CS 357: Advanced Topics in Formal Methods

Fall 2019

Lecture 3

Aleksandar Zeljić
(materials by Clark Barrett)
Stanford University

Outline

- Computability and Decidability
- DP
- DPLL
- Abstract DPLL

Computability

The important notion of *computability* relies on a formal model of computation.

Many formal models have been proposed:

Computability

The important notion of *computability* relies on a formal model of computation.

Many formal models have been proposed:

1. General recursive functions defined by means of an equation calculus (Gödel-Herbrand-Kleene)
2. λ-definable functions (Church)
3. μ-recursive functions and partial recursive functions (Gödel-Kleene)
4. Functions computable by finite machines known as Turing machines (Turing)
5. Functions defined from canonical deduction systems (Post)
6. Functions given by certain algorithms over a finite alphabet (Markov)
7. Universal Register Machine-computable functions (Shepherdson-Sturgis)

Fundamental Result

All of these (and many other) models of computation are equivalent. That is, they give rise to the same class of functions.

Computability and Decidability

All of these models are equivalent to what can be achieved by a computer with any standard programming language, given arbitrary (but finite) time and memory.

Church's Thesis

A notion known as Church's thesis states that all models of computation are either equivalent to or less powerful than those just described.

We will accept Church's thesis and thus define a function to be *computable* if we can describe precisely (using any model of computation) how to compute it. Such a description will be called an *effective procedure*.

Computability and Decidability

All of these models are equivalent to what can be achieved by a computer with any standard programming language, given arbitrary (but finite) time and memory.

Church's Thesis

A notion known as Church's thesis states that all models of computation are either equivalent to or less powerful than those just described.

We will accept Church's thesis and thus define a function to be *computable* if we can describe precisely (using any model of computation) how to compute it. Such a description will be called an *effective procedure*.

Decidability

Given a universal set U, a set $S \subseteq U$ is decidable if there exists a computable function $f : U \to \{\mathbf{F}, \mathbf{T}\}$ such that $f(x) = \mathbf{T}$ iff $x \in S$.

Computability and Decidability

All of these models are equivalent to what can be achieved by a computer with any standard programming language, given arbitrary (but finite) time and memory.

Church's Thesis

A notion known as Church's thesis states that all models of computation are either equivalent to or less powerful than those just described.

We will accept Church's thesis and thus define a function to be *computable* if we can describe precisely (using any model of computation) how to compute it. Such a description will be called an *effective procedure*.

Decidability

Given a universal set U, a set $S \subseteq U$ is decidable if there exists a computable function $f : U \to \{\mathbf{F}, \mathbf{T}\}$ such that $f(x) = \mathbf{T}$ iff $x \in S$.

Decidability of W

Earlier, we presented an algorithm which, given any expression α determines whether the expression is well-formed. Thus, the set W of well-formed formulas is decidable.

Decidability

Some decidable sets

- For a given finite set of *wffs* Σ, the set of all *tautological consequences* of Σ (i.e. $\{\alpha \mid \Sigma \models \alpha\}$) is decidable.

Decidability

Some decidable sets

- For a given finite set of *wffs* Σ, the set of all *tautological consequences* of Σ (i.e. $\{\alpha \mid \Sigma \models \alpha\}$) is decidable.

  The truth table algorithm given earlier decides $\Sigma \models \alpha$.

Decidability

Some decidable sets

- For a given finite set of *wffs* Σ, the set of all *tautological consequences* of Σ (i.e. $\{\alpha \mid \Sigma \models \alpha\}$) is decidable.

  The truth table algorithm given earlier decides $\Sigma \models \alpha$.

- The set of tautologies is decidable.

Decidability

Some decidable sets

- For a given finite set of *wffs* Σ, the set of all *tautological consequences* of Σ (i.e. $\{\alpha \mid \Sigma \models \alpha\}$) is decidable.

  The truth table algorithm given earlier decides $\Sigma \models \alpha$.

- The set of tautologies is decidable.

  The set of tautologies is just the set of tautological consequences of the empty set.

Decidability

Some decidable sets

- For a given finite set of *wffs* Σ, the set of all *tautological consequences* of Σ (i.e. $\{\alpha \mid \Sigma \models \alpha\}$) is decidable.

