Principles of Safe Autonomy
Safety and Verification (ECE498)

Lecture 16-17

Sayan Mitra
University of Illinois at Urbana-Champaign
mitras@Illinois.edu
Autonomy pipeline

GEM platform

**Sensing**
Physics-based models of camera, LIDAR, RADAR, GPS, etc.

**Perception**
Programs for object detection, lane tracking, scene understanding, etc.

**Decisions and planning**
Programs and multi-agent models of pedestrians, cars, etc.

**Control**
Dynamical models of engine, powertrain, steering, tires, etc.
How to **assure safety** of the entire autonomous system?

“How to build systems that you can bet your life on?” – Jeannette Wing
Accumulating more test-driven miles will not make autonomous cars safer

- Probability of a fatality caused by an accident per one hour of (human) driving: $10^{-6}$
- Assume* that for autonomous cars this has to be: $10^{-9}$
- Data required to guarantee this probability of failure: $10^9$ hours of driving or 30 billion miles
  - 10 disengagements per 300 million miles is not enough
  - Any change to the software will require a new data collection
  - Data related issues: transparency, interpretability, and explainability
- Purely data-based approach for safety is “naïve at best”

*inspired from the fatality rate of air bags and from aviation standards
Outline

• Limitations of data-based approaches
• Safety and safety standards
  • Model-based approaches
  • Responsibility Sensitive Safety (more on this later)
• Discrete transition models
  • Examples
  • Semantics
  • Reachability
  • Invariants
• Satisfiability and SMT solvers
  • Checking invariants with SMT
Certifying airworthiness of aviation software

What fraction of the cost of developing a new aircraft is in SW?

DO178C

Primary document by which FAA & EASA approves software-based aerospace systems.

DAL establishes the rigor necessary to demonstrate compliance

<table>
<thead>
<tr>
<th>Dev. Assurance Level (DAL)</th>
<th>Hazard Classification</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Catastrophic</td>
<td>71</td>
</tr>
<tr>
<td>B</td>
<td>Hazardous</td>
<td>69</td>
</tr>
<tr>
<td>C</td>
<td>Major</td>
<td>62</td>
</tr>
<tr>
<td>D</td>
<td>Minor</td>
<td>26</td>
</tr>
<tr>
<td>E</td>
<td>No Effect</td>
<td>0</td>
</tr>
</tbody>
</table>

Statement Coverage: Every statement of the source code must be covered by a test case

Condition Coverage: Every condition within a branch statement must be covered by a test case

“Special credits”: For using formal methods based tools recently introduced
Safety Certification (cont)

• A component’s DAL level is determined from a safety assessment process and hazard analysis through examination of the effects of a failure condition in the system. The failure conditions are categorized by their effects on the aircraft, crew, and passengers.
  • For example, Level A is assigned for “Catastrophic Outcome,” and Level E is for “No Safety Effect.”

• DAL level then establishes the rigor necessary to demonstrate compliance with DO-178C

• E.g., components that command, control, or monitor safety critical functions are Level A. The standard requires any Level A software to be tested to cover every statement, branch, and function call, and also to pass the Modified Condition Decision Coverage (MC/DC) tests
  • Requires that (i) each entry and exit point in the code be invoked, (ii) each decision take every possible value, and (iii) each condition in a decision take every possible value
  • For certain levels, DO-178C requires that the testing, verification, and validation be performed by a team that is independent of the software development team

• Again, certification is process-based and does not eliminate possibility of bad logic, interference in the control code

• Dozens of commercial tools (e.g., MATLAB, Esterel, Cantata, VectorCAST, Rapita Systems, and CodeSonar) can support DO-178C certification by applying formal verification.
The formal verification problem

Example requirements:
Safety: “For all nominal behaviors of the car, the separation between the cars must be always > 1 m”
Efficiency: “For all nominal driver inputs, the air-fuel ratio must be in the range [1,4]”
Privacy: “Using GPS does not compromise user’s location”
Fairness: “Similar people’s loan approval are decided similarly”
Garden of models

Time
- Discrete transition systems, automata
  - Dynamical systems
  - Differential inclusions
  - Hybrid systems

State

Uncertainty
- Markov chains
  - Probabilistic automata, Markov decision processes (MDP)
  - Continuous time, continuous state MDPs
  - Stochastic Hybrid systems
Outline

• Limitations of data-based approaches

• Safety and safety standards
  • Model-based approaches
  • Responsibility Sensitive Safety (more on this later)

• Discrete transition models
  • Examples
  • Semantics
  • Reachability
  • Invariants

