ECE584 L3
Invariance and Satisfiability

Verifying cyberphysical systems

Sayan Mitra

mitras@illinois.edu

Some of the slides for this lecture are adapted from slides by Clark Barrett

Readings

- Chapter 7
- Appendix C

Outline

- Propositional Satisfiability problem
- Normal forms
- DPLL algorithm

STEVEN ROSENBUSH

Meet Neurosymbolic AI, Amazon's Method for Enhancing Neural Networks

A hybrid approach to AI is powering Amazon's Rufus shopping assistant and cutting-edge warehouse robots

By Steven Rosenbush Follow

Aug. 12, 2025 11:00 am ET

Gift unlocked article

Listen (6 min)

Neurosymbolic AI has another advantage, according to Passmore: cost. Unlike pure LLM deployments requiring clusters of expensive graphics processing units, neurosymbolic agents split their workloads. They use GPUs to handle language understanding and standard CPUs to manage complex reasoning and verification.

Amazon has about 20 teams building automated reasoning in combination with other techniques for various uses across the company.

Invariants

All executions of Collatz with $x_0 \in [1,200]$ stays inside $x_i \leq 10{,}000$

"Exactly one process has a token"

$I_1: |\{i \in [N] \,|hasToken(\boldsymbol{x}, i)\}| \geq 1$

"At least one process has a token"

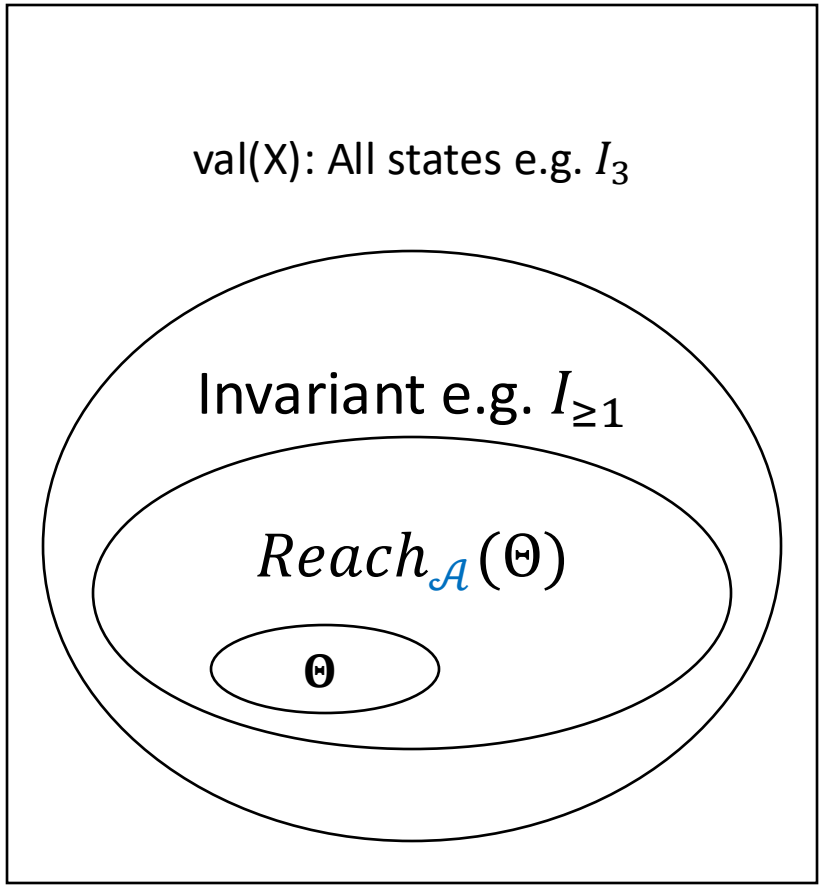
$I_{\geq 1}: |\{i \in [N] \,|hasToken(\boldsymbol{x}, i)\}| \geq 1$

"All processes have values at most K-1"

$I_3: \forall i \in [N], x[i] \leq K - 1$

"Car always stops at stop sign"

"Drone does not deviate more that 5m from ref path"



Asynchronous Nondeterministic Automaton

We will write automata precisely using this style

automaton DijkstraTR(N:$\mathbb{N}$, K: $\mathbb{N}$), **where** K > N

variables
x:[[N] -> [K]]

actions
update(i:[N])

transitions
Update(p=0)
Pre has_token(i) and i== p
Eff x[i] = (x[i] + 1) % K
Update(p>0)
Pre has_token(i) and i == p
Eff x[i] = x[i-1]

An **automaton** $\mathcal{A}$ is a 4-tuple $\langle V, \Theta, A, \mathcal{D} \rangle$ where

- $V = \{x\}$
- $type(x) = [N] \rightarrow [K]$
- **State space** $\equiv [K]^N$
- $\Theta = \cdots$
- $A = \{update(0), update(1), \ldots, update(N-1)\}$
- $\mathcal{D} = \{\langle \boldsymbol{x}, update(i), \boldsymbol{x}' \rangle \mid i \in [N] \wedge hasToken(\boldsymbol{x}, i)$
  - If $i = 0$ then $\boldsymbol{x}'[0] = \boldsymbol{x}[N-1] + 1 \,\%\, K$
  - If $i \neq 0$ then $\boldsymbol{x}'[i] = \boldsymbol{x}[i-1]\}$

This is a nondeterministic automaton.

