COMP 409, Lecture 4

Tuesday, January 28, 2003

1 Last Time

Last time, we looked at *Three Central Definitions of Semantics*:

$\tau \models \varphi \leftrightarrow \varphi(\tau) = 1 \leftrightarrow \tau \in models(\varphi)$

where $\varphi \in formula$, $\tau \in 2^{Prop'}$ provided that $Prop' \supseteq AP(\varphi)$

That is:

Philosopher's Perspective $\leftrightarrow$ EE's Perspective $\leftrightarrow$ Software Engrg Perspective

Where:

1. Philosopher's view of semantics is a binary relation ($\models$) between structures and formulas.
2. Electrical Engineer's view of semantics is a mapping of voltages on a wire (circuits, with gates as connectives).
3. Software Engineer's view of semantics is that we can describe a set of worlds (or truth assignments) in which a given formula is true.

2 The Algorithmic Perspective

Now let us take a look at the *Algorithmic Perspective*:

if we have: $((p \wedge q) \vee r)$ with set $\tau : p \mapsto 0, q \mapsto 1, r \mapsto 0$

does: $\tau \models ((p \wedge q) \vee r)$ hold?

No. We were able to easily get to this answer by calculating it with a "mental algorithm". However, we need a more formal way of describing the evaluation process.

2.1 Devising An Algorithm for Truth Evaluation

Given a formula $\varphi \in Form$, a truth assignment $\tau \in 2^{Prop'}$, and that AP(φ) $\subseteq$ Prop', we want to design an algorithm $Eval$ such that $Eval(\varphi, \tau) = 1$ iff $\varphi(\tau) = 1$, that is, we want an algorithm that checks whether a given truth assignment satisfies a given formula.

Now for any algorithm we design, the following needs to hold:

1. Correctness - Does this algorithm solve the right problem? (Correctness here means that $Eval(\varphi, \tau) = \varphi(\tau)$.) Correctness is typically a relatively straightforward idea.

2. Performance - How fast does it solve the problem? This is a much more complicated issue than correctness; in fact, it is a major issue in Computer Science. This leads us to the question: *how is performance measured*? The initial answer is *it depends*.

2.2 Measuring Performance

In order to measure performance and perform optimization on our algorithm, we must look at:

1. Scalability - For example, if we run two algorithms on a benchmark and we want to scale these algorithms up for a larger program, a good algorithm would *scale up gracefully*. That is, that the scale-up of the algorithm requires a minimum amount of resources (the resources we are considering are time and space/memory). We are also assuming, of course, that the algorithm still performs as desired.

2. Worst-Case Behavior - For example, how does the algorithm behave with the *hardest possible problems*? If the algorithm behaves in a "reasonable" way, then it handles worst-case behavior well.

Typically, we express the time and space requirement of a problem as functions of the input size, and how it grows with an increase in the input size. We generally speak of three types of problems that have different degrees of growth:

- Slow growth - logarithmic
- Medium growth - polynomial
- Fast growth - exponential

The following table summarizes the different major complexity classes with respect to time and space requirements.

The class PTIME in table 1 is the class of problems (or algorithms) that can be solved in polynomial running time. Similarly, PSPACE is the class of problems that require space polynomial in the size of the input problem to be solved.

Table 1: **Complexity Classes (Growth on horizontal axis, Resources on vertical axis)**

	Log	Poly	Exp
Time	-	PTIME	EXPTIME
Space	LOGSPACE	PSPACE	EXPSPACE

To solve any problem we have to at least read the problem first, that is, at least as much time as the size of the problem is required to read the problem, that is why we do not have a LOGTIME class in table 1 analogous to LOGSPACE.

2.3 Complexity Theory

Complexity theory tries to understand the relationship between different complexity classes. We know for a fact that with PTIME and PSPACE, we must have:

$$PTIME \subseteq PSPACE$$

(if we have PTIME, then we must have PSPACE) and analogously,

$$EXPTIME \subseteq EXPSPACE$$

But what about LOGSPACE and PTIME?

Imagine that we are writing only 0's and 1's. The amount of memory used is *10 log n*. So, the number of possible configurations is $10logn = n^{10}$. So, in general we have $2^{c\ log\ n} = n^c$. Therefore, if the amount of necessary memory is logarithmic in the input size, then the time needed is polynomial in the input size. Therefore, we have

$$LOGSPACE \subseteq PTIME$$

and similarly,

$$PSPACE \subseteq EXPTIME$$

Through this reasoning, we can derive the *Periodic Table*:

$$LOGSPACE \subseteq PTIME \subseteq PSPACE \subseteq EXPTIME \subseteq EXPSPACE.$$

3 Strict Containment

What else do we need to know about complexity classes besides containment? We need to know their precise relationships – are they the same? (if they are, the whole thing "collapses" and things are much easier to prove!)

Notice 1 *Exponential gaps matter!*

Theorem 1

1. $LOGSPACE \subset PSPACE$
2. $PTIME \subset EXPTIME$
3. $PSPACE \subset EXPSPACE$
   True due to exponential gaps