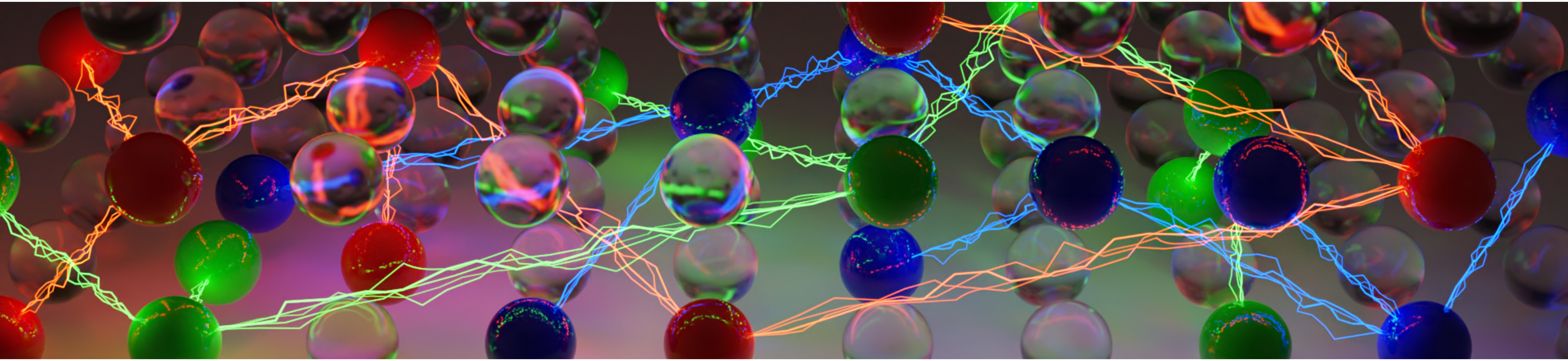


Practical SAT Solving

Lecture 1 – Organization, Introduction, Incremental SAT Solving

Markus Iser, Dominik Schreiber | April 22, 2025

Organisation

- **14 lectures:** Mondays at 3:45 pm, room 301
- **7 exercises:** Tuesdays at 3:45 pm, room 301 (starting May 6, every other week)
- Bring your laptop if you can!
- Sign up:
  - `http://campus.studium.kit.edu`
- Find material (lecture plan, slides, exercises, code) on our course website:
  - `https://satlecture.github.io/kit2025/`

Organisers

- Lecturer: **Dr. Markus Iser** – Post-doc at Algorithm Engineering group
  Contact: markus.iser@kit.edu

  Expert in Boolean Reasoning and Empirical Algorithmics

  Involved in this lecture since 2020 (guest lectures before then)

- Lecturer: **Dr. Dominik Schreiber** – Scalable Automated Reasoning (SatRes) group leader
  Contact: dominik.schreiber@kit.edu

  Expert in parallel and distributed SAT solving

  Involved in this lecture since 2023 (guest lectures before then)

- Co-manager of exercises: **Niccolò Rigi-Luperti** – Doctoral researcher @ SAtRes group
  Contact: niccolo.rigi-luperti@kit.edu

- Previous Lecturers: Prof. Carsten Sinz, Dr. Tomáš Balyo
  – Founders of the Practical SAT Solving Lecture at KIT in 2015

Homework, Competitions, and Oral Exam

- You earn exercise points for doing homework and coming to class with your solutions.
  - For each exercise, the lucky “referee” is chosen randomly among all who stated that they prepared it
  - No need to prepare a highly polished artifact! You need a clear idea and approach (+ code for practical exercises).
  - Not being able to present an exercise after stating to be able invalidates all earned points so far
- You can earn at least 120 exercise points during the semester (+ more bonus points).
  - Some exercises will be in the form of small implementation contests.
  - Contest winners will receive bonus points.
  - We encourage group work! Jointly winning a contest gets each participant an equal share of the bonus points.
- You must earn at least 60 points to participate in the oral exam.
- Earning at least 120 points will improve the grade of your passed oral exam by one step.

Goals of this Lecture

Practical knowledge and transferable skills in the following areas:

Using SAT Solving

Solver usage, general encoding techniques, efficient CNF encodings of constraints, properties of encodings, ...