All processes with tokens **could** perform a transition

Proving invariants by induction (Chapter 7)

Given an automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a candidate invariant $I \subseteq Val(X)$ we want to check that $Reach_{\mathcal{A}} \subseteq I$.

One approach for finite automata: Perform BFS (as in reachability analysis) and check if any state in I^c is reached

This may not scale. Alternatively:

Theorem 7.1. Given an automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a set of states $I \subseteq \mathrm{val}(X)$ 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 (Chapter 7)

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

Proof. Consider any reachable state $\boldsymbol{x}$. By the definition of a reachable state, there exists an execution α of $\mathcal{A}$ such that $\alpha.lstate = \boldsymbol{x}$.

We proceed by induction on the length α

For the base case, α consists of a single starting state $\alpha = \boldsymbol{x} \in \Theta$, and by the Start condition, $\boldsymbol{x} \in I$.

For the inductive step, $\alpha = \alpha' a\, \boldsymbol{x}$ where $a \in A$. By the induction hypothesis, we know that $\alpha'.lstate \in I$.

Invoking Transition closure on $\alpha'.lstate \rightarrow_a \boldsymbol{x}$ we obtain $\boldsymbol{x} \in I$. QED

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 $I \subseteq \mathrm{val}(X)$ 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 $Reach_{\mathcal{A}}(\Theta) \subseteq I$.

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

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

- I_1: "Exactly one process has the token".
- $I_1 \equiv x[0] = x[n-1] \ \overline{\vee}\ x[1] \neq x[0] \ \overline{\vee}\ x[2] \neq x[1] \ \ldots \overline{\vee}\ x[n-2] \neq x[n-$

(Start condition): Fix a $\boldsymbol{x} \in \Theta. \boldsymbol{x} \vDash \forall i \ \boldsymbol{x}.x[i] = 0$ therefore $\boldsymbol{x} \vDash I_1$

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

Two cases to consider.

1. If $a = update(0)$ then
   a) since $\boldsymbol{x} \vDash Pre(update(0))$ it follows that $\boldsymbol{x}.x[0] = \boldsymbol{x}.x[N-1]$
   b) since $\boldsymbol{x} \vDash I_1$ it follows that $\forall i > 0 \ \boldsymbol{x}.x[i] = \boldsymbol{x}.x[i-1]$
   c) $\boldsymbol{x}'.x[0] \neq \boldsymbol{x}'.x[N-1]$ by applying (a) and $Eff(update(0))$ to $\boldsymbol{x}$
   d) $\boldsymbol{x}'.x[1] \neq \boldsymbol{x}'.x[0]$ by applying (b) $Eff(update(0))$ to $\boldsymbol{x}$
   e) $\forall i > 1 \ \boldsymbol{x}'.x[i] = \boldsymbol{x}'.x[i-1]$ by applying (b) $Eff(update(0))$ to $\boldsymbol{x}$
   Therefore $\boldsymbol{x}' \vDash I$.
2. If $a = update(i)$, i > 0 then fix arbitrary $i > 0$ … (do it as an exercise)

From **Theorem** it follows that I_1 is an invariant of DijkstraTR

Summary

- Invariants over-approximate reachable states

- Given an automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a candidate invariant $I \subseteq Val(X)$, to check that $Reach_{\mathcal{A}} \subseteq I$.
  - A sufficient condition is to check that I satisfies start condition and transition condition of **Theorem 7.1**
  - Such an invariant is called an inductive invariant
  - Use proof failure to get I' which may be a stronger invariant, i.e., $I' \Rightarrow I$
- **All invariants are not necessarily inductive**
  - **Show that $Reach_{\mathcal{A}}$ is an inductive invariant**

Boolean *satisfiability* problem

Given a *well-formed formula* in propositional logic, determine whether there exists a satisfying solution

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

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

Each variable is Boolean: $type(x_i) = \{0,1\}$

Formula α is *well-formed* if it uses propositional operators, and $\wedge$, or $\vee$, not $\neg$, iff $\leftrightarrow$ etc., properly

