CS:4980
Foundations of Embedded Systems

Safety Requirements
Part II

A Brief Detour into Computational Complexity

Goal: Classify computational problems in terms of (roughly) how many basic operations it takes to solve the problem, as function of input size

Example 1: Finding maximum of a list of numbers

- Time complexity is linear: $O(n)$

Example 2: Sorting a list of numbers

- Algorithm (e.g. selection-sort) with doubly-nested loop: $O(n^2)$
- More efficient algorithm (e.g. quicksort) possible: $O(n \log n)$

A Brief Detour into Computational Complexity

Goal: Classify computational problems in terms of (roughly) how many basic operations it takes to solve the problem, as function of input size

Example 3: Expression evaluation. Given

1. an expression e (with not/or/ and as operations) over Boolean vars, and
2. an assignment a of 0/1 values to vars,

determine whether e evaluates to 1 or 0.

Linear-time $O(n)$

Example 4: Boolean satisfiability. Given an expression e, determine if there is an assignment a to vars that makes the expression evaluate to 1

- Naïve algorithm: Evaluate e on every possible assignment a
- Exponentially many choices for a: algorithm is $O(2^k)$, k = no. of vars

The Class P

- *Polynomial-time* algorithm means an algorithm with time complexity such as $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, or $O(n^c)$, for constant c
- A problem is in P if there is a polynomial-time algorithm to solve it
- **Examples:**
  - Finding maximum
  - Sorting
  - Expression evaluation
  - Finding shortest path in a graph
- P is the class of *tractable* (i.e., efficiently solvable) problems
  - Problem can be solved exactly
  - Solution will scale reasonably well as input size grows
  - In principle, $O(n)$ is better than $O(n^2)$

NP-Complete Problems

- SAT: Given an expression e over Boolean variables, check if there exists an assignment of 0/1 values to vars for which e evaluates to 1
  - No proof that SAT is in P (no known polynomial-time algorithm)
  - No proof that SAT is not in P
- Cook (1972): SAT is *NP-complete*
- Hundreds of problems equivalent to SAT
  - Hamiltonian Path: Is there a path in a graph from source to destination that visits each vertex exactly once
  - Max Clique: Given a graph, find largest subset of vertices such that there is an edge between every pair of vertices in this set
- Grand Challenge Open Problem : Is P = NP?
  - If you find a polynomial-time algorithm for SAT, then P = NP, and many other problems will have polynomial-time algorithms
  - If you prove SAT is not in P, then P != NP, and many other problems then provably don't have efficient algorithms

NP-Completeness Continued

- Known algorithms for SAT are exponential-time in the worst-case, but
  - Highly efficient implementations, SAT solvers, exist
  - Can handle millions of variables
  - Many practical problems solved by encoding into SAT
- Key feature of NP problems such as SAT: suffices to find one satisfying assignment
- This does not hold for all intractable problems
  - Validity: Given a Boolean expression e, is it the case that e evaluates to 1 no matter what values we give to its variables
- Many complexity classes beyond NP: coNP, PSPACE, Exptime, …
  - Problems may require exponential-time (or more) to solve
  - Not all exponential-time problems are equal.

(Un)Decidability

- Some problems cannot be solved by a computer at all!
- Fundamental Theorem of CS (Alan Turing, 1936):
  - The *Halting problem* for Turing machines is undecidable

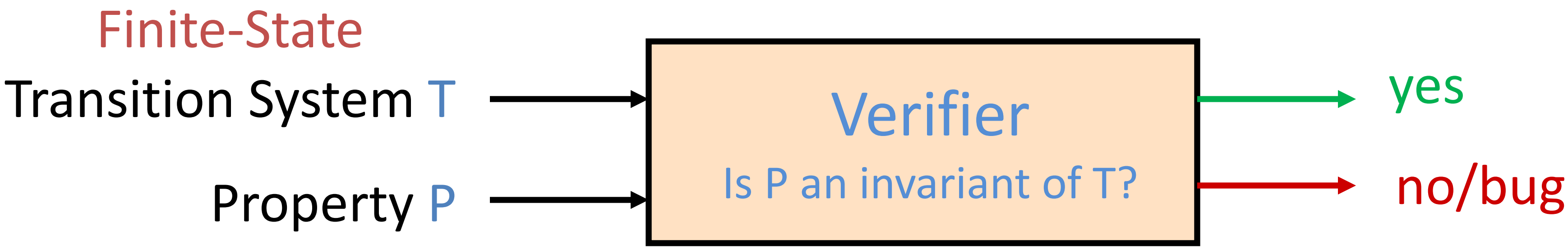
    There is no program that takes as its input an arbitrary program C and an arbitrary input x, and determines if C terminates on x
- **Intuition:** If a program could analyze other programs exactly, then it can analyze itself, and this suffices to set up a logical contradiction!
- A surprisingly undecidable problem: Does a given a polynomial (e.g., $x^3 + 2xy^2 - 15xy + 156$) have integer roots?
- **Decidable Problems:** There exists a program (or Turing machine) that solves the problem correctly (gives the right answer and stops)
  - Includes problems in P as well as intractable classes such as NP, Exptime, etc.