Goals of this Lecture

Practical knowledge and transferable skills in the following areas:

Using SAT Solving

Solver usage, general encoding techniques, efficient CNF encodings of constraints, properties of encodings, ...

Practical Hardness of SAT

Tractable classes, instance structure, hardest instances, proof complexity, ...

Goals of this Lecture

Practical knowledge and transferable skills in the following areas:

Using SAT Solving

Solver usage, general encoding techniques, efficient CNF encodings of constraints, properties of encodings, . . .

Practical Hardness of SAT

Tractable classes, instance structure, hardest instances, proof complexity, . . .

Methods for SAT Solving

Algorithms, and data structures, heuristics, implementation techniques, parallelization, proof formats, . . .

Goals of this Lecture

Practical knowledge and transferable skills in the following areas:

Using SAT Solving

Solver usage, general encoding techniques, efficient CNF encodings of constraints, properties of encodings, . . .

Practical Hardness of SAT

Tractable classes, instance structure, hardest instances, proof complexity, . . .

Methods for SAT Solving

Algorithms, and data structures, heuristics, implementation techniques, parallelization, proof formats, . . .

Applications of SAT Solving

Hardware and software verification, planning and scheduling, security and cryptography, Explainable AI, . . .

Plan for Today

Outline

- Basic Definitions, History, and Complexity of SAT
- Applications of SAT Solving
- Examples from Mathematics and Computer Science
- Incremental SAT Solving
- Live Demo and Coding

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)

$$
\begin{aligned}
F &= (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1) \\
\text{vars}(F) &= \\
\text{lits}(F) &= \\
\text{clauses}(F) &=
\end{aligned}
$$

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)

$$\begin{aligned} F &= (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1) \\ \text{vars}(F) &= \{x_1, x_2, x_3\} \\ \text{lits}(F) &= \{x_1, \neg x_1, x_2, \neg x_2, x_3\} \\ \text{clauses}(F) &= \{\{\neg x_1, x_2\}, \{\neg x_1, \neg x_2, x_3\}, \{x_1\}\} \end{aligned}$$

Satisfiability

The *Satisfiability Problem* is to determine whether a given formula is satisfiable.

A CNF formula F is *satisfiable* iff there exists a truth assignment to vars(F) that satisfies F.

Satisfying Assignment

Given a CNF formula over variables V, a *truth assignment* $\phi : V \longrightarrow \{\top, \bot\}$ 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 $\phi(x) = \top$ (True)

... a negative literal $\neg x$ if $\phi(x) = \bot$ (False)

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

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

Example (CNF Encoding)

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

$$F = \{\{\neg x_3\},$$

$$\}$$

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering the following constraints

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

Example (CNF Encoding)

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

$$F = \{\{\neg x_3\}, \{x_1, x_3\},$$

$$\}$$

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering the following constraints

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

Example (CNF Encoding)

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

$$F = \{\{\neg x_3\}, \{x_1, x_3\}, \{\neg x_5\},$$

$$\}$$

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering the following constraints

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

Example (CNF Encoding)

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

$$F = \{\{\neg x_3\}, \{x_1, x_3\}, \{\neg x_5\}, \{x_4, x_5\},$$

$$\}$$

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering the following constraints

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

Example (CNF Encoding)

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

$$\begin{aligned} F = \{&\{\neg x_3\}, \{x_1, x_3\}, \{\neg x_5\}, \{x_4, x_5\}, \\ &\{\neg x_1, \neg x_2\}, \{\neg x_1, \neg x_3\}, \{\neg x_1, \neg x_4\}, \{\neg x_1, \neg x_5\}, \\ &\{\neg x_2, \neg x_3\}, \{\neg x_2, \neg x_4\}, \{\neg x_2, \neg x_5\}, \\ &\{\neg x_3, \neg x_4\}, \{\neg x_3, \neg x_5\}, \\ &\{\neg x_4, \neg x_5\}\} \end{aligned}$$

Satisfiability

Example (Scheduling)

Schedule a meeting of Adam, Bridget, Charles, and Darren considering the following constraints

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

Example (CNF Encoding)