Recall, a valuation **x** of X maps each x_i to a value 0 or 1

A valuation **x** of X *satisfies* α is each 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: with $\boldsymbol{x} \equiv \langle x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 0 \rangle$; $\boldsymbol{x} \vDash \alpha$

Boolean *satisfiability* problem (SAT)

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

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

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

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

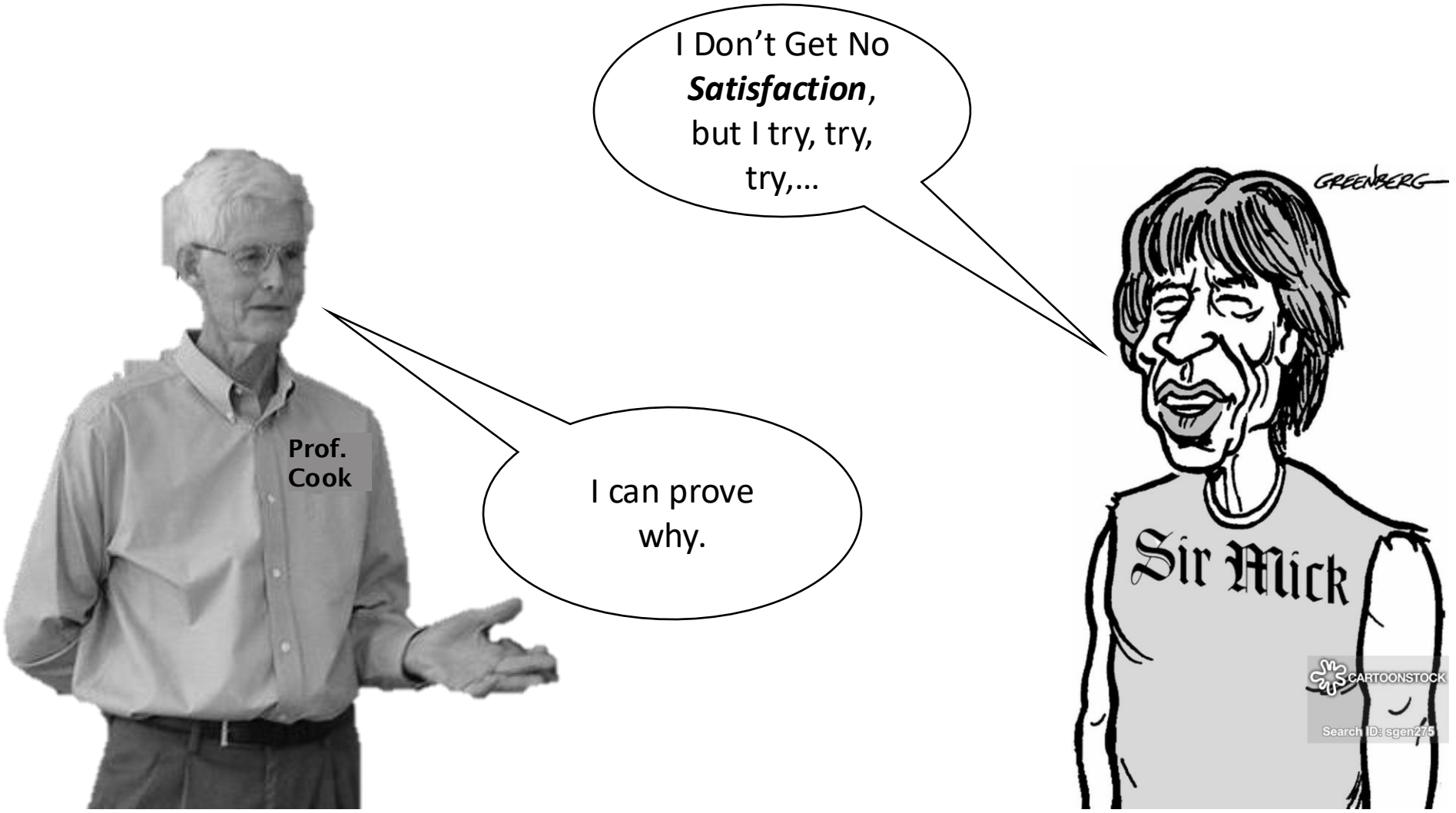
If α is valid then $\neg\alpha$ is unsatisfiable

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

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

What is a naïve method for solving SAT?

What is the complexity of this approach? How many evaluations of $\alpha(x_1, x_2, \ldots, x_n)$?