• Satisfiability and SMT solvers
  • Checking invariants with SMT
Two vehicles on a single lane

- \( a_f = a_{min} \)
- \( a_r = [a_{min}, a_{max}] \)

\( c_r, v_r, a_r \)
\( c_f, v_f, a_f \)

\( a_{min} < 0 < a_{max} \): acceleration limits
Two vehicles on a single lane: discrete time continuous state model

- $a_f[t] = a_{min}$
- $v_f[t + 1] = v_f[t] + a_f[t] \Delta$
- $c_f[t + 1] = c_f[t] + v_f[t] \Delta + \frac{1}{2} a[t] \Delta^2$

- $a_r[t] \in [a_{min}, a_{max}]$
- $v_r[t + 1] = v_r[t] + a_r[t] \Delta$
- $c_r[t + 1] = c_r[t] + v_r[t] \Delta + \frac{1}{2} a_r[t] \Delta^2$

Exercise: Update the model so that the cars never go backwards
Discrete transition systems

Discrete transition systems or **automata** is a mathematical model for describing computations or processes evolving in discrete steps.

*States* can be discrete or continuous valued.

State updates or **transitions** define how the states can change.

Transitions can be **non-deterministic**: multiple next states from a single state.

No inherent notion of *time*, but each transition can be thought of as passage of a fixed amount of time (like $\Delta$ in the previous example).

Common for describing programs, decision logics, computer hardware, circuits, control systems.
Discrete Transition System or Automaton

An **automaton** is a tuple $\mathcal{A} = \langle Q, \Theta, A, D \rangle$ where

- $Q$ is a set of states
  - Often we will find it convenient to define $Q$ by a set of variables $X$. Each variable $x \in X$ is associated with a type, $\text{type}(x)$
- $\Theta \subseteq Q$ is the set of initial or start states
- $A$ is a set of actions or labels
- $D \subseteq Q \times A \times Q$ is the set of transitions
  - a transition is a triple $(u, a, u')$
  - We write $(u, a, u') \in D$ in short as $u \xrightarrow{a} u'$
  - Enabled, pre-state, and post-state
Two vehicles on a single lane: discrete time continuous state model

- \( X = \{a_f, v_f, c_f, a_r, v_r, c_r\} \)
- \( \text{type}(a_f) = \mathbb{R} \)
- \( Q = \mathbb{R}^6 \)
- \( A = (a_1, a_2) \) the acceleration choices
- \( \mathcal{D} \subseteq \mathbb{R}^6 \times A \times \mathbb{R}^6 \)

Actually, in this example
\[
\mathcal{D}: \mathbb{R}^6 \times A \rightarrow \mathbb{R}^6
\]

It is a \textit{transition function} for any action \((a_1, a_2) \in A\) and any state \(u \in Q\) the next state \(u' = \mathcal{D}(u, a)\) is given by the function written in the previous slide.

\( c_r, v_r, a_r \quad \text{and} \quad c_f, v_f, a_f \)

\( a_{\text{min}} < 0 < a_{\text{max}} \): acceleration limits
Example 2: Informal description

A **token-based** mutual exclusion algorithm on a **ring network**

Collection of processes send and receive bits over a ring network so that only one of them has a “token”

Discrete

Each process has variables that take only **discrete values**
Time elapses in **discrete steps** (This is a modeling choice)
Token ring: Informal problem specification

1. There is always at least one token
2. Legal configuration = exactly one “token” in the ring
3. Single token circulates in the ring
4. Even if multiple tokens somehow arise, e.g. with failures, if the algorithm continues to work correctly, then eventually there is a single token
Dijkstra’s Algorithm [‘74]

N processes: 0, 1, ..., N-1
state of each process j is a single integer variable x[j] ∈ {0, 1, 2, K-1}, where K > N

\[
P_0 \quad \text{if } x[0] = x[N-1] \quad \text{then } x[0] := x[0] + 1 \text{ mod } K
\]

\[
P_j \quad j > 0 \quad \text{if } x[j] \neq x[j - 1] \quad \text{then } x[j] := x[j-1]
\]

(pi has TOKEN if and only if the blue conditional is true)
Sample executions: from a legal state (single token)

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 $K > N$

$P_0$
if $x[0] = x[N-1]$
then $x[0] := x[0] + 1 \mod K$

$P_j \; j > 0$
if $x[j] \neq x[j -1]$
then $x[j] := x[j-1]$
Execution from an illegal state