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

$$\begin{aligned} F = \{&\{\neg x_3\}, \{x_1, x_3\}, \{\neg x_5\}, \{x_4, x_5\}, \\ &\{\neg x_1, \neg x_2\}, \{\neg x_1, \neg x_3\}, \{\neg x_1, \neg x_4\}, \{\neg x_1, \neg x_5\}, \\ &\{\neg x_2, \neg x_3\}, \{\neg x_2, \neg x_4\}, \{\neg x_2, \neg x_5\}, \\ &\{\neg x_3, \neg x_4\}, \{\neg x_3, \neg x_5\}, \\ &\{\neg x_4, \neg x_5\}\} \end{aligned}$$

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 algorithms for SAT have exponential runtime in the **worst case** ☹

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 algorithms for SAT have exponential runtime in the **worst case** ☹

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

History of Propositional Satisfiability

Historic Landmarks

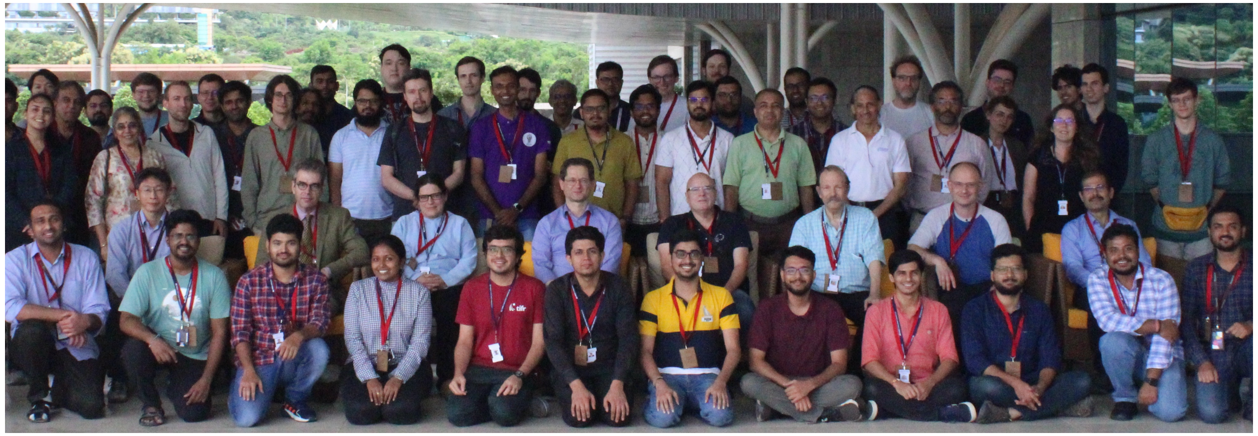
- 1960: DP Algorithm (first SAT solving algorithm)
- 1962: DPLL Algorithm (improving upon DP algorithm)
- 1971: SAT is NP-Complete
- 1992: Local Search Algorithm Selman et al.: A New Method for Solving Hard Satisfiability Problems
- 1992: The First International SAT Competition (followed by 1993, 1996, 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

