ECE/CS 584: Verification of Embedded and Cyberphysical systems

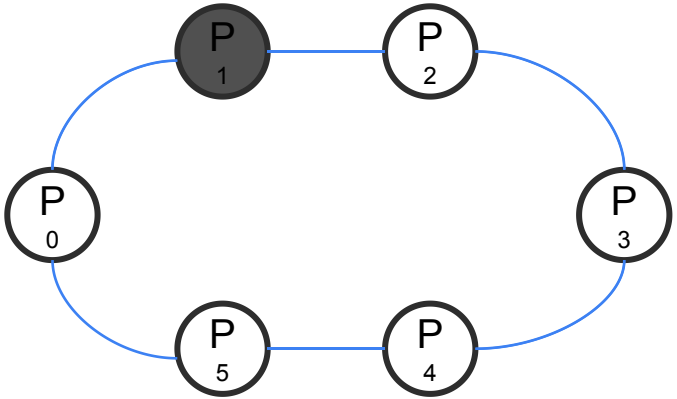
Lecture 3: Models, Satisfiability Problem

Prof. Huan Zhang

Office: CSL 262

huan@huan-zhang.com

Review: Dijkstra's mutual exclusion Algorithm



N processes: 0, 1, …, N-1

state of each process j is a single integer variable x[j] $\in$ {0, 1, 2, K-1}, where K > N

p_i has TOKEN if and only if the blue conditional below is true

The "update" action is defined differently for P0 vs. others

P_0	if x[0] = x[N-1]	then x[0] := x[0] + 1 mod K
P_j , j > 0	if x[j] ≠ x[j -1]	then x[j] := x[j-1]

Review: Automaton for Dijkstra's Token Ring

```
automaton DijkstraTR(N:Nat, K:Nat), where K > N
  type ID: enumeration [0,...,N-1]
  type Val: enumeration [0,...,K-1]
  actions
     update(i:ID)
  variables
     x:[ID -> Val]  initially forall i:ID x[i] = 0
   transitions
     update(i:ID)
       pre i = 0 /\ x[i] = x[N-1]
       eff x[i] := (x[i] + 1) %  K

      update(i:ID)
        pre i >0  /\ x[i] ~= x[i-1]
        eff x[i] := x[i-1]
```

Automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$

Review: Proving invariants by induction (Chapter 7)

Theorem 7.1. Given a automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a set of states $I \subseteq val(X)$ if:

- (Start condition) for any $\boldsymbol{x} \in \Theta$ implies $\boldsymbol{x} \in I$, and
- (Transition closure) for any $\boldsymbol{x} \rightarrow_a \boldsymbol{x}'$ and $\boldsymbol{x} \in I$ implies $\boldsymbol{x}' \in I$

then I is an (inductive) invariant of $\mathcal{A}$. That is $Reach_{\mathcal{A}}(\Theta) \subseteq I$.

Proving invariants by induction for Dijkstra

Theorem 7.1. Given a automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a set of states $I \subseteq val(X)$ if:

- (Start condition) for any $x \in \Theta$ implies $x \in I$, and
- (Transition closure) for any $x \rightarrow_a x'$ and $x \in I$ implies $x' \in I$

then I is an (inductive) invariant of $\mathcal{A}$. That is $Reach_{\mathcal{A}}(\Theta) \subseteq I$.

- I: "Exactly one process has the token".

(Start condition): since $\forall i \ \boldsymbol{x}[x[i] = 0$, only P0 has token, so $\boldsymbol{x} \vDash I$

(Transition closure): Fix a $\boldsymbol{x} \rightarrow_a \boldsymbol{x}'$ such that $\boldsymbol{x} \in I$.

Two cases to consider.

$a = update(0)$

$a = update(i)$, $i > 0$

```
automaton DijkstraTR(N:Nat, K:Nat), where
K > N
  type ID: enumeration [0,...,N-1]
  type Val: enumeration [0,...,K-1]
  actions
     update(i:ID)
  variables
     x:[ID -> Val] initially forall i:ID x[i] = 0
  transitions
     update(i:ID)
       pre i = 0 /\ x[i] = x[(N-1)]
       eff x[i] := (x[i] + 1) %  K

      update(i:ID)
        pre i >0  /\ x[i] ~= x[i-1]
        eff x[i] := x[i-1]
```

Proving invariants by induction for Dijkstra

Theorem 7.1. Given a automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a set of states $I \subseteq val(X)$ if:

- (Start condition) for any $x \in \Theta$ implies $x \in I$, and
- (Transition closure) for any $x \rightarrow_a x'$ and $x \in I$ implies $x' \in I$

then I is an (inductive) invariant of $\mathcal{A}$. That is $Reach_{\mathcal{A}}(\Theta) \subseteq I$.

- I: "Exactly one process has the token".

(**Transition closure**): Fix a $x \rightarrow_a x'$ such that $x \in I$.

Two cases to consider.