  The truth table algorithm given earlier decides $\Sigma \models \alpha$.

- The set of tautologies is decidable.

  The set of tautologies is just the set of tautological consequences of the empty set.

Existence of undecidable sets

A simple argument shows the existence of undecidable sets of expressions: an algorithm is completely determined by its finite description. Thus, there are only countably many effective procedures. But there are uncountably many sets of expressions.

Why?

Decidability

Some decidable sets

- For a given finite set of *wffs* Σ, the set of all *tautological consequences* of Σ (i.e. $\{\alpha \mid \Sigma \models \alpha\}$) is decidable.

  The truth table algorithm given earlier decides $\Sigma \models \alpha$.

- The set of tautologies is decidable.

  The set of tautologies is just the set of tautological consequences of the empty set.

Existence of undecidable sets

A simple argument shows the existence of undecidable sets of expressions: an algorithm is completely determined by its finite description. Thus, there are only countably many effective procedures. But there are uncountably many sets of expressions.

Why?

The set of expressions is countably infinite. Therefore, its power set is uncountable.

Semi-Decidability

Suppose we wish to determine whether $\Sigma \models \alpha$ where Σ is infinite. In general, this is not decidable.

But we can obtain a weaker result:

A set A is *semi-decidable* (or *effectively enumerable*) if there is an effective procedure which lists, in some order, every member of A.

Note that if A is infinite, then the procedure will never finish, but every member of A must appear in the list after some finite amount of time.

Semi-Decidability

Suppose we wish to determine whether $\Sigma \models \alpha$ where Σ is infinite. In general, this is not decidable.

But we can obtain a weaker result:

A set A is *semi-decidable* (or *effectively enumerable*) if there is an effective procedure which lists, in some order, every member of A.

Note that if A is infinite, then the procedure will never finish, but every member of A must appear in the list after some finite amount of time.

Theorem

A set A of expressions is effectively enumerable iff there is an effective procedure which, given any expression α, produces the answer "yes" iff $\alpha \in A$.

Semi-Decidability

Suppose we wish to determine whether $\Sigma \models \alpha$ where Σ is infinite. In general, this is not decidable.

But we can obtain a weaker result:

A set A is *semi-decidable* (or *effectively enumerable*) if there is an effective procedure which lists, in some order, every member of A.

Note that if A is infinite, then the procedure will never finish, but every member of A must appear in the list after some finite amount of time.

Theorem

A set A of expressions is effectively enumerable iff there is an effective procedure which, given any expression α, produces the answer "yes" iff $\alpha \in A$.

Proof

If A is effectively enumerable, then we simply enumerate its members and check each one to see if it is equivalent to α. If it is, we return "yes" and stop. Otherwise, we keep going. Thus, if $\alpha \in A$, the procedure produces "yes". If $\alpha \notin A$, the procedure runs forever.

Proof, continued

On the other hand, suppose that we have an effective procedure P which produces "yes" iff $\alpha \in A$. To produce an enumeration of A, we proceed as follows. First enumerate all expressions:

$$\epsilon_1, \epsilon_2, \epsilon_3, \ldots$$

Then proceed as follows.

- Break the procedure P into a finite number of "steps".
- Run P on ϵ_1 for 1 step.
- Run P on ϵ_1 for 2 steps, and then run P on ϵ_2 for 2 steps.
- ...
- Run P on each of $\epsilon_1, \ldots, \epsilon_n$ for n steps each
- ...

If at any time, the procedure P produces "yes", then we list the expression which produced "yes" and continue.

This procedure will eventually enumerate all members of A. □

Semi-Decidability

Theorem

A set is decidable iff both it and its complement (with respect to a given universal set) are effectively enumerable.

Semi-Decidability

Theorem

A set is decidable iff both it and its complement (with respect to a given universal set) are effectively enumerable.

Proof

Alternate between running the procedure for the set and the procedure for its complement. One of them will eventually produce "yes".

Semi-Decidability

Theorem

A set is decidable iff both it and its complement (with respect to a given universal set) are effectively enumerable.

Proof

Alternate between running the procedure for the set and the procedure for its complement. One of them will eventually produce "yes".

Properties of decidable and semi-decidable sets

Decidable sets are closed under union, intersection, and complement.

