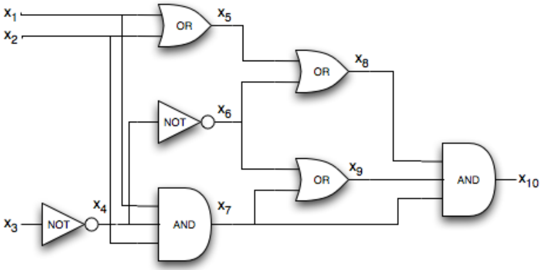
Karlsruher Institut für Technologie

Practical SAT Solving

Lecture 1

Markus Iser, Dominik Schreiber, Tomáš Balyo | April 15, 2024



Organisation

- 14 Lectures: Mondays at 3:45 pm, room 301 (starting today)
- 6 Exercises: Tuesdays at 3:45 pm, room 301 (starting 4/23, every other week!)
- Bring your laptop if you can!
- Sign up:
  - `http://campus.studium.kit.edu`
- Find material (slides, exercises, etc.):
  - `https://github.com/satlecture/kit2024`
- Interact with us (feedback, questions, announcements, etc.):
  - `https://ilias.studium.kit.edu`

Lecturers

- Markus Iser, markus.iser@kit.edu
  - post-doc at ITI Sanders, involved in this lecture since 2020
  - expert on SAT solvers and benchmarks
- Dominik Schreiber, dominik.schreiber@kit.edu
  - post-doc at ITI Sanders, involved in this lecture since 2023
  - expert on massively parallel SAT solving
- Tomáš Balyo, tomas@filuta.ai
  - previously post-doc at ITI Sanders, started this lecture in 2016 with Carsten Sinz
  - now research engineer at a composite AI start-up
  - will offer some guest lectures

Homework, Competitions, and Oral Exam

- You earn exercise points for doing homework and coming to class with your solutions.
- You can earn at least 120 exercise points during the semester (plus many more bonus points).
  - Some exercises will be in the form of small implementation contests.
  - Contest winners will receive bonus points.
- You must earn at least 60 points to participate in the oral exam.
- Bonus points for homework will improve your grade.

Goals of this Lecture

Efficient Methods for SAT Solving

Algorithms, Heuristics, Data Structures, Implementation Techniques, Parallelism, Proof Systems, ...

Goals of this Lecture

Efficient Methods for SAT Solving

Algorithms, Heuristics, Data Structures, Implementation Techniques, Parallelism, Proof Systems, ...

Applications of SAT Solving

Verification of Hardware and Software, Planning, Scheduling, Cryptography, Explainable AI, ...

Goals of this Lecture

Karlsruher Institut für Technologie

Efficient Methods for SAT Solving

Algorithms, Heuristics, Data Structures, Implementation Techniques, Parallelism, Proof Systems, . . .

Applications of SAT Solving

Verification of Hardware and Software, Planning, Scheduling, Cryptography, Explainable AI, . . .

Efficient Encodings of Problems into SAT

General Encoding Techniques, CNF Encodings of Constraints, Properties of CNF Encodings, . . .

Goals of this Lecture

Efficient Methods for SAT Solving

Algorithms, Heuristics, Data Structures, Implementation Techniques, Parallelism, Proof Systems, . . .

Applications of SAT Solving

Verification of Hardware and Software, Planning, Scheduling, Cryptography, Explainable AI, . . .

Efficient Encodings of Problems into SAT

General Encoding Techniques, CNF Encodings of Constraints, Properties of CNF Encodings, . . .

Practical Hardness of SAT

Tractable Classes, Instance Structure, Hardest Instances, Proof Complexity, . . .

Basic Definitions

In this lecture, propositional formulas are given in *conjunctive normal form* (CNF), and if not, we convert them.

CNF Formulas

- A *CNF formula* is a conjunction (and = $\wedge$) of clauses.
- A *clause* is a disjunction (or = $\vee$) of literals.
- A *literal* is a Boolean variable x (positive literal) or its negation $\overline{x}$ (negative literal).

