EC3204: Programming Languages and Compilers

Basic Concepts in Computer Science
Undecidability, Halting Problem -

Sunbeom So
Fall 2024

Decision Problem

- A question that can be answered with "yes" or "no".
  - Examples: string recognition with FA, the ambiguity of CFG, boolean satisfiability, first-order logic satisfiability, program equivalence
- Many other computational problems can be reduced into decision problems. Consider the traveling salesman problem (TSP).
  - Optimization problem: what is the shortest route that visits each city exactly once and returns to the starting city?
  - Decision problem: given a distance d, is there a round-trip route with a cost less than d?
- Decision problems are classified into **decidable** and **undecidable** problems.
  - **decidable**: there is an algorithm that can always produce a correct answer (yes or no) for any given input within a finite amount of time.
  - **undecidable**: a decision problem proven to be impossible to construct an algorithm that can always produce a correct yes-no answer.

Exapmles

Examples of decidable problems:

- Regular expression matching: does a string s match a regex r?
- String recognition with FA: can a string s be accepted by FA?
- Decision problem version of TSP: we can answer either "yes" or "no" by performing an exhaustive search!
- Set membership checking: is y is an element of a finite set X?

Examples of undecidable problems:

- Halting problem: given a computer program p and its input i, does $p(i)$ terminate or loop forever?
- CFG ambiguity: is the grammar G ambiguous or not?
- CFG equivalence: given G_1 and G_2, $L(G_1) = L(G_2)$?
- Proving program correctness/safety: does a program p always satisfy a property ϕ?

Fortunately(?), many interesting research problems are undecidable ...

Informal Proof: Undecidability of Halting Problem

- Suppose there is an algorithm `halt` that solves the Halting problem. That is, `halt(p,i)` returns `true` iff `p(i)` terminates.

Informal Proof: Undecidability of Halting Problem

- Suppose there is an algorithm `halt` that solves the Halting problem. That is, `halt(p,i)` returns `true` iff `p(i)` terminates.
- Define the function `inverse` that inverses the effects of `halt`.

```
function inverse (p) {
  if halt(p,p) = false
    return true
  else
    loop forever
}
```

Q. Is `halt(inverse,inverse) = true`?

Informal Proof: Undecidability of Halting Problem

- Suppose there is an algorithm `halt` that solves the Halting problem. That is, `halt(p,i)` returns `true` iff `p(i)` terminates.
- Define the function `inverse` that inverses the effects of `halt`.

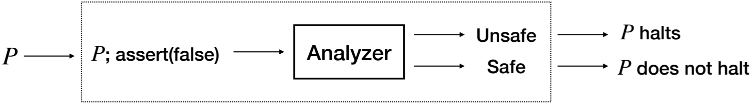
```
1 function inverse (p) {
2   if halt(p,p) = false
3     return true
4   else
5     loop forever
6 }
```

Q. Is `halt(inverse,inverse) = true`?

- Suppose `halt(inverse,inverse) = true`.
  - By definition of `halt`, `inverse(inverse)` must terminate.
  - By lines 4–5, `inverse(inverse)` loops forever. Contradiction!
- Suppose `halt(inverse,inverse) = false`.
  - By definition of `halt`, `inverse(inverse)` must not terminate.
  - By lines 1–2, `inverse(inverse)` terminates. Contradiction!

Informal Proof: Undecidability of Program Analysis

- (Claim) We cannot have an ideal analyzer that can always produce a correct answer (`safe` or `unsafe`) for any program and specification.
- (Proof by Contradiction) Suppose exact analysis is possible. Then, we can solve the Halting problem using it!



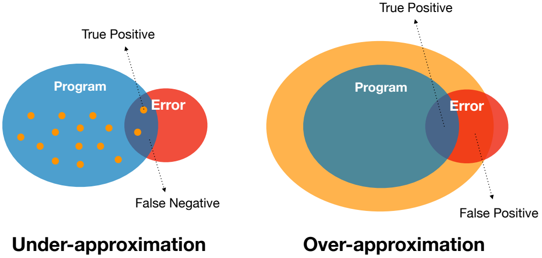
Theorem (Rice Theorem)

Let $\mathbb{L}$ be a Turing-complete language. Let ϕ be a nontrivial semantic property, i.e., there are $\mathbb{L}$ programs that satisfy ϕ and $\mathbb{L}$ programs that do not satisfy ϕ. There exists no algorithm A such that

$$\text{for every program } p \in \mathbb{L},\ A(p, \phi) = \texttt{true} \iff p \text{ satisfies } \phi.$$

Summary

- Decision problem: yes-no question
- Not all decision problems can be solved by computers (Turing machines). There are no perfect solutions for undecidable problems.
- We challenge the impossibility by approximations!



- Research topics in each approach (not limited to the below)
  - fuzzing/symbolic execution: how to minimize false negatives?
  - abstract interpretation/verification: how to minimize false positives?