1. If $a = update(0)$ then
   - (a) since $x \models Pre(update(0))$ it follows that $x[x[0] = x[x[N-1]$
   - (b) since $x \models I$ it follows that $\forall i > 0,\ x[x[i] = x[x[i-1]$
   - (c) $x'[x[0] \neq x'[x[N-1]$ by applying (a) and $Eff(update(0))$ to x
   - (d) $x'[x[1] \neq x'[x[0]$ by applying (b) and $Eff(update(0))$ to x
   - (e) $\forall i > 1\ x'[x[i] = x'[x[i-1]$ by applying (b) and $Eff(update(0))$ to x

   Now there is only one token hold by P1. Therefore $x' \models I$.
2. If $a = update(i)$, $i > 0$ then fix arbitrary $i > 0$ … (do it as an exercise)

```
automaton DijkstraTR(N:Nat, K:Nat), where
K > N
  type ID: enumeration [0,...,N-1]
  type Val: enumeration [0,...,K-1]
  actions
     update(i:ID)
  variables
     x:[ID -> Val] initially forall i:ID x[i] = 0
  transitions
     update(i:ID)
       pre i = 0 /\ x[i] = x[(N-1)]
       eff x[i] := (x[i] + 1) %  K

     update(i:ID)
       pre i >0  /\ x[i] ~= x[i-1]
       eff x[i] := x[i-1]
```

From above **Theorem** it follows that I is an invariant of DijkstraTR

Proving invariants by induction for Dijkstra

Theorem 7.1. Given a automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a set of states $I \subseteq val(X)$ if:

- (Start condition) for any $x \in \Theta$ implies $x \in I$, and
- (Transition closure) for any $x \rightarrow_a x'$ and $x \in I$ implies $x' \in I$

then I is an (inductive) invariant of $\mathcal{A}$. That is $Reach_{\mathcal{A}}(\Theta) \subseteq I$.

- I: "Exactly one process has the token".

(**Transition closure**): Fix a $x \rightarrow_a x'$ such that $x \in I$.

Two cases to consider.

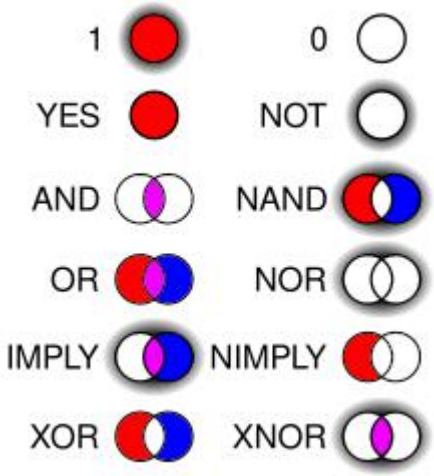
1. If $a = update(0)$ then
   - (a) since $x \models Pre(update(0))$ it follows that $x[x[0] = x[x[N-1]$
   - (b) since $x \models I$ it follows that $\forall i > 0,\ x[x[i] = x[x[i-1]$
   - (c) $x'[x[0] \neq x'[x[N-1]$ by applying (a) and $Eff(update(0))$ to x
   - (d) $x'[x[1] \neq x'[x[0]$ by applying (b) and $Eff(update(0))$ to x
   - (e) $\forall i > 1\ x'[x[i] = x'[x[i-1]$ by applying (b) and $Eff(update(0))$ to x

   Now there is only one token hold by P1. Therefore $x' \models I$.
2. If $a = update(i)$, $i > 0$ then fix arbitrary $i > 0$ … (do it as an exercise)

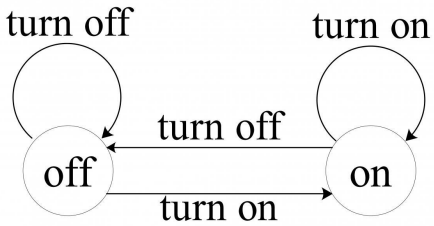
```
automaton DijkstraTR(N:Nat, K:Nat), where
K > N
  type ID: enumeration [0,...,N-1]
  type Val: enumeration [0,...,K-1]
  actions
     update(i:ID)
  variables
     x:[ID -> Val] initially forall i:ID x[i] = 0
  transitions
     update(i:ID)
       pre i = 0 /\ x[i] = x[(N-1)]
       eff x[i] := (x[i] + 1) %  K

     update(i:ID)
       pre i >0  /\ x[i] ~= x[i-1]
       eff x[i] := x[i-1]
```

Can we prove this part automatically?
Yes! Use a *satisifiability* solver! (HW1)

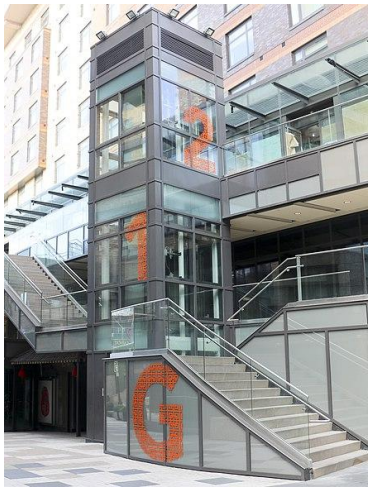
Many different types of models…



Boolean logic, propositional logic



Automaton



"Whenever the nth floor's call button is pressed, the cabin will eventually stop at the nth floor and open the door"

