 1 \leq j \leq k \), where \( v_j \) is true, if node \( v \) gets color \( j \).
- **Clauses?**
  - \((v_1 \lor \cdots \lor v_k)\) for \( v \in V \) ("every node gets a color")
  - \((\overline{u_j} \lor \overline{v_j})\) for \( u \rightarrow v \), \( 1 \leq j \leq k \) ("adjacent nodes have diff. colors")

What about multiple colors for a node? → At-most-one constraints
Graph Coloring: Encoding in SAT

Definition: Graph coloring

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.

- **SAT encoding:** use $k \cdot |V|$ Boolean variables $v_j$ for $v \in V$, $1 \leq j \leq k$, where $v_j$ is true, if node $v$ gets color $j$.

- **Clauses?**
  
  \[ (v_1 \lor \cdots \lor v_k) \text{ for } v \in V \]  
  (“every node gets a color”)

  \[ (\bar{u}_j \lor v_j) \text{ for } u \rightarrow v, 1 \leq j \leq k \]  
  (“adjacent nodes have diff. colors”)

- **What about multiple colors for a node?** → At-most-one constraints
Graph Coloring: Encoding in SAT

Definition: Graph coloring

Given an undirected graph \( G = (V, E) \), a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most \( k \) colors is called a \( k \)-coloring. The Graph Coloring Problem asks whether a \( k \)-coloring for \( G \) exists.

- SAT encoding: use \( k \cdot |V| \) Boolean variables \( v_j \) for \( v \in V, 1 \leq j \leq k \), where \( v_j \) is true, if node \( v \) gets color \( j \).
- Clauses?
  - \((v_1 \lor \cdots \lor v_k)\) for \( v \in V \) (“every node gets a color”)
  - \((\overline{u_j} \lor \overline{v_j})\) for \( u \rightarrow v, 1 \leq j \leq k \) (“adjacent nodes have diff. colors”)
- What about multiple colors for a node? → At-most-one constraints
Graph Coloring: Example

- \( V = \{u, v, w, x, y\} \)
- Colors: red (=1), green (=2), blue (=3)
- Clauses:
  - “every node gets a color”
    \[
    (u_1 \lor u_2 \lor u_3) \quad \vdots \quad (y_1 \lor y_2 \lor y_3)
    \]
  - “adjacent nodes have different colors”
    \[
    (\overline{u_1} \lor \overline{v_1}) \land \cdots \land (\overline{u_3} \lor \overline{v_3})
    \quad \vdots \quad (\overline{x_1} \lor \overline{y_1}) \land \cdots \land (\overline{x_3} \lor \overline{y_3})
    \]
Graph Coloring: Example

- \( V = \{u, v, w, x, y\} \)
- Colors: red (=1), green (=2), blue (=3)
- Clauses:
  - “every node gets a color”
    \[(u_1 \lor u_2 \lor u_3) \lor \cdots \lor (y_1 \lor y_2 \lor y_3)\]
  - “adjacent nodes have different colors”
    \[(\overline{u_1} \lor \overline{v_1}) \land \cdots \land (\overline{u_3} \lor \overline{v_3}) \lor \cdots \lor (\overline{x_1} \lor \overline{y_1}) \land \cdots \land (\overline{x_3} \lor \overline{y_3})\]
Using a SAT Solver – Input

- SAT solvers are cmdline application that take 1 argument – a text file with a formula which looks like:

```
INPUT (myformula.txt)
```

- lines starting "c" are comments and are ignored by the SAT solver
- a line starting with "p cnf" is the problem definition line containing the number of variables and clauses.
- the rest of the lines represent clauses, literals are integers (starting with variable 1), clauses are terminated by a zero.
Using a SAT Solver – Output

- calling ./mysolver myformula.txt would print on the screen something like

```plaintext
OUTPUT (./solver myformula.txt)
c comments, usually some statistics about the solving
s SATISFIABLE
v 1 2 -3 -4
v 5 -6 -7 0
```

- the solution line (starting with ”s”) can also contain UNSATISFIABLE and UNKNOWN
- the truth values of variables are printed in lines starting with ”v”, the last value is followed by a ”0”
Incremental SAT Solving

- We often need to solve a sequence of similar SAT instances
  - for example planning as sat, sokoban, bounded model checking
  - the instances share most of the clauses with their neighbors

- Can we solve these sequences of instances more efficiently?