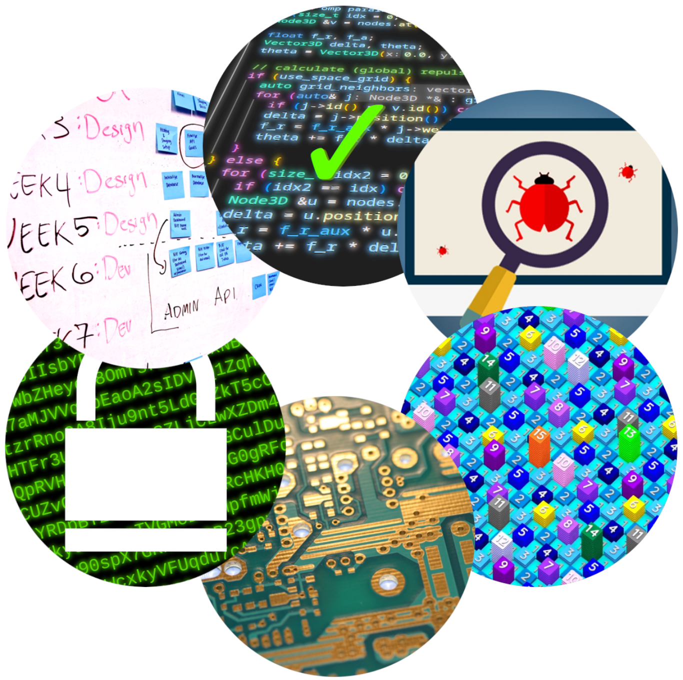
2022, Haifa, Israel



2024, Pune, India

Applications of SAT Solving

- Hardware verification and design
  - Major hardware companies (Intel, . . .) use SAT to verify chip designs
  - Computer Aided Design of electronic circuits
- Software verification
  - SAT-based SMT solvers are used to verify Microsoft software products and Amazon Web Services' core software libraries
  - Embedded software in cars, airplanes, refrigerators, . . .
  - Unix utilities
- Automated planning and scheduling
  - Job shop scheduling, train scheduling, multi-agent path finding
- Cryptanalysis
  - Test/prove properties of cryptographic ciphers, hash functions
- Number theoretic problems (Pythagorean triples, grid coloring)
- 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

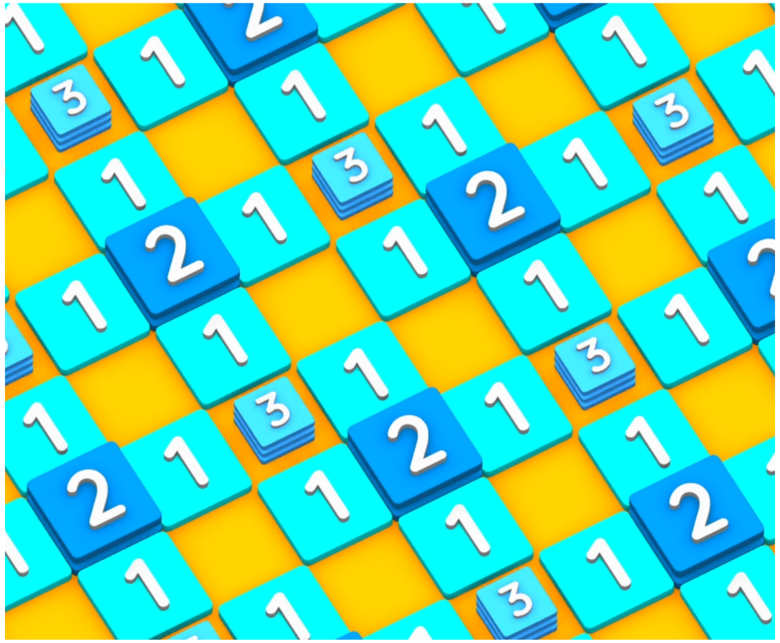
Supercomputer als Mathematiker

COMBINATORICS

The Number 15 Describes the Secret Limit of an Infinite Grid

13

The "packing coloring" problem asks how many numbers are needed to fill an infinite grid so that identical numbers never get too close to one another. A new computer-assisted proof finds a surprisingly straightforward answer.



Pythagorean Triples

Problem Definition

Can we assign each integer $1, 2, \ldots, n$ to one of two colors such that the following holds for all integers a, b, c:
If $a^2 + b^2 = c^2$, then a, b and c do not all have the same color.

- Solution: **No!** Impossible for $n = 7825$
- Proof obtained by a SAT solver has 200 Terabytes – back then the largest Math proof yet

Pythagorean Triples

Problem Definition

Can we assign each integer $1, 2, \ldots, n$ to one of two colors such that the following holds for all integers a, b, c:
If $a^2 + b^2 = c^2$, then a, b and c do not all have the same color.

- Solution: **No!** Impossible for $n = 7825$
- Proof obtained by a SAT solver has 200 Terabytes – back then the largest Math proof yet

How to encode this?

Pythagorean Triples

Problem Definition

Can we assign each integer $1, 2, \ldots, n$ to one of two colors such that the following holds for all integers a, b, c:
If $a^2 + b^2 = c^2$, then a, b and c do not all have the same color.

- Solution: **No!** Impossible for $n = 7825$
- Proof obtained by a SAT solver has 200 Terabytes – back then the largest Math proof yet

How to encode this?