Example (CNF Formula)

$$\begin{aligned} F &= (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (x_1) \\ \mathrm{vars}(F) &= \{x_1, x_2, x_3\} \\ \mathrm{lits}(F) &= \{x_1, \overline{x_1}, x_2, \overline{x_2}, x_3\} \\ \mathrm{clss}(F) &= \{\{\overline{x_1}, x_2\}, \{\overline{x_1}, \overline{x_2}, x_3\}, \{x_1\}\} \end{aligned}$$

Typically, a CNF formula is given as a set of clauses, where each clause is a set of literals (as in $\mathrm{clss}(F)$).

Satisfiability

The *Satisfiability Problem* is to determine whether a given formula is satisfiable.
A CNF formula F is *satisfiable* iff there exists an assignment to $\text{vars}(F)$ that satisfies F.

Satisfying Assignment

Given a CNF formula F over variables $V := \text{vars}(F)$, a *truth assignment* $\phi : V \to \{\top, \bot\}$ assigns a truth value $\top$ (True) or $\bot$ (False) to each Boolean variable in V.

We say that ϕ satisfies

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

Satisfiability

Example (Satisfiable or Unsatisfiable?)

$$F_1 = \{\{x_1\}\}$$

$$F_2 = \{\{x_1\}, \{\overline{x_1}\}\}$$

$$F_3 = \{\{x_2, x_8, \overline{x_3}\}\}$$

$$F_4 = \{\{x_1\}, \{\overline{x_2}\}, \{x_2, \overline{x_1}\}\}$$

$$F_5 = \{\{x_1, x_2\}, \{\overline{x_1}, x_2\}, \{x_1, \overline{x_2}\}, \{\overline{x_1}, \overline{x_2}\}\}$$

$$F_6 = \{\{\overline{x_1}, x_2\}, \{\overline{x_1}, \overline{x_2}, x_3\}, \{x_1\}\}$$

Satisfiability

Example (Satisfiable or Unsatisfiable?)

$$F_1 = \{\{x_1\}\}$$

$$F_2 = \{\{x_1\}, \{\overline{x_1}\}\}$$

$$F_3 = \{\{x_2, x_8, \overline{x_3}\}\}$$

$$F_4 = \{\{x_1\}, \{\overline{x_2}\}, \{x_2, \overline{x_1}\}\}$$

$$F_5 = \{\{x_1, x_2\}, \{\overline{x_1}, x_2\}, \{x_1, \overline{x_2}\}, \{\overline{x_1}, \overline{x_2}\}\}$$

$$F_6 = \{\{\overline{x_1}, x_2\}, \{\overline{x_1}, \overline{x_2}, x_3\}, \{x_1\}\}$$

Edge Cases:

What are the shortest satisfiable / unsatisfiable CNF formulas?

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering 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

$$\text{vars}(F) = \{x_1, x_2, x_3, x_4, x_5\}$$
$$F =$$

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering 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

$$\mathrm{vars}(F) = \{x_1, x_2, x_3, x_4, x_5\}$$
$$F = (x_1 \vee x_3) \wedge (\overline{x_3}) \wedge (\overline{x_5}) \wedge (x_4 \vee x_5)$$

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering 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

$$\begin{aligned} \text{vars}(F) &= \{x_1, x_2, x_3, x_4, x_5\} \\ F &= (x_1 \lor x_3) \land (\overline{x_3}) \land (\overline{x_5}) \land (x_4 \lor x_5) \\ &\quad \land \text{AtMostOne}(x_1, x_2, x_3, x_4, x_5) \end{aligned}$$

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering 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

$$
\begin{aligned}
\mathrm{vars}(F) &= \{x_1, x_2, x_3, x_4, x_5\} \\
F &= (x_1 \lor x_3) \land (\overline{x_3}) \land (\overline{x_5}) \land (x_4 \lor x_5) \\
&\quad \land (\overline{x_1} \lor \overline{x_2}) \land (\overline{x_1} \lor \overline{x_3}) \land (\overline{x_1} \lor \overline{x_4}) \land (\overline{x_1} \lor \overline{x_5}) \\
&\quad \land (\overline{x_2} \lor \overline{x_3}) \land (\overline{x_2} \lor \overline{x_4}) \land (\overline{x_2} \lor \overline{x_5}) \\
&\quad \land (\overline{x_3} \lor \overline{x_4}) \land (\overline{x_3} \lor \overline{x_5}) \land (\overline{x_4} \lor \overline{x_5})
\end{aligned}
$$

Is this Scheduling Instance Satisfiable?