  - What is incremental SAT solving?
    - Clauses can be added to and removed from the SAT solver

  - Why not call the solver with the new formula every time?
    - The solver can remember learned clauses and other stuff (variable scores required for heuristics)
    - (de)initialization overheads removed
Incremental SAT Solving

- We often need to solve a sequence of similar SAT instances
  - for example planning as sat, sokoban, bounded model checking
  - the instances share most of the clauses with their neighbors
- Can we solve these sequences of instances more efficiently?
- What is incremental SAT solving?
  - Clauses can be added to and removed from the SAT solver
- Why not call the solver with the new formula every time?
  - The solver can remember learned clauses and other stuff (variable scores required for heuristics)
  - (de)initialization overheads removed
Incremental SAT Solving

- We often need to solve a sequence of similar SAT instances
  - for example planning as sat, sokoban, bounded model checking
  - the instances share most of the clauses with their neighbors
- Can we solve these sequences of instances more efficiently?
- What is incremental SAT solving?
  - Clauses can be added to and removed from the SAT solver
- Why not call the solver with the new formula every time?
  - The solver can remember learned clauses and other stuff (variable scores required for heuristics)
  - (de)initialization overheads removed
Previously each SAT solver had a different incremental interface

For the 2015 SAT Race a unified interface was defined – IPASIR

IPASIR = Re-entrant Incremental Satisfiability Application Program
Interface (acronym reversed)

Currently around 10 SAT solvers are IPASIR compatible
IPASIR Overview

- Based on Lingeling incremental interface
- Clauses are added one literal at a time
  - To add \((x_1 \lor \overline{x_4})\) call `add(1); add(-4); add(0);`
- You can call a SAT solver with a set of assumptions
  - Assumptions are basically temporary unit clauses
  - Assumptions are cleared after each ”solve” call
- Clause removal is done via activation literals and assumptions
  - You must know ahead which clauses you will maybe want to remove
  - Add the clause with an additional fresh variable (activation literal)
  - example: instead of \((x_1 \lor x_2)\) add \((x_1 \lor x_2 \lor a_1)\)
  - solve with with assumption \(\overline{a_1}\) to enforce \((x_1 \lor x_2)\)
  - drop the assumption \(\overline{a_1}\) to drop \((x_1 \lor x_2)\)
Ipasir Interface

ipasir.h

const char* ipasir_signature();
void* ipasir_init();
void ipasir_release(void* solver);
void ipasir_set_terminate(void* solver, void* state,
    int (*terminate)(void* state));
void ipasir_set_learn (void * solver, void * state,
    int max_length, void (*learn)(void * state, int * clause))
void ipasir_add(void* solver, int lit_or_zero);
void ipasir_assume(void* solver, int lit);
int ipasir_solve(void* solver);
int ipasir_val(void* solver, int lit);
int ipasir_failed(void* solver, int lit);

For more details and examples of usage see http://baldur.iti.kit.edu/sat-competition-2017/downloads/ipasir.zip
IPASIR Functions

- signature – return the name and version of the solver
- init – initialize the solver, the pointer it returns is used for the rest of the functions
- add – add clauses one literal at a time
- assume – add an assumption, the assumptions are cleared after a "solve" call
- solve – solve the formula, return SAT, UNSAT or INTERRUPTED
- val – return the truth value of a variable (if solve returned SAT)
- failed – returns true if the given assumption was required for the unsatisfiability of the formula (if solver returned UNSAT)
Example – Essential Variables

- For a satisfiable formula $F$ a variable $x$ is essential if and only if $x$ has to be assigned (True or False) in each satisfying assignment of $F$.

- Task: find all the essential variables of a given formula

- How to do it:
  - use Dual Rail Encoding – for each variable $x$ add two new variables $x_P$ and $x_N$, replace each positive (negative) occurrence of $x$ with $x_P$ ($x_N$), add a clause $(\overline{x_P} \lor \overline{x_N})$ (meaning $x$ cannot be both true and false).
  - for each variable $x$ solve the formula with the assumptions $\overline{x_P}$ and $\overline{x_N}$. If the formula is UNSAT then $x$ is essential.
Example – Essential Variables – code

```c
int pdr(int var) { return 2*var; }
int ndr(int var) { return 2*var - 1; }
int dr(int lit) { return lit > 0 ? pdr(lit) : ndr(-lit); }

void Essentials(Formula f) {
    void* s = ipasir_init();
    for (int c = 0; c < f.clauses; c++) {
        for (int k = 0; k < f.clause[c].size; k++) {
            ipasir_add(s, dr(f.clause[c].lit[k]));
        }
        ipasir_add(s, 0);
    }
    for (int v = 1; v <= f.variables; v++) {
        ipasir_add(s, -pdr(v));
        ipasir_add(s, -ndr(v));
        ipasir_add(s, 0);
    }
    for (int v = 1; v <= f.variables; v++) {
        ipasir_assume(s, -pdr(v));
        ipasir_assume(s, -ndr(v));
        if (ipasir_solve(s) == 20) {
            printf("%d is Essential\n", v);
        } else {
            printf("%d is not Essential\n", v);
        }
    }
    ipasir_release(s);
}
```


References II