- For each integer i, we have a Boolean variable x_i which represents the color of i.
- For each a, b, c such that $a^2 + b^2 = c^2$, encode two clauses: $(x_a \vee x_b \vee x_c)$ and $(\overline{x_a} \vee \overline{x_b} \vee \overline{x_c})$

Arithmetic Progressions

Problem Definition

Find a binary sequence $x_1, \ldots, x_n$ that has no k equally spaced zeroes and no k equally spaced ones.

Arithmetic Progressions

Problem Definition

Find a binary sequence $x_1, \ldots, x_n$ that has no k equally spaced zeroes and no k equally spaced ones.

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

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced zeroes and no three equally spaced ones.

- What about 01001011?

Arithmetic Progressions

Problem Definition

Find a binary sequence $x_1, \dots, x_n$ that has no k equally spaced zeroes and no k equally spaced ones.

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

Find a binary sequence $x_1, \dots, x_8$ that has no three equally spaced zeroes and no three equally spaced ones.

- What about 01001011? No, the ones 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 zeroes and no k equally spaced ones.

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

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced zeroes and no three equally spaced ones.

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

Arithmetic Progressions

Problem Definition

Find a binary sequence $x_1, \ldots, x_n$ that has no k equally spaced zeroes and no k equally spaced ones.

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

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced zeroes and no three equally spaced ones.

- What about 01001011? No, the ones at x_2, x_5, x_8 are equally spaced.
- 6 Solutions: 00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to $n = 9$ digits, no solution remains. How can we show this with a SAT solver?

Arithmetic Progressions

Problem Definition

Find a binary sequence $x_1, \ldots, x_n$ that has no k equally spaced zeroes and no k equally spaced ones.

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

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced zeroes and no three equally spaced ones.

- What about 01001011? No, the ones at x_2, x_5, x_8 are equally spaced.
- 6 Solutions: 00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to $n = 9$ digits, no solution 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 zeroes and no k equally spaced ones.

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

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced zeroes and no three equally spaced ones.

- What about 01001011? No, the ones at x_2, x_5, x_8 are equally spaced.
- 6 Solutions: 00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to $n = 9$ digits, no solution remains. How can we show this with a SAT solver?
- Encode what's forbidden: $x_2x_5x_8 \neq 111$ is the same as $(\overline{x_2} \vee \overline{x_5} \vee \overline{x_8})$.
- Writing, e.g., $\bar{2}\bar{5}\bar{8}$ for the clause $(\overline{x_2} \vee \overline{x_5} \vee \overline{x_8})$, we arrive at 32 clauses for the 9 digit sequence:
  $123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159, \bar{1}\bar{2}\bar{3}, \bar{2}\bar{3}\bar{4}, \ldots, \bar{7}\bar{8}\bar{9}, \bar{1}\bar{3}\bar{5}, \bar{2}\bar{4}\bar{6}, \ldots, \bar{5}\bar{7}\bar{9}, \bar{1}\bar{4}\bar{7}, \bar{2}\bar{5}\bar{8}, \bar{3}\bar{6}\bar{9}, \bar{1}\bar{5}\bar{9}$.

Background: Van der Waerden Numbers

Let's generalize our sequence to feature the numbers ("colors") $\{0, 1, \ldots, r-1\}$ (so far: $r = 2$).

Theorem (van der Waerden)

For any $r, k \in \mathbb{N}^+$, there exists some $n \in \mathbb{N}^+$ such that:
Every sequence $x_1, \ldots, x_n$ of numbers $0 \leq x_i < r$ has at least k equally spaced positions with the same number.

- The smallest such n is the van der Waerden number $W(r, k)$.
- For larger r, k, the numbers are only partially known.

Background: Van der Waerden Numbers

Let's generalize our sequence to feature the numbers ("colors") $\{0, 1, \ldots, r-1\}$ (so far: $r = 2$).

Theorem (van der Waerden)

For any $r, k \in \mathbb{N}^+$, there exists some $n \in \mathbb{N}^+$ such that:
Every sequence $x_1, \ldots, x_n$ of numbers $0 \leq x_i < r$ has at least k equally spaced positions with the same number.