Back To Invariant Verification Problem

Theorem: The invariant verification problem is undecidable.

Proof idea: undecidable problems for Turing machines can be recast as invariant verification problems for transition systems with integer state variables

Finite-State Invariant Verification Problem

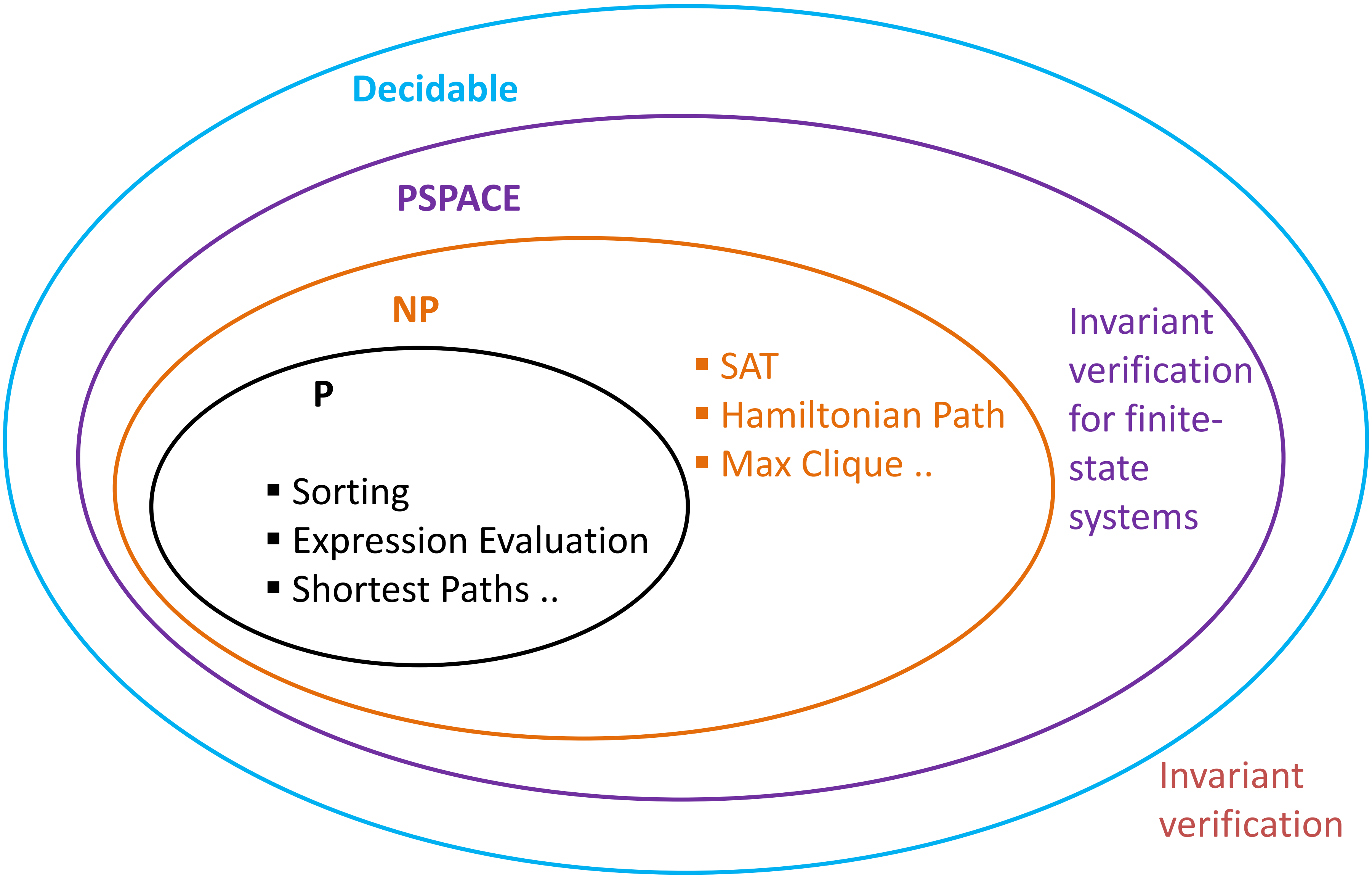


Theorem: The invariant verification problem for finite-state systems is decidable

Proof sketch: If T has k Boolean state vars, then total number of states is 2^k.

Verifier can systematically search through all possible states.

Complexity is exponential. More precisely, it is PSPACE, a class of problems harder than NP-complete problems such as SAT.


Decidable
PSPACE
NP
P
Sorting
Expression Evaluation
Shortest Paths ..
SAT
Hamiltonian Path
Max Clique ..
Invariant verification for finite-state systems
Invariant verification

Credits

Notes based on Chapter 3 of

Principles of Cyber-Physical Systems
by Rajeev Alur
MIT Press, 2015