N processes: 0, 1, ..., N-1
state of each process j is a single integer variable x[j] ∈ {0, 1, 2, K-1}, where K > N

\[ P_0 \]
if \( x[0] = x[N-1] \) then \( x[0] := x[0] + 1 \mod K \)

\[ P_j \ j > 0 \]
if \( x[j] \neq x[j-1] \) then \( x[j] := x[j-1] \)
Specifying automata precisely

automaton DijkstraTR(N: Nat, K: Nat), where K > N

type ID: enumeration [0, ..., N-1]
type Val: enumeration [0, ..., K]

actions
  update(i: ID)

variables
  x: [ID -> K] initially x[i] = 0

transitions
  update(i: ID)
    pre i = 0 \& x[i] = x[(i-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]
Transitions

\( \mathcal{D} \subseteq \text{val}(X) \times A \times \text{val}(X) \) is the set of transitions

**internal update**(i:indices)

pre i = 0 \( \land \) \( x[i] = x[n-1] \)

\( \text{eff} \) \( x[i] := x[i] + 1 \pmod{k} \);

\( (u, a, u') \in \mathcal{D} \) iff \( u \models Pre_a \) and \( (u, u') \in Eff_a \)

**internal update**(i:indices)

pre i \( \neq \) 0 \( \land \) \( x[i] \neq x[i-1] \)

\( \text{eff} \) \( x[i] := x[i-1] \);

\( (u, \text{step}(i), u') \in \mathcal{D} \) iff

(a) \( i = 0 \land u.x[0] = u.x[5] \land u'.x[0] = u.x[0] + 1 \pmod{6} \) \( \lor \)

(b) \( i \neq 0 \land u.x[i] \neq u.x[i-1] \land u'.x[i] = u.x[i-1] \)
Properties can be stated as Invariants

- **Invariant** (informal def.): A property of the system that always* holds

- Examples:
  - “Always at least one process has a token”
  - “Always exactly one process has the token”
  - “Always all processes have values at most \( k-1 \)”
  - “Even if there are multiple tokens, eventually there is exactly one token” (not strictly an invariant)
Nondeterminism, pre-states and post-states

• For an action $a \in A$, Pre(a) is the formula defining its pre-condition, and Eff(a) is the relation defining the effect.

• States satisfying precondition are said to enable the action

• In general Eff(a) could be a relation, but for this example it is a function

• Nondeterminism
  • Multiple actions may be enabled from the same state
  • There may be multiple post-states from the same action
Executions, Reachability, & Invariants

An execution of $\mathcal{A}$ is an alternating (possibly infinite) sequence of states and actions

$$\alpha = u_0 a_1 u_1 a_2 u_3 \ldots$$ such that:

- $u_0 \in \Theta$
- $\forall i$ in the sequence, $u_i \xrightarrow{a_{i+1}} u_{i+1}$

A state $u$ is **reachable** if there exists an execution that ends at $u$. The set of reachable states is denoted by $\text{Reach}_\mathcal{A}$. 

Lecture Slides by Sayan Mitra mitras@illinois.edu
Invariants (Formal)

What does it mean for $I$ to hold “always” for $\mathcal{A}$?
- $I$ holds at all states along any execution $u_0a_1u_1a_2u_3$
- $I$ holds in all reachable states of $\mathcal{A}$
- $\text{Reach}_{\mathcal{A}} \subseteq [[I]]$

Invariants capture most properties that you will encounter in practice
- safety: “aircraft always maintain separation”
- bounded reaction time: “within 15 seconds of press, light must turn to walk”

How to verify if $I$ is an invariant?
- Does there exist reachable state $u$ such that $u \not\equiv I$ ?
Reachability Problem

• Given a directed graph $G = (V, E)$, and two sets of vertices $S, T \subseteq V$, $T$ is reachable from $S$ if there is a path from $S$ to $T$.

• Reachability Problem $(G, S, T)$ : decide if $T$ is reachable from $S$ in $G$. 
Algorithm for deciding Reachability G, S, T

Set Marked := {}
Queue Q := S
Marked := Marked $\cup$ S

**while** Q is not empty

\[ t \leftarrow \text{Q.dequeue()} \]

**if** $t \in T$ return "yes"

**for each** $(t, u) \in E$

\[ \text{if } u \notin \text{Marked then} \]

Marked := Marked $\cup$ \{u\}
Q := enqueue(Q, u)

**return** "no"
Verifying Invariants by solving Reachability

Given $\mathcal{A} = \langle X, \Theta, A, D \rangle$ and a candidate invariant $I$, how to check that $I$ is indeed an invariant of $\mathcal{A}$?