- The smallest such n is the van der Waerden number $W(r, k)$.
- For larger r, k, the numbers are only partially known.

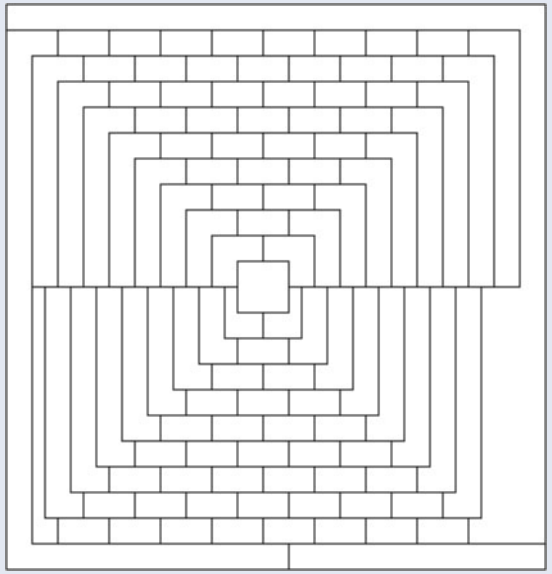
Example (Van der Waerden Numbers)

- 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)

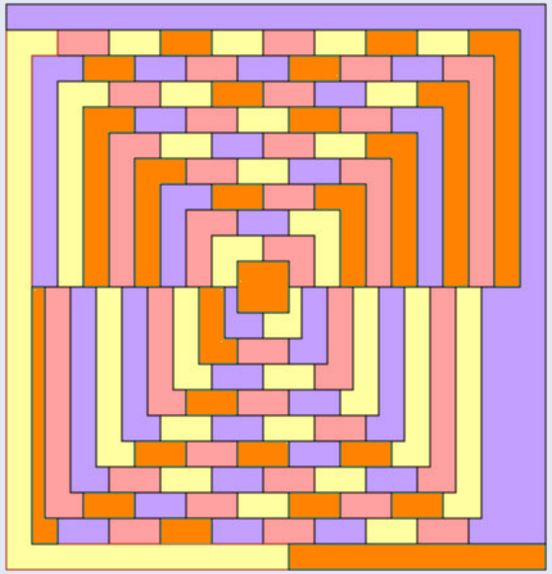
Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, M. Gardner's "Mathematical Games")



Graph Coloring

Example (McGregor Graph, 110 nodes, planar)

Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, M. Gardner's "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:

Clauses:

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 $v \in V, 1 \leq j \leq k$; v_j is true iff node v gets color j.

Clauses:

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 $v \in V, 1 \leq j \leq k$; v_j is true iff node v gets color j.

Clauses:

- Every node gets some color:

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 $v \in V, 1 \leq j \leq k$; v_j is true iff node v gets color j.

Clauses:

- Every node gets some 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 $v \in V, 1 \leq j \leq k$; v_j is true iff node v gets color j.

Clauses:

- Every node gets some color:

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

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 $v \in V, 1 \leq j \leq k$; v_j is true iff node v gets color j.

Clauses:

- Every node gets some 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

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 $v \in V, 1 \leq j \leq k$; v_j is true iff node v gets color j.

Clauses:

- Every node gets some 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"

  $(u_1 \vee u_2 \vee u_3)$

  $\vdots$

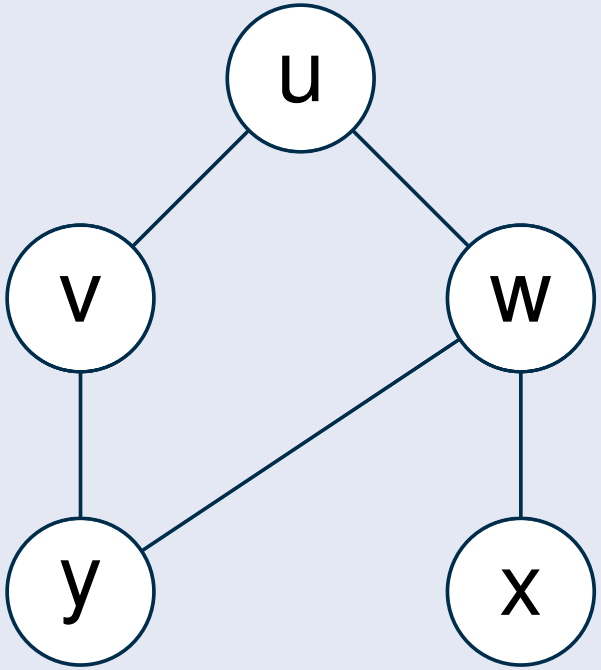
  $(y_1 \vee y_2 \vee y_3)$

  "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

Example (Graph Coloring Problem)

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

  "every node gets a color"

  $(u_1 \vee u_2 \vee u_3)$

  $\vdots$

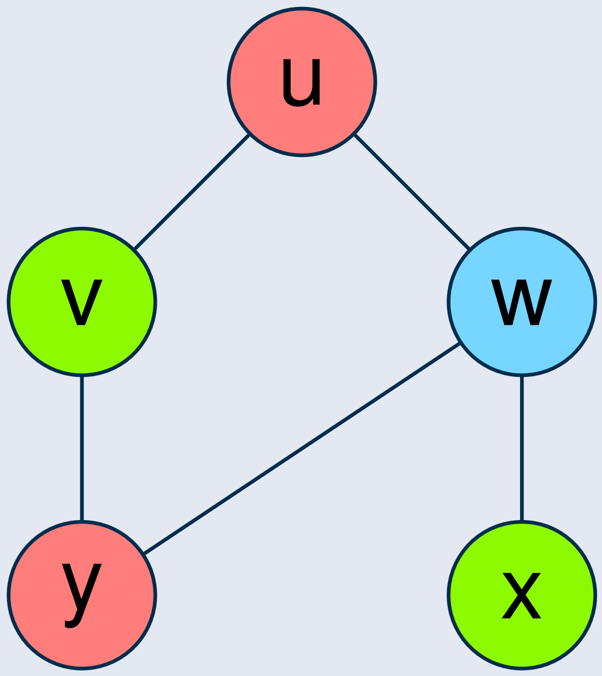
  $(y_1 \vee y_2 \vee y_3)$

  "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.