Complexity of Propositional Satisfiability

A decision problem is NP-complete if it is in NP and every problem in NP can be reduced to it in polynomial time.

SAT is NP-complete (Cook-Levin Theorem)

- SAT is in NP
  Proof: solution can be checked in polynomial time
- Every problem in NP can be reduced to SAT in polynomial time
  Proof: encode the run of a non-deterministic Turing machine as a CNF formula

Complexity of Propositional Satisfiability

A decision problem is NP-complete if it is in NP and every problem in NP can be reduced to it in polynomial time.

SAT is NP-complete (Cook-Levin Theorem)

- SAT is in NP
  Proof: solution can be checked in polynomial time
- Every problem in NP can be reduced to SAT in polynomial time
  Proof: encode the run of a non-deterministic Turing machine as a CNF formula

Consequences of NP-completeness of SAT

- We do not have a polynomial algorithm for SAT (yet) :(
- If $P \neq NP$ then we will never have a polynomial algorithm for SAT :'(
- All the known NP-complete algorithms have exponential runtime in the **worst case**.

Example (Hardness)

Try it yourself: `http://www.cs.utexas.edu/~marijn/game/`

History of Propositional Satisfiability

Historic Landmarks

- 1960: DP Algorithm (first SAT solving algorithm)
- 1962: DPLL Algorithm (improving upon DP algorithm)
- 1971: SAT is NP-Complete
- 1992: Local Search Algorithm Selman et al.: A New Method for Solving Hard Satisfiability Problems
- 1992: The First International SAT Competition (followed by 1993, 1996, since 2002 every year)
- 1996: The First International SAT Conference (Workshop) (followed by 1998, since 2000 every year)
- 1999: Conflict Driven Clause Learning (CDCL) Algorithm

Advancements From 1992 to 2024, SAT solvers have improved by several orders of magnitude in terms of feasible problem size. From 100 variables and 200 clauses to 21,000,000 variables and 96,000,000 clauses.

SAT Conference 2022

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, . . .)

SAT Solving in the News

SPIEGEL ONLINE DER SPIEGEL SPIEGEL TV Anmelden

WISSENSCHAFT Schlagzeilen | Wetter | DAX 12.060,78 | TV-Programm | Abo

Nachrichten > Wissenschaft > Mensch > Mathematik > Der längste Mathe-Beweis der Welt umfasst 200 Terabyte

Zahlenrätsel

Der längste Mathe-Beweis der Welt

Drei Mathematiker haben ein Zahlenrätsel geknackt - mithilfe eines Supercomputers. Der Beweis umfasst 200 Terabyte. Sie wollen wissen, worum es geht? Okay, versuchen wir es.

Von Holger Dambeck

DPA

Supercomputer als Mathematiker

Pythagorean Triples

Karlsruher Institut für Technologie

Problem Definition

Is it possible to assign to each integer $1, 2, \dots, n$ 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

Pythagorean Triples

Problem Definition

Is it possible to assign to each integer $1, 2, \dots, n$ 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

Karlsruher Institut für Technologie

Problem Definition

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

Arithmetic Progressions

Karlsruher Institut für Technologie

Problem Definition

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

Example ($n = 8$, $k = 3$)

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?

Arithmetic Progressions

Karlsruher Institut für Technologie

Problem Definition

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

Example ($n = 8, k = 3$)

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.

Arithmetic Progressions

Problem Definition

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

Example ($n = 8$, $k = 3$)

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.

Arithmetic Progressions

Karlsruher Institut für Technologie

Problem Definition

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

Example ($n = 8$, $k = 3$)

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 remains. How can we show this with a SAT solver?