Define a graph $G = \langle V, E \rangle$ where

$$V = \text{val}(X)$$

$$E = \{(u, u') | \exists a \in A, u \xrightarrow{a} u' \}$$

Claim. $[[I]]^c$ is not reachable from $\Theta$ in $G$ iff $I$ is an invariant of $\mathcal{A}$. 

Lecture Slides by Sayan Mitra mitras@illinois.edu
Summary so far

• Requirement: A set of (allowed) system behaviors
• Verification: Science of checking whether a given system (model) satisfies a requirement
• Discrete transition system (automaton) models capture uncertainty in terms of sets: set of initial states, set of transitions, etc.
  • A behavior or execution is a sequence of states and actions
  • Requirements are sets of executions
  • Invariants are requirements that “always” hold
  • Reachability analysis can prove invariants
• For finite state models, reachability analysis can be performed using BFS
• For certain classes of models reachability analysis and invariant verification can be automated using Satisfiability solvers
Satisfiability

Sayan Mitra
mitras@illinois.edu
Some of the slides for this lecture are adapted from slides by Clark Barrett
Outline

• Propositional Satisfiability problem (SAT)
• Verification with SAT
• Normal forms
• DPLL algorithm
• SMT
Boolean **satisfiability** problem

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

Example: $\alpha(x_1, x_2, \ldots, x_n) \equiv (x_1 \land x_2 \lor x_3) \land (x_1 \land \neg x_3 \lor x_2)$

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

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

Formula $\alpha$ is *well-formed* if it uses propositional operators, and $\land$, or $\lor$, 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* $\alpha$ is each each $x_i$ in $\alpha$ replaced by the corresponding value in $x$ evaluates to true. We write this as $x \models \alpha$

Otherwise, we write $x \not\models \alpha$

Example: with $x \equiv (x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 0); x \models \alpha$
Boolean *satisfiability* problem (SAT)

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

Restatement: \( \exists x \in \text{val}(X): x \models \alpha \)?

If the answer is “No” then \( \alpha \) is said to be *unsatisfiable*

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

If \( \alpha \) is valid then \( \neg \alpha \) is unsatisfiable

\( \alpha \) and \( \alpha' \) are *tautologically equivalent* if they have the same truth tables

\[ \forall x \in \text{val}(X): x \models \alpha \iff x \models \alpha' \]

What is a naïve method for solving SAT?

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

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

Details

We will assume $\alpha$ 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., $(x_1 \lor x_2 \lor \neg x_3)$
- **CNF formula**: conjunction (and) of clauses,
  e.g., $(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_2 \lor x_1)$

A variable may appear *positively* or *negatively* in a clause.
Logic and circuits

Repeated subexpression is inefficient
Solution: rename \((A \land B) \leftrightarrow E\)

\[ I \equiv (D \land (A \land B)) \lor (\neg C \land (A \land B)) \]

\[ I' \equiv (D \land E) \lor (\neg C \land E) \land ((A \land B) \leftrightarrow E) \]

\(I\) 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
Standard representations of CNF

• \((\neg A \vee \neg B \vee E) \land (\neg E \vee A) \land (\neg E \vee B)\)
• \((A' + B' + E)(E' + A)(E' + B)\)
• \((-1 \ - 2 \ 5)(-5 \ 1)(-5 \ 2)\) DIMACS
• SMTLib
Verification with SAT and SMT

• We can define an automaton $\mathcal{A} = \langle Q, \Theta, A, D \rangle$ using formulas in logic (propositional logic or more expressive logics that allow integer and real valued variables)

• Example automaton

  variables $x_1, x_2$: Bool; initially $x_1 = 1$

    $\text{pre } x_1 \text{ eff } x_2 := \neg x_2$

    $\text{pre true eff } x_1 = \neg x_1$

  $\Theta(x_1, x_2) := x_1 = 1$

  $D(x_1, x_2, x_1', x_2') := (x_1 = 1 \land x_2' = \neg x_2 \land x_1' = x_1) \lor (x_1' = \neg x_1 \land x_2' = x_2)$
Verification with SAT and SMT

- Recall from Lecture 2. In order to prove that $Inv$ is an invariant of automaton $A$, we had to check:
  - (initial) For any state $x \in \Theta$, $x$ satisfies $Inv$ and
  - (transition) For any transition $(x, x') \in D$ if $x$ satisfies $Inv$ then $x'$ satisfies $Inv$