Example (Input: DIMACS CNF format)

```
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.

Example (Input: DIMACS CNF format)

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

Example (Output)

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

Running a SAT Solver

Let's try it!

1. Download and build a SAT solver:

   CaDiCaL, Kissat, Minisat, CryptoMinisat, Maplesat, . . .

   In this lecture, we use: CaDiCaL

   - Competitive performance
   - Originally written by Armin Biere
   - Highly educational, includes elaborate comments and references to the literature
   - Reference implementations of the standard SAT solver interface: IPASIR

2. Download a SAT formula:

   Global Benchmark Database

3. Determine its satisfiability

Incremental SAT Solving

There are many applications that require us to solve a sequence of similar SAT instances:

Software verification, planning, scheduling, MaxSAT, . . .

Incremental SAT Solving

- The SAT solver is initialized once with a formula
- Assumptions can be added before each call to `solve()`

  $\rightarrow$ assumptions are literals that serve as a partial assignment to their variables
- Between `solve()` calls, new clauses can be added
- Advantage 1: (De-)Initialization overheads removed
- Advantage 2: Keep preprocessed formula, learned clauses, dynamic heuristic parameters, . . .

An Incremental SAT Solver Interface: IPASIR

IPASIR = **R**e-entrant **I**ncremental **S**atisfiability **API** (acronym reversed)

IPASIR

- Defined for SAT Race 2015 to unify incremental SAT solver interfaces
- IPASIR has become a standard interface of incremental SAT solving
- Extension by User Propagators (UP): IPASIR UP

An Incremental SAT Solver Interface: IPASIR

IPASIR = **R**e-entrant **I**ncremental **S**atisfiability **API** (acronym reversed)

IPASIR

- Defined for SAT Race 2015 to unify incremental SAT solver interfaces
- IPASIR has become a standard interface of incremental SAT solving
- Extension by User Propagators (UP): IPASIR UP
- Version 2 is in the works

IPASIR Functions

Function	Description
`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 SAT)
`failed`	returns true if the given assumption was part of reason for UNSAT

For more details and examples of usage see `https://github.com/biotomas/ipasir`

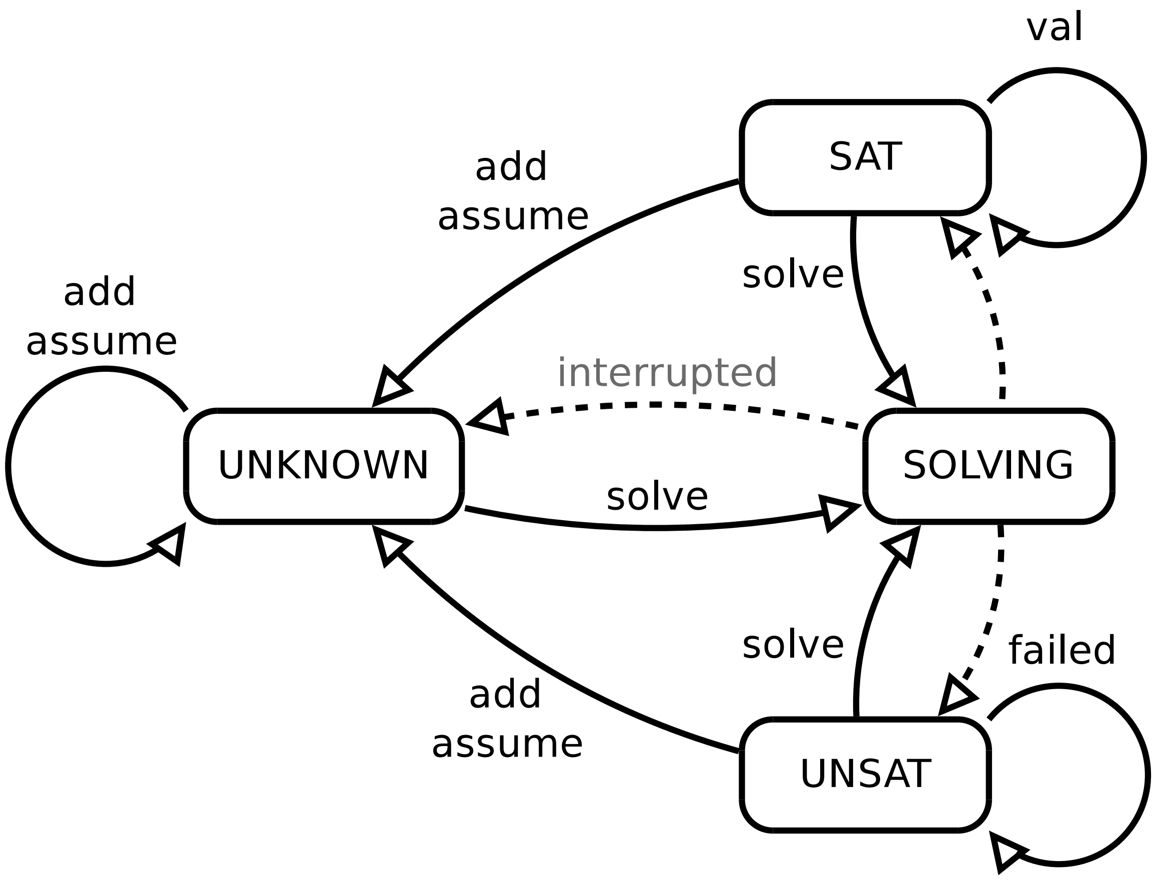
IPASIR Functions

Function	**Description**
`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 SAT)
`failed`	returns true if the given assumption was part of reason for UNSAT

For more details and examples of usage see `https://github.com/biotomas/ipasir`

How to delete a clause?

IPASIR Solver States



Code Demo: Essential Variables

Given a CNF formula F over variables $V := \text{vars}(F)$, satifying assignments to F can be partial, i.e., some variables in V are not assigned but still the formula is satisfied.

A variable x is *essential* if and only x it has to be assigned (True or False) in each satisfying assignment.

Example (Enumerating Essential Variables)

- Create the *Dual Rail Encoding* of F:
  - 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 under these assumptions then x is essential.

Let's implement it!