Arithmetic Progressions

Karlsruher Institut für Technologie

Problem Definition

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

Example ($n = 8, k = 3$)

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 remains. How can we show this with a SAT solver?
- Encode what's forbidden: $x_2x_5x_8 \neq 111$ is the same as $(\overline{x_2} \vee \overline{x_5} \vee \overline{x_8})$.

Arithmetic Progressions

Problem Definition

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

Example ($n = 8$, $k = 3$)

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 remains. 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 $(\overline{x_2} \vee \overline{x_5} \vee \overline{x_8})$.
- Writing, e.g., $\overline{2}\overline{5}\overline{8}$ for the clause $(\overline{x_2} \vee \overline{x_5} \vee \overline{x_8})$, we arrive at 32 clauses for the 9 digit sequence:
  $123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159,$
  $\overline{1}\overline{2}\overline{3}, \overline{2}\overline{3}\overline{4}, \ldots, \overline{7}\overline{8}\overline{9}, \overline{1}\overline{3}\overline{5}, \overline{2}\overline{4}\overline{6}, \ldots, \overline{5}\overline{7}\overline{9}, \overline{1}\overline{4}\overline{7}, \overline{2}\overline{5}\overline{8}, \overline{3}\overline{6}\overline{9}, \overline{1}\overline{5}\overline{9}.$

Background: Van der Waerden Numbers

Karlsruher Institut für Technologie

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 $W(r, k)$.
- For larger r, k the numbers are only partially known.

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 $W(r, k)$.
- For larger r, k the numbers are only partially known.

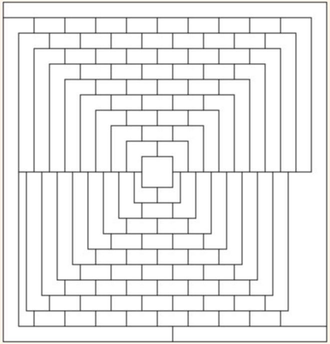
Example (Van der Waerden Numbers)

- We have seen that $W(2, 3) = 9$.
- $W(2, 6) = 1132$ was shown in [2008 by Kouril and Paul] (using a SAT solver!)
- but $W(2, 7)$ is yet unknown.
- $2^{2^{r^{2^{2^{k+9}}}}}$ is an upper bound for $W(r, k)$ (shown in [2001 by Gowers]).

Graph Coloring

Example (McGregor Graph, 110 nodes, planar)

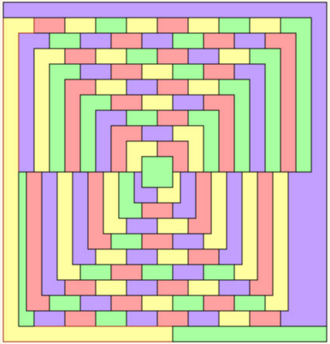
Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner's column "Mathematical Games")



Graph Coloring

Example (McGregor Graph, 110 nodes, planar)

Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner's column "Mathematical Games")



Graph Coloring: SAT Encoding

Definition: Graph Coloring Problem (GCP)

Given an undirected graph $G = (V, E)$ and a number k, a k-coloring assigns one of k colors to each node, such that all adjacent nodes have a different color. The GCP asks whether a k-coloring for G exists.

Graph Coloring: SAT Encoding

Definition: Graph Coloring Problem (GCP)

Given an undirected graph $G = (V, E)$ and a number k, a k-coloring assigns one of k colors to each node, such that all adjacent nodes have a different color. The GCP asks whether a k-coloring for G exists.

SAT Encoding

- Variables:

Graph Coloring: SAT Encoding

Karlsruher Institut für Technologie

Definition: Graph Coloring Problem (GCP)

Given an undirected graph $G = (V, E)$ and a number k, a k-coloring assigns one of k colors to each node, such that all adjacent nodes have a different color. The GCP asks whether a k-coloring for G exists.

SAT Encoding

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

Graph Coloring: SAT Encoding

Karlsruher Institut für Technologie

Definition: Graph Coloring Problem (GCP)

Given an undirected graph $G = (V, E)$ and a number k, a k-coloring assigns one of k colors to each node, such that all adjacent nodes have a different color. The GCP asks whether a k-coloring for G exists.