- Now that we have represented $\Theta$ and $D$ as formulas, this becomes:
  - (initial) $\forall x, \Theta(x) \Rightarrow Inv(x) \land$
  - (transition) $\forall x, x', D(x, x') \land Inv(x) \Rightarrow Inv(x')$

- That is, to verify $Inv$ we would like to check the satisfiability of the negation:
  - $\exists x, \Theta(x) \land \neg Inv(x) \lor \exists x, x', D(x, x') \land Inv(x) \land \neg Inv(x')$
Reachability Problem

• Given a directed graph $G = (V, E)$, and two sets of vertices $S, T \subseteq V$, $T$ is reachable from $S$ if there is a path from $S$ to $T$.

• Application: Check whether a bad state can be reached by an automaton $A = \langle Q, \Theta, A, D \rangle$. $V = Q$ and $E$ defined by $D$.

• Reachability Problem $(G, S, T)$: decide if $T$ is reachable from $S$ in $G$, in $k$ steps.
Exercises

- How to encode reachability (from vertex S to vertex T in a graph) as a satisfiability problem?
- Consider an automaton A with only boolean variables, and a set of unsafe states U. How can we check whether any reachable states of A within k steps are unsafe using a SAT solver?
- Solve n-queens, Sudoku using Z3.
Z3: An SMT Solver

• Z3 is an SMT solver from Microsoft Research
• [https://github.com/Z3Prover/z3](https://github.com/Z3Prover/z3)
• API in Python, C++, etc.
• Variable types: Booleans, reals, integers
• add() for adding constraints
• check() for checking satisfiability
• model() give satisfying solution

• Tutorials:
  Leonardo de Moura (slides) [http://leodemoura.github.io/slides.html](http://leodemoura.github.io/slides.html)
  [http://www.cs.tau.ac.il/~msagiv/courses/asv/z3py/fixedpoints-examples.htm](http://www.cs.tau.ac.il/~msagiv/courses/asv/z3py/fixedpoints-examples.htm)
  Stackoverflow [http://stackoverflow.com/questions/tagged/z3](http://stackoverflow.com/questions/tagged/z3)
Back to SAT: Davis Putnam Logemann Loveland Algorithm (DPLL) 1962

Transform the given formula $\alpha$ by applying a sequence of satisfiability preserving rules.

A clause is a constraint that a solution needs to satisfy.

If final result has an empty clause then *unsatisfiable*.

If final result has no clauses then the formula is *satisfiable*. 
Davis Putnam Algorithm (DP) 1960

Rule 1. Unit propagation
Rule 2. Pure literal
Rule 3. Resolution
Rule 1. **Unit propagation**
A clause has a single literal

\[ \alpha \equiv \cdots \land \cdots \land p \land \cdots \land \cdots \]

What choice do we really have?

\[ \alpha \equiv \cdots \land (x_1 \lor \neg p \lor x_2) \land p \land \cdots \land (\neg x_3 \lor \neg p \lor x_1) \cdots \]
Rule 1. **Unit propagation**

A clause has a single literal

\[ \alpha \equiv \cdots \land \cdots \land p \land \cdots \land \cdots \]

What choice do we really have?

\[ \alpha' \equiv \cdots \land (x_1 \lor x_2) \land \cdots \land (\neg x_3 \lor x_1) \cdots \]

\( \alpha \) and \( \alpha' \) are equisatisfiable
Rule 1. **Unit propagation**

Rule 2. **Pure literal**

A literal appears only positively (or negatively) in $\alpha$

$$\alpha \equiv \cdots \land (x_1 \lor \neg p \lor x_2) \land (x_4 \lor \neg p) \land \cdots \land (\neg x_3 \lor \neg p \lor x_1) \cdots$$

$p$ does not appear anywhere

Makes sense to set $p = 0$ and remove all occurrences of $\neg p$
Rule 1. **Unit propagation**

Rule 2. **Pure literal**

A literal appears only positively (or negatively) in $\alpha$

$$\alpha \equiv \cdots \land (x_1 \lor \neg p \lor x_2) \land (x_4 \lor \neg p) \land \cdots \land (\neg x_3 \lor x_1) \cdots$$

$p$ does not appear anywhere

Makes sense to set $p = 0$ and remove all clauses in which $\neg p$ occurs

$\alpha$ and $\alpha'$ are equisatisfiable

$$\alpha' \equiv \cdots \land \cdots \land \cdots \land (\neg x_3 \lor x_1) \cdots \ [p = 0]$$
Davis Putnam Algorithm (DP) 1960