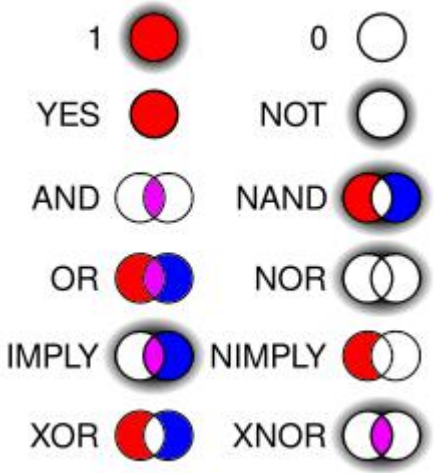
Temporal logic

Hybrid automaton

https://gcallah.github.io/DiscreteMathematics/proplogic.html

And many others…

Today: boolean logic and the satisfiability problem

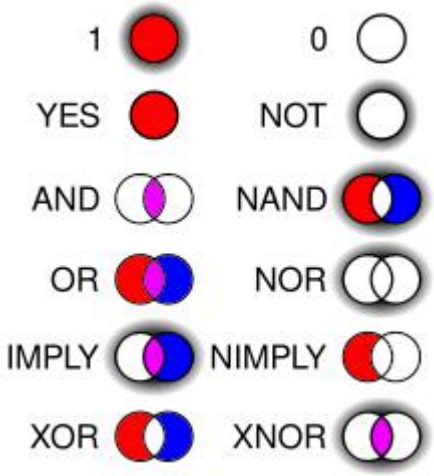


Set of **variables**: $X = \{x_1, x_2, ..., x_n\}$

Each variable is Boolean {0, 1} or {T, F}

Formula α built from propositional operators (“and” $\wedge$, “or” $\vee$, “not” $\neg$, ”implies” $\rightarrow$, “exclusive or” $\oplus$, “iff” $\leftrightarrow$)

Today: boolean logic and the satisfiability problem



Set of **variables**: $X = \{x_1, x_2, ..., x_n\}$

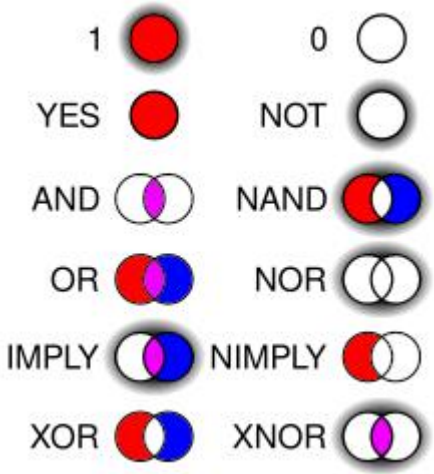
Each variable is Boolean {0, 1} or {T, F}

Formula α built from propositional operators ("and" $\wedge$, "or" $\vee$, "not" $\neg$, "implies" $\rightarrow$, "exclusive or" $\oplus$, "iff" $\leftrightarrow$)

p	q	p $\rightarrow$q
0	0	1
0	1	1
1	0	0
1	1	1

p	q	p$\oplus$ q
0	0	0
0	1	1
1	0	1
1	1	0

Today: boolean logic and the satisfiability problem



A "**valuation**" **x** of X maps each x_i to a value 0 or 1

$val(X)$ denotes the set of all possible valations

A valuation **x** of X *satisfies* α when each x_i in α replaced by the corresponding value in **x** evaluates to *true.*

We write this as $\boldsymbol{x} \vDash \alpha$; otherwise, we write $\boldsymbol{x} \nvDash \alpha$

Example:

$\alpha(x_1, x_2, \ldots, x_n) \equiv (x_1 \wedge x_2 \vee x_3) \wedge (x_1 \wedge \neg x_3 \vee x_2)$

$\boldsymbol{x} \equiv \langle x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 0 \rangle$

$\boldsymbol{x} \vDash \alpha$

Example: model a problem using boolean logic

Suppose you need to schedule 6 exams (E1, E2, ..., E6) on a single day. The day is divided into 4 time slots (S1, S2, S3, S4).
The following constraints must be satisfied:

1. Each exam must be scheduled exactly once.
2. The student conflicts are:
   - E1 and E2 cannot be at the same time
   - E3 and E4 cannot be at the same time
   - E3 and E6 cannot be at the same time
3. Due to prerequisites, E1 must be scheduled in the morning (first 2 slots) and E4 must be in the afternoon (last 2 slots)

Can you use boolean logic to represent all the constraints?

Example: model a problem using boolean logic

Suppose you need to schedule 6 exams (E1, E2, ..., E6) on a single day. The day is divided into 4 time slots (S1, S2, S3, S4).

First step: define variables x_{ij} which indicates example *i* is schedule to slot *j*

Example: model a problem using boolean logic

Suppose you need to schedule 6 exams (E1, E2, ..., E6) on a single day. The day is divided into 4 time slots (S1, S2, S3, S4).

First step: define variables x_{ij} which indicates example *i* is schedule to slot *j*

1. Each exam *i* must be scheduled exactly once.

Ensure each example is scheduled in some slot: $(x_{i1} \vee x_{i2} \vee x_{i3} \vee x_{i4})$ for all i $\in$ {1,2,3,4,5,6}
Ensure each pair of slots j and k, ensure the example is not scheduled to more than one slot:
$(\neg x_{1j} \vee \neg x_{1k})$ for all pairs of j ≠ k; j, k $\in$ {1,2,3,4}

Example: model a problem using boolean logic

Suppose you need to schedule 6 exams (E1, E2, ..., E6) on a single day. The day is divided into 4 time slots (S1, S2, S3, S4).

2. The student conflicts are:
    - E1 and E2 cannot be at the same time
    - E3 and E4 cannot be at the same time
    - E3 and E6 cannot be at the same time

Example: model a problem using boolean logic

Suppose you need to schedule 6 exams (E1, E2, ..., E6) on a single day. The day is divided into 4 time slots (S1, S2, S3, S4).

2. The student conflicts are:
   - E1 and E2 cannot be at the same time
   - E3 and E4 cannot be at the same time
   - E3 and E6 cannot be at the same time

$(\neg x_{1j} \vee \neg x_{2j})$ $(\neg x_{3j} \vee \neg x_{4j})$ $(\neg x_{3j} \vee \neg x_{6j})$ for all j

Example: model a problem using boolean logic

Suppose you need to schedule 6 exams (E1, E2, ..., E6) on a single day. The day is divided into 4 time slots (S1, S2, S3, S4).

3. Due to prerequisites, E1 must be scheduled in the morning (first 2 slots) and E4 must be in the afternoon (last 2 slots)

$(x_{11} \vee x_{12})$ $(x_{43} \vee x_{44})$

The satisfiability problem: can we find an exam schedule?

Suppose you need to schedule 6 exams (E1, E2, ..., E6) on a single day. The day is divided into 4 time slots (S1, S2, S3, S4).

$(x_{i1} \vee x_{i2} \vee x_{i3} \vee x_{i4})$ for all i $\in$ {1,2,3,4,5,6}

$(\neg x_{1j} \vee \neg x_{1k})$ for all pairs of j ≠ k; j, k $\in$ {1,2,3,4}

$(\neg x_{1j} \vee \neg x_{2j})$ $(\neg x_{3j} \vee \neg x_{4j})$ $(\neg x_{3j} \vee \neg x_{6j})$ for all j $\in$ {1,2,3,4}

$(x_{11} \vee x_{12})$ $(x_{43} \vee x_{44})$

The satisfiability problem (SAT)

Given a well-formed formula in boolean logic, determine whether there exists a satisfying solution:

$$\exists \boldsymbol{x} \in val(X) : \boldsymbol{x} \vDash \alpha ?$$

If the answer is “No” then α is said to be *unsatisfiable*

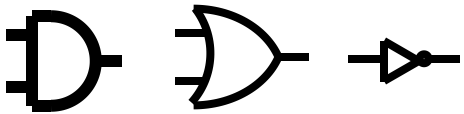
If $\forall \boldsymbol{x} \in val(X) : \boldsymbol{x} \vDash \alpha$ then α is said to be *valid* or *a tautology*

If α is valid then $\neg\alpha$ is unsatisfiable

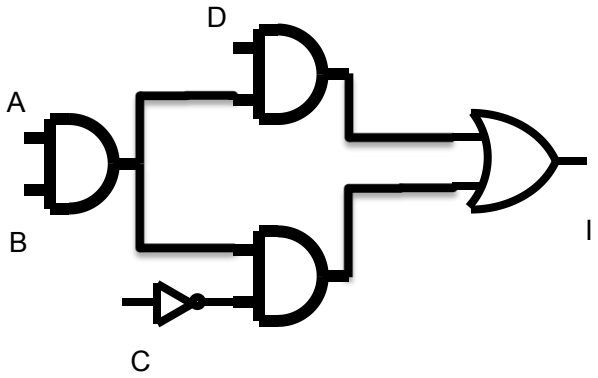
α and α' are *tautologically equivalent* if they have the same truth tables

$$\forall \boldsymbol{x} \in val(X) : \boldsymbol{x} \vDash \alpha \leftrightarrow \boldsymbol{x} \vDash \alpha'$$

Equisatisfiability



$I \equiv \big(D \wedge (A \wedge B)\big) \vee \big(\neg C \wedge (A \wedge B)\big)$



Repeated subexpression is inefficient
How about: rename $(A \wedge B) \leftrightarrow E$, reducing circuit complexity
$I' \equiv (D \wedge E) \vee (\neg C \wedge E) \wedge \big((A \wedge B) \leftrightarrow E\big)$

Recall that:

$A \leftrightarrow B$
$(A \rightarrow B) \wedge (B \rightarrow A)$
$(\neg A \vee B) \wedge (\neg B \vee A)$

I and I' are **not** *tautologically equivalent*
$C = 0,\ A = B = 1,\ E = 0$ satisfies I since I does not mention E

But they are *equisatisfiable,* i.e., I is satisfiable iff I' is also satisfiable
(there could be two different valuations to make them satisfiable)

The CNF Form

We will assume α to be in *conjunctive normal form (CNF)*

literals: variable or its negation, e.g., x_3, $\neg x_3$

clause: disjunction (or) of literals, e.g., $(x_1 \vee x_2 \vee \neg x_3)$

CNF formula: conjunction (and) of clauses,

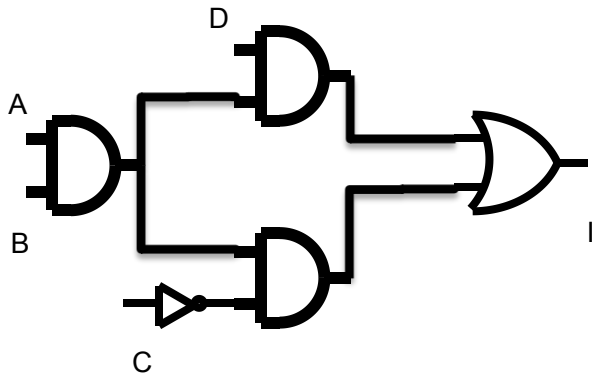
e.g., $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_1)$

A variable may appear *positively* or *negatively* in a clause

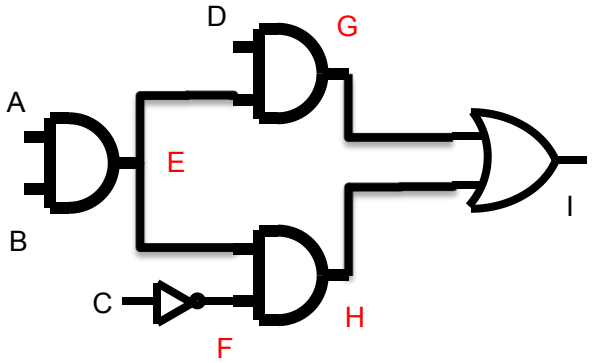
Any boolean formula can be converted to CNF

One approach:

- View the formula as a graph
- Give new names (variables) to non-leafs
- Relate the inputs and the outputs of the nonleafs and add this as a new clause
- Take conjunction of all of this



Converting to CNF



$I \equiv \big(D \wedge (A \wedge B)\big) \vee \big(\neg C \wedge (A \wedge B)\big)$

- $F \leftrightarrow \neg C$
  - $F \rightarrow \neg C \wedge \neg C \rightarrow F$
  - $(\neg F \vee \neg C) \wedge (C \vee F)$
- $(A \wedge B) \leftrightarrow E$
  - $\big((A \wedge B) \rightarrow E\big) \wedge \big(E \rightarrow (A \wedge B)\big)$
  - $(\neg(A \wedge B) \vee E) \wedge \big(\neg\, E \vee (A \wedge B)\big)$
  - $(\neg A \vee \neg B \vee E) \wedge \big((\neg\, E \vee A) \wedge (\neg E \vee B)\big)$
- $(D \wedge E) \leftrightarrow G$
  - $(\neg D \vee \neg E \vee G) \wedge (\neg\, G \vee D) \wedge (\neg G \vee E)$
- $(F \wedge E) \leftrightarrow H$
  - $(\neg F \vee \neg E \vee H) \wedge \big(\neg\, H \vee F) \wedge (\neg H \vee E)\big)$
- $(G \vee H) \leftrightarrow I$
  - $\big((G \vee H) \rightarrow I\big) \wedge \big(I \rightarrow (G \vee H)\big)$
  - $(\neg G \wedge \neg H \vee I) \wedge (\neg I \vee G \vee H)$
  - $(\neg G \vee I\,) \wedge (\neg H \vee I) \wedge (\neg I \vee G \vee H)$

Challenge: how to solve the SAT problem?

Try this one:

$(x \vee y) \wedge (\neg x \vee z) \wedge (y \vee \neg z) \wedge (\neg y \vee \neg z)$

If satisfiable, what is the valuation of x, y, z?

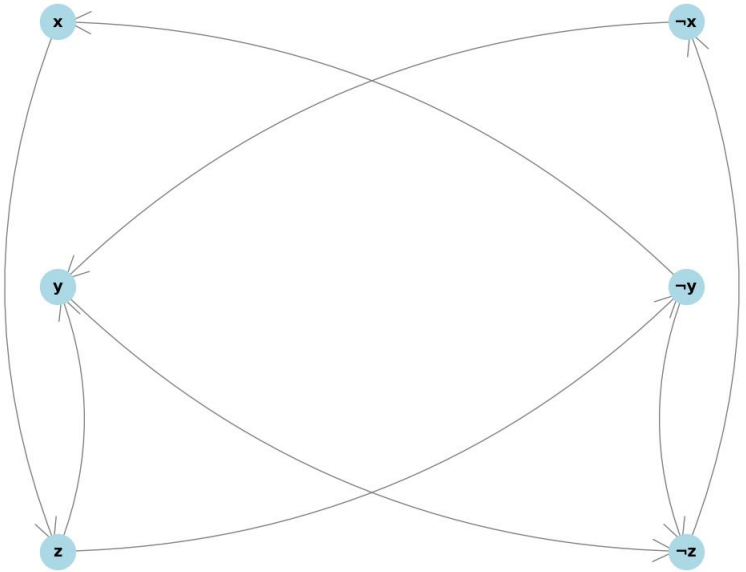
If not satisfiable, why?

2-SAT

$(x \vee y) \wedge (\neg x \vee z) \wedge (y \vee \neg z) \wedge (\neg y \vee \neg z)$

2-SAT can be solved in polynomial time

Idea is to build the **implication graph:**
e.g., $(x \vee y)$ becomes $(\neg x \rightarrow y) \wedge (\neg y \rightarrow x)$
(following the definition of implication)

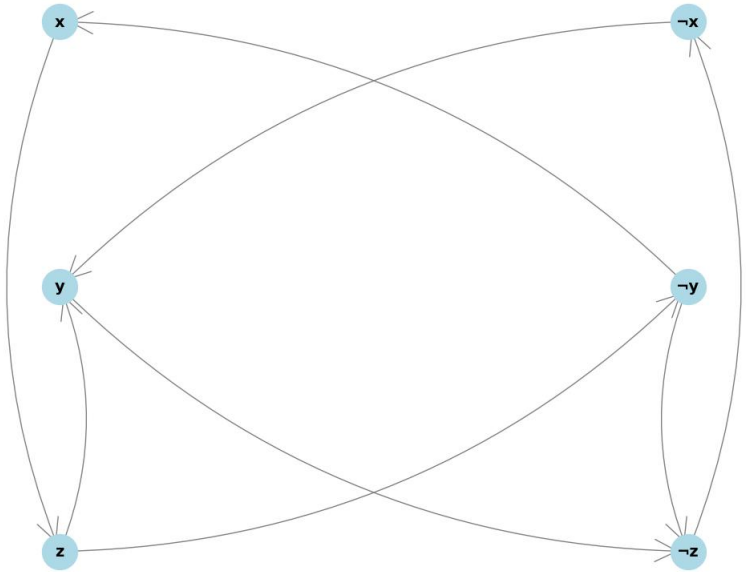


2-SAT

$(x \vee y) \wedge (\neg x \vee z) \wedge (y \vee \neg z) \wedge (\neg y \vee \neg z)$

2-SAT can be solved in polynomial time

Idea is to build the **implication graph:**
e.g., $(x \vee y)$ becomes $(\neg x \rightarrow y) \wedge (\neg y \rightarrow x)$
(following the definition of implication)

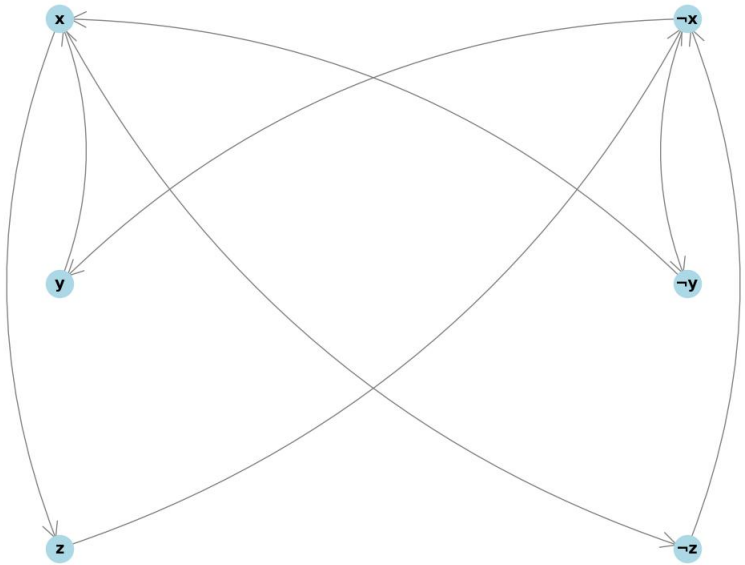


Follow the graph we have
$x \rightarrow \neg x$, $\neg y \rightarrow y$, $z \rightarrow \neg z$
How do you assign x, y, z to make them true?

2-SAT another example

$(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee z) \wedge (\neg x \vee \neg z)$

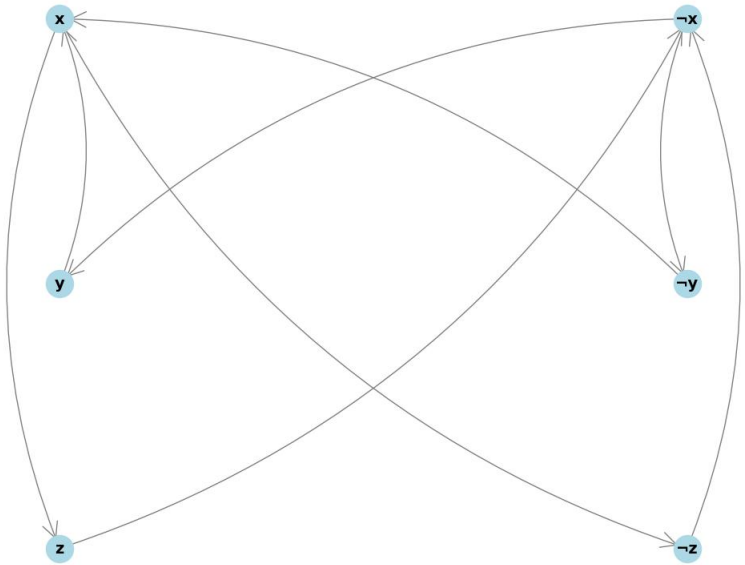
Idea is to build the **implication graph:**
e.g., $(x \vee y)$ becomes $(\neg x \rightarrow y) \wedge (\neg y \rightarrow x)$



2-SAT another example

$(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee z) \wedge (\neg x \vee \neg z)$

Idea is to build the **implication graph:**
e.g., $(x \vee y)$ becomes $(\neg x \rightarrow y) \wedge (\neg y \rightarrow x)$



Follow the graph we have
$x \rightarrow \neg x$, $\neg x \rightarrow x$
No possible assignment for x, UNSAT

3-SAT

$(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (y \vee \neg z) \wedge (\neg y \vee \neg z)$

The implication graph idea cannot be applied anymore.

Any other ideas to solve this problem?

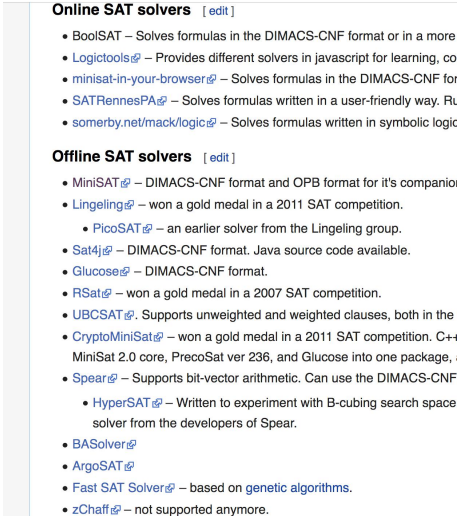
SAT is NP-complete

SAT was the first problem shown to be NP-complete [Cook 71]

2-SAT can be solved in polynomial (actually, linear) time

This has real implications

1. Essentially we don't know better than the naïve algorithm
2. A solver for SAT can be used to solve any other problem in the NP class with only polytime slowdown. i.e., makes a lot of sense to build SAT solvers
3. SAT/SMT solving is the cornerstone of *many* verification procedures



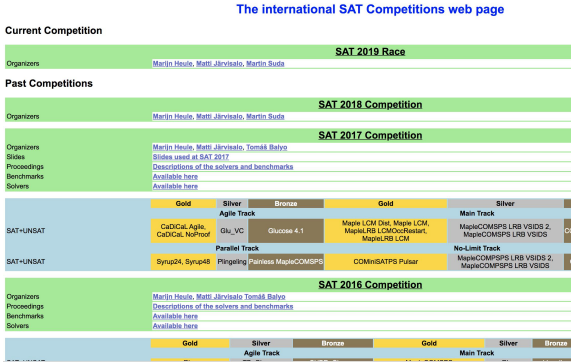
Online SAT solvers [edit]

- BoolSAT – Solves formulas in the DIMACS-CNF format or in a more
- Logictools – Provides different solvers in javascript for learning, co
- minisat-in-your-browser – Solves formulas in the DIMACS-CNF for
- SATRennesPA – Solves formulas written in a user-friendly way. Ru
- somerby.net/mack/logic – Solves formulas written in symbolic logic

Offline SAT solvers [edit]

- MiniSAT – DIMACS-CNF format and OPB format for it's companio
- Lingeling – won a gold medal in a 2011 SAT competition.
  - PicoSAT – an earlier solver from the Lingeling group.
- Sat4j – DIMACS-CNF format. Java source code available.
- Glucose – DIMACS-CNF format.
- RSat – won a gold medal in a 2007 SAT competition.
- UBCSAT. Supports unweighted and weighted clauses, both in the
- CryptoMiniSat – won a gold medal in a 2011 SAT competition. C++ MiniSat 2.0 core, PrecoSat ver 236, and Glucose into one package,
- Spear – Supports bit-vector arithmetic. Can use the DIMACS-CNF
  - HyperSAT – Written to experiment with B-cubing search space solver from the developers of Spear.
- BASolver
- ArgoSAT
- Fast SAT Solver – based on genetic algorithms.
- zChaff – not supported anymore.

thousands variables
millions of clauses
are solvable



The international SAT Competitions web page

Current Competition

SAT 2019 Race

Organizers: Marijn Heule, Matti Järvisalo, Martin Suda

Past Competitions

SAT 2018 Competition

Organizers: Marijn Heule, Matti Järvisalo, Martin Suda

SAT 2017 Competition

Organizers: Marijn Heule, Matti Järvisalo, Tomáš Balyo
Slides: Slides used at SAT 2017
Proceedings: Descriptions of the solvers and benchmarks
Benchmarks: Available here
Solvers: Available here

	Gold	Silver	Bronze	Gold	Silver
	Agile Track			Main Track	
SAT+UNSAT	CaDiCaL Agile, CaDiCaL NoProof	Glu_VC	Glucose 4.1	Maple LCM Dist, Maple LCM, MapleLRB LCMOccRestart, MapleLRB LCM	MapleCOMSPS LRB VSIDS 2, MapleCOMSPS LRB VSIDS
	Parallel Track			No-Limit Track	
SAT+UNSAT	Syrup24, Syrup48	Plingeling	Painless MapleCOMSPS	COMiniSATPS Pulsar	MapleCOMPSPS LRB VSIDS 2, MapleCOMPSPS LRB VSIDS

SAT 2016 Competition

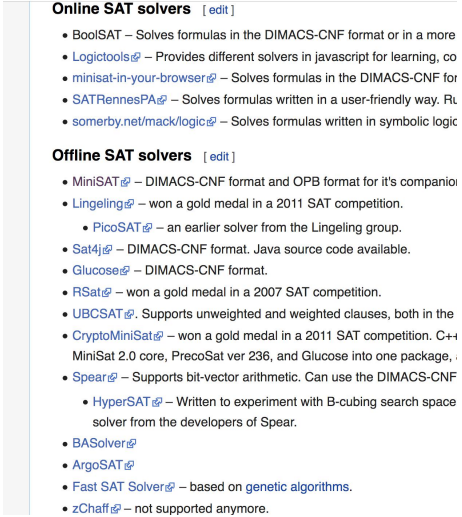
Organizers: Marijn Heule, Matti Järvisalo Tomáš Balyo
Proceedings: Descriptions of the solvers and benchmarks
Benchmarks: Available here
Solvers: Available here

	Gold	Silver	Bronze	Gold	Silver	Bronze
	Agile Track			Main Track		

Stephen Cook, The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on theory of computing. STOC '71.

SAT is NP-complete

Next lecture: solve SAT problems (DPLL algorithm)



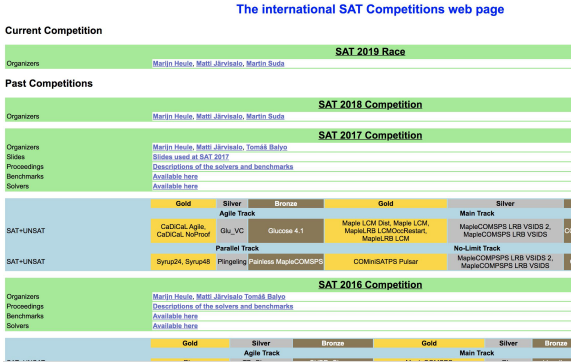
Online SAT solvers [edit]

- BoolSAT – Solves formulas in the DIMACS-CNF format or in a more
- Logictools – Provides different solvers in javascript for learning, co
- minisat-in-your-browser – Solves formulas in the DIMACS-CNF for
- SATRennesPA – Solves formulas written in a user-friendly way. Ru
- somerby.net/mack/logic – Solves formulas written in symbolic logic

Offline SAT solvers [edit]

- MiniSAT – DIMACS-CNF format and OPB format for it's companio
- Lingeling – won a gold medal in a 2011 SAT competition.
  - PicoSAT – an earlier solver from the Lingeling group.
- Sat4j – DIMACS-CNF format. Java source code available.
- Glucose – DIMACS-CNF format.
- RSat – won a gold medal in a 2007 SAT competition.
- UBCSAT. Supports unweighted and weighted clauses, both in the
- CryptoMiniSat – won a gold medal in a 2011 SAT competition. C++ MiniSat 2.0 core, PrecoSat ver 236, and Glucose into one package,
- Spear – Supports bit-vector arithmetic. Can use the DIMACS-CNF
  - HyperSAT – Written to experiment with B-cubing search space solver from the developers of Spear.
- BASolver
- ArgoSAT
- Fast SAT Solver – based on genetic algorithms.
- zChaff – not supported anymore.

thousands variables
millions of clauses
are solvable



The international SAT Competitions web page

Current Competition

SAT 2019 Race

Organizers: Marijn Heule, Matti Järvisalo, Martin Suda

Past Competitions

SAT 2018 Competition

Organizers: Marijn Heule, Matti Järvisalo, Martin Suda

SAT 2017 Competition

Organizers: Marijn Heule, Matti Järvisalo, Tomáš Balyo
Slides: Slides used at SAT 2017
Proceedings: Descriptions of the solvers and benchmarks
Benchmarks: Available here
Solvers: Available here

	Gold	Silver	Bronze	Gold	Silver
	Agile Track			Main Track	
SAT+UNSAT	CaDiCaL Agile, CaDiCaL NoProof	Glu_VC	Glucose 4.1	Maple LCM Dist, Maple LCM, MapleLRB LCMOccRestart, MapleLRB LCM	MapleCOMSPS LRB VSIDS 2, MapleCOMSPS LRB VSIDS
	Parallel Track			No-Limit Track	
SAT+UNSAT	Syrup24, Syrup48	Plingeling	Painless MapleCOMSPS	COMiniSATPS Pulsar	MapleCOMPSPS LRB VSIDS 2, MapleCOMPSPS LRB VSIDS

SAT 2016 Competition

Organizers: Marijn Heule, Matti Järvisalo Tomáš Balyo
Proceedings: Descriptions of the solvers and benchmarks
Benchmarks: Available here
Solvers: Available here

	Gold	Silver	Bronze	Gold	Silver	Bronze
	Agile Track			Main Track		

Stephen Cook, The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on theory of computing. STOC ‘71.