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

Graph Coloring: SAT Encoding

Karlsruher Institut für Technologie

Definition: Graph Coloring Problem (GCP)

Given an undirected graph $G = (V, E)$ and a number k, a k-coloring assigns one of k colors to each node, such that all adjacent nodes have a different color. The GCP asks whether a k-coloring for G exists.

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

    $(v_1 \vee \cdots \vee v_k)$ for $v \in V$

Graph Coloring: SAT Encoding

Definition: Graph Coloring Problem (GCP)

Given an undirected graph $G = (V, E)$ and a number k, a k-coloring assigns one of k colors to each node, such that all adjacent nodes have a different color. The GCP asks whether a k-coloring for G exists.

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

    $(v_1 \vee \cdots \vee v_k)$ for $v \in V$
  - Adjacent nodes have different colors:

Graph Coloring: SAT Encoding

Karlsruher Institut für Technologie

Definition: Graph Coloring Problem (GCP)

Given an undirected graph $G = (V, E)$ and a number k, a k-coloring assigns one of k colors to each node, such that all adjacent nodes have a different color. The GCP asks whether a k-coloring for G exists.

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:

    $(v_1 \vee \cdots \vee v_k)$ for $v \in V$
  - Adjacent nodes have different colors:

    $(\overline{u_j} \vee \overline{v_j})$ for $u, v \in E, 1 \leq j \leq k$

Graph Coloring: SAT Encoding

Karlsruher Institut für Technologie

Definition: Graph Coloring Problem (GCP)

Given an undirected graph $G = (V, E)$ and a number k, a k-coloring assigns one of k colors to each node, such that all adjacent nodes have a different color. The GCP asks whether a k-coloring for G exists.

SAT Encoding

- Variables:
  - use $k \cdot |V|$ Boolean variables v_j for $v \in V$, where v_j is true, if node v gets color j $(1 \leq j \leq k)$.
- Clauses:
  - Every node gets a color:
    $(v_1 \vee \cdots \vee v_k)$ for $v \in V$
  - Adjacent nodes have different colors:
    $(\overline{u_j} \vee \overline{v_j})$ for $u, v \in E, 1 \leq j \leq k$
  - Suppress multiple colors for a node: At-most-one constraints

Graph Coloring: Example

Example (Graph Coloring Problem)

- $V = \{u, v, w, x, y\}$
- Colors: red (=1), green (=2), blue (=3)
- Clauses:

"every node gets a color"

$(u_1 \vee u_2 \vee u_3)$

$\vdots$

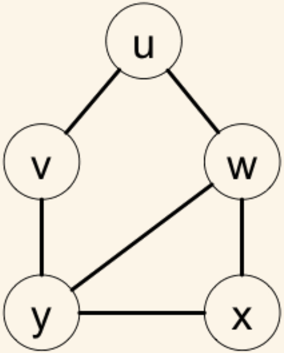
$(y_1 \vee y_2 \vee y_3)$

"adjacent nodes have different colors"

$(\overline{u_1} \vee \overline{v_1}) \wedge \cdots \wedge (\overline{u_3} \vee \overline{v_3})$

$\vdots$

$(\overline{x_1} \vee \overline{y_1}) \wedge \cdots \wedge (\overline{x_3} \vee \overline{y_3})$



Graph Coloring: Example

Karlsruher Institut für Technologie

Example (Graph Coloring Problem)

- $V = \{u, v, w, x, y\}$
- Colors: red (=1), green (=2), blue (=3)
- Clauses:

  "every node gets a color"

  $(u_1 \vee u_2 \vee u_3)$

  $\vdots$

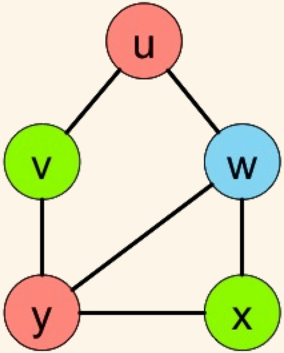
  $(y_1 \vee y_2 \vee y_3)$

  "adjacent nodes have different colors"

  $(\overline{u_1} \vee \overline{v_1}) \wedge \cdots \wedge (\overline{u_3} \vee \overline{v_3})$

  $\vdots$

  $(\overline{x_1} \vee \overline{y_1}) \wedge \cdots \wedge (\overline{x_3} \vee \overline{y_3})$



Using a SAT Solver

SAT solvers are command line applications that take as argument a text file with a formula (DIMACS format).