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

Rule 3. **Resolution**

Choose a literal $p$ that appears with both polarity in $\alpha$. Suppose $(\ell_1 \lor \ell_2 \lor \cdots \lor p)$ be a clause in which $p$ appears positively, and $(k_1 \lor k_2 \lor \cdots \lor \neg p)$ be a clause in which $p$ appears negatively

Then the resolved clause is $(\ell_1 \lor \ell_2 \lor \cdots \lor k_1 \lor k_2 \lor \cdots k_m)$

Pairwise, resolve each clause in which $p$ appears positively with a clause in which $p$ appears negatively, and take the conjunction of all the results

**Why is the result equisatisfiable?**

**What is the size of the resulting formula?**
DPLL modifies resolution in DP with recursive DFS rule

Rule 1. **Unit propagation**

Rule 2. **Pure literal**

Rule 3’. Let $\Delta$ be the current set of clauses. Choose a literal $p$ in $\Delta$.

Check satisfiability of $\Delta \cup \{ p \}$ (guessing $p = 1$)

If satisfiable then return True else

return result of checking satisfiability of $\Delta \cup \{ \neg p \}$

This is essentially a depth first search
A simple greedy algorithm for SAT (GSAT)

Input: Set of clauses $C$ over $X$, parameters $max$-flips, $max$-tries
Output: A satisfying assignment for $C$, or ∅ if none found

for $i = 1$ to $max$-tries
    $v :=$ random truth assignment in $val(X)$
    for $j = 1$ to $max$-flips
        if $v \vDash C$ then return $v$
        $p :=$ variable in $C$ such that flipping its value gives the largest increase in the number of clauses of $C$ that are satisfied by $v$
        $v := v$ with the assignment to $p$ flipped
    return ∅
<table>
<thead>
<tr>
<th>Problem</th>
<th>tautology</th>
<th>dptaut</th>
<th>dplltaut</th>
</tr>
</thead>
<tbody>
<tr>
<td>prime 3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>prime 4</td>
<td>0.02</td>
<td>0.06</td>
<td>0.04</td>
</tr>
<tr>
<td>prime 9</td>
<td>18.94</td>
<td>2.98</td>
<td>0.51</td>
</tr>
<tr>
<td>prime 10</td>
<td>11.40</td>
<td>3.03</td>
<td>0.96</td>
</tr>
<tr>
<td>prime 11</td>
<td>28.11</td>
<td>2.98</td>
<td>0.51</td>
</tr>
<tr>
<td>prime 16</td>
<td>&gt;1 hour</td>
<td>out of memory</td>
<td>9.15</td>
</tr>
<tr>
<td>prime 17</td>
<td>&gt;1 hour</td>
<td>out of memory</td>
<td>3.87</td>
</tr>
<tr>
<td>ramsey 3 3 5</td>
<td>0.03</td>
<td>0.06</td>
<td>0.02</td>
</tr>
<tr>
<td>ramsey 3 3 6</td>
<td>5.13</td>
<td>8.28</td>
<td>0.31</td>
</tr>
<tr>
<td>mk_adder_test 3 2</td>
<td>&gt;&gt;1 hour</td>
<td>6.50</td>
<td>7.34</td>
</tr>
<tr>
<td>mk_adder_test 4 2</td>
<td>&gt;&gt;1 hour</td>
<td>22.95</td>
<td>46.86</td>
</tr>
<tr>
<td>mk_adder_test 5 2</td>
<td>&gt;&gt;1 hour</td>
<td>44.83</td>
<td>170.98</td>
</tr>
<tr>
<td>mk_adder_test 5 3</td>
<td>&gt;&gt;1 hour</td>
<td>38.27</td>
<td>250.16</td>
</tr>
<tr>
<td>mk_adder_test 6 3</td>
<td>&gt;&gt;1 hour</td>
<td>out of memory</td>
<td>1186.4</td>
</tr>
<tr>
<td>mk_adder_test 7 3</td>
<td>&gt;&gt;1 hour</td>
<td>out of memory</td>
<td>3759.9</td>
</tr>
</tbody>
</table>
Summary

• The satisfiability problem asks whether a set of constraints (Boolean or otherwise) have a satisfying solution
• SAT/SMT solvers are powerful tools for solving very large satisfiability problems
• Core SAT algorithms are based on DPLL
• Questions about invariance and reachability can be restated as satisfiability problems
• Learn and use z3
  • https://ericpony.github.io/z3py-tutorial/guide-examples.htm