Semi-decidable sets are closed under union and intersection.

Semi-Decidability

Theorem

If Σ is an effectively enumerable set of *wffs*, then the set of tautological consequences of Σ is effectively enumerable.

Semi-Decidability

Theorem

If Σ is an effectively enumerable set of *wffs*, then the set of tautological consequences of Σ is effectively enumerable.

Proof

Consider an enumeration of the elements of Σ:

$$\sigma_1, \sigma_2, \sigma_3, \ldots$$

By the compactness theorem, $\Sigma \models \alpha$ iff $\{\sigma_1, \ldots, \sigma_n\} \models \alpha$ for some n.

Hence, it is sufficient to successively test:

$$\begin{array}{lcl} \emptyset & \models & \alpha \\ \{\sigma_1\} & \models & \alpha \\ \{\sigma_1, \sigma_2\} & \models & \alpha \\ \ldots & & \end{array}$$

If any of these conditions is met (each of which is decidable), the answer is "yes".

Semi-Decidability

Theorem

If Σ is an effectively enumerable set of *wffs*, then the set of tautological consequences of Σ is effectively enumerable.

Proof (continued)

This demonstrates that there is an effective procedure that, given any *wff* α, will output "yes" iff α is a tautological consequence of Σ.

Thus, the set of tautological consequences of Σ is effectively enumerable. $\square$

Davis-Putnam Algorithm

From now on, unless otherwise indicated, we assume formulas are in CNF, or, equivalently, that we have a set of clauses to check for satisfiability (i.e. the conjunction is implicit).

The first algorithm to try something more sophisticated than the truth-table method was the *Davis-Putnam (DP)* algorithm, published in 1960.

It is often confused with the later, more popular algorithm presented by Davis, Logemann, and Loveland in 1962, which we will refer to as *Davis-Putnam-Logemann-Loveland (DPLL)*.

We first consider the original DP algorithm.

Davis-Putnam Algorithm

There are three satisfiability-preserving transformations in DP.

- The 1-literal rule
- The affirmative-negative rule
- The rule for eliminating atomic formulas

The first two steps reduce the total number of literals in the formula.

The last step reduces the number of variables in the formula.

By repeatedly applying these rules, eventually we obtain a formula containing an empty clause, indicating unsatisfiability, or a formula with no clauses, indicating satisfiability.

Davis-Putnam Algorithm

The 1-literal rule

Also called *unit propagation*.

Suppose (p) is a unit clause (clause containing only one literal). Let $-p$ denote the negation of p where double negation is collapsed (i.e. $-\neg q \equiv q$).

- Remove all instances of $-p$ from clauses in the formula (shortening the corresponding clauses).
- Remove all clauses containing p (including the unit clause itself).

Davis-Putnam Algorithm

The affirmative-negative rule

Also called the *pure literal rule*.

If a literal appears *only positively* or *only negatively*, delete all clauses containing that literal.

Why does this preserve satisfiability?

Davis-Putnam Algorithm

Rule for eliminating atomic formulas

Also called the *resolution rule*.

- Choose a propositional symbol p which occurs positively in at least one clause and negatively in at least one other clause.
- Let P be the set of all clauses in which p occurs positively.
- Let N be the set of all clauses in which p occurs negatively.
- Replace the clauses in P and N with those obtained by resolution on p using all pairs of clauses from P and N.

For a single pair of clauses, $(p \vee l_1 \vee \cdots \vee l_m)$ and $(\neg p \vee k_1 \vee \cdots \vee k_n)$, *resolution on* p forms the new clause $(l_1 \vee \cdots \vee l_m \vee k_1 \vee \cdots \vee k_n)$.

DPLL Algorithm

In the worst case, the resolution rule can cause a quadratic expansion every time it is applied.

For large formulas, this can quickly exhaust the available memory.

The DPLL algorithm replaces resolution with a *splitting rule*.

- Choose a propositional symbol p occuring in the formula.
- Let Δ be the current set of clauses.
- Test the satisfiability of $\Delta \cup \{(p)\}$.
- If satisfiable, return *True*.
- Otherwise, return the result of testing $\Delta \cup \{(\neg p)\}$ for satisfiability.