Example (Input)

```
c comments, ignored by solver
p cnf 7 22
1 -2 7 0
...
-7 -3 -2 0
```

Using a SAT Solver

SAT solvers are command line applications that take as argument a text file with a formula (DIMACS format).

Example (Input)

```
c comments, ignored by solver
p cnf 7 22
1 -2 7 0
...
-7 -3 -2 0
```

Example (Output)

```
c comments, usually some stastitics about the solving
s SATISFIABLE
v 1 2 -3 -4
v 5 -6 -7 0
```

Running a SAT Solver

Let's try it!

- Download and Build a SAT solver:
  - Kissat
  - CaDiCaL
  - Minisat
  - . . .
- Download a CNF formula:
  - Global Benchmark Database
- Run the SAT solver with the CNF formula as input

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?

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?

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 Library Track – IPASIR

- 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)
- IPASIR has become a standard interface of incremental SAT solving, version 2 is in the works.

IPASIR Overview

- Based on Lingeling incremental interface
- Clauses are added one literal at a time
  - To add $(x_1 \vee \overline{x_4})$ call `add(1); add(-4); add(0);`

IPASIR Overview

Karlsruher Institut für Technologie

- Based on Lingeling incremental interface
- Clauses are added one literal at a time
  - To add $(x_1 \vee \overline{x_4})$ 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 ßolve"call

IPASIR Overview

- Based on Lingeling incremental interface
- Clauses are added one literal at a time
  - To add $(x_1 \vee \overline{x_4})$ 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 ßolve"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 $(x_1 \vee x_2)$ add $(x_1 \vee x_2 \vee a_1)$
  - solve with with assumption $\overline{a_1}$ to enforce $(x_1 \vee x_2)$
  - drop the assumption $\overline{a_1}$ to drop $(x_1 \vee x_2)$

Ipasir Interface

ipasir.h

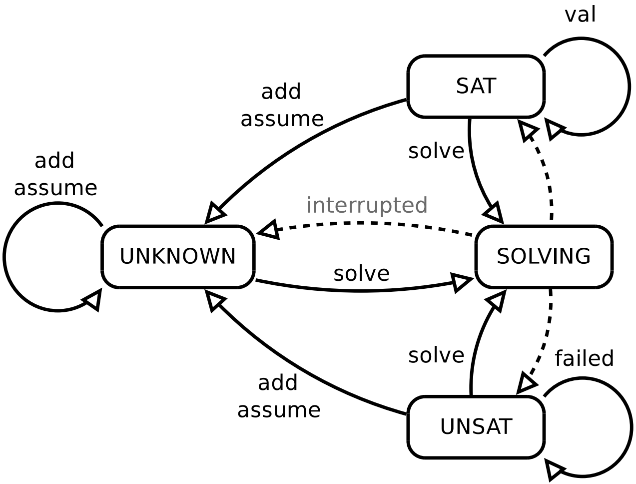
```
const char* ipasir_signature();
void* ipasir_init();
void ipasir_release(void* solver);
void ipasir_set_terminate(void* solver, void* state,
    int (*terminate)(void* state));
void ipasir_set_learn (void * solver, void * state,
    int max_length, void (*learn)(void * state, int * clause));
void 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 `https://github.com/biotomas/ipasir`

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 ßolve"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)

IPASIR Solver States



Example – Essential Variables

- For a satisfiable formula F, a variable x is *essential* if and only x it 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 $(\overline{x_P} \vee \overline{x_N})$ (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

```
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);
}
```