Stephen A. Cook:
The Complexity of Theorem-Proving Procedures. STOC 1971: 151-158

Slide by Sayan Mitra using pictures
from Wikipedia and cartoonstock.com

SAT is NP-complete

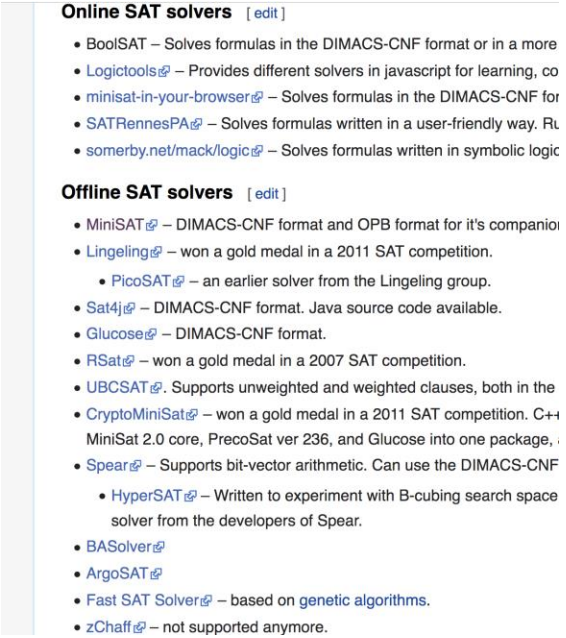
SAT was the first problem shown to be NP-complete [Cook 71]

2-SAT can be solved in polynomial time (Exercise)

(Read definition of NP: Nondeterministic Polytime in Appendix C)

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

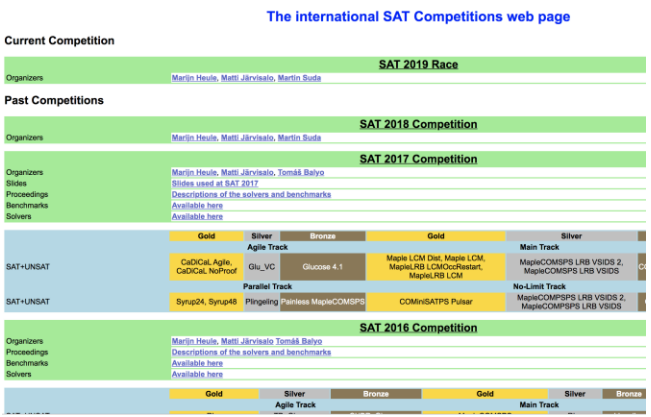


Online SAT solvers [edit]

- 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.



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 problem in detail

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

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

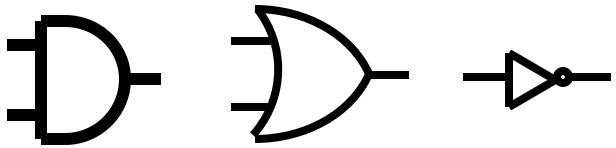
clause: disjunction (or) of literals, e.g., $(x_1 \vee x_2 \vee \neg x_3)$

CNF formula: conjunction (and) of clauses,

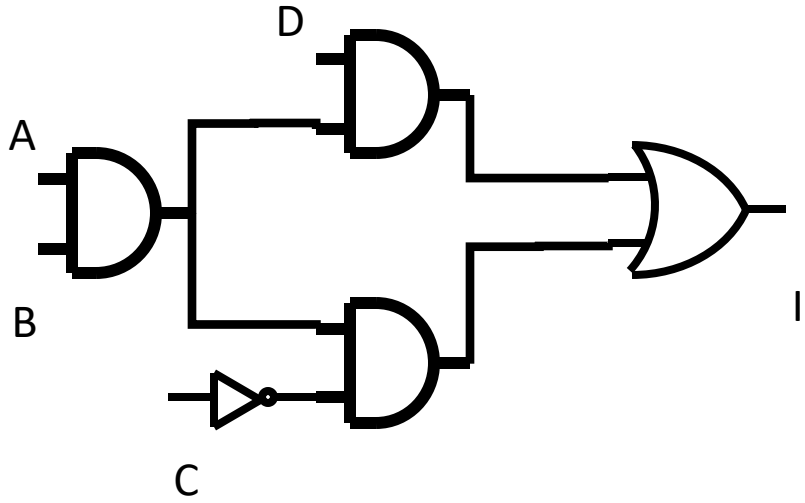
e.g., $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_1)$

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

Logic and circuits



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



Repeated subexpression (subcircuit) is inefficient

Solution: rename $(A \wedge B) \leftrightarrow E$

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

I and I' are **not** tautologically equivalent

$C = 0, A = B = 1, E = 0$ satisfies I

But they are *equisatisfiable,* i.e., I is satisfiable iff I' is also satisfiable

Assignments

- HW1
- Learn z3
  - https://ericpony.github.io/z3py-tutorial/guide-examples.htm

Readings

- Chapter 7.5.3 of CPSBook on using SAT/SMT for verification
- Read chapter 4 for next week
- Reading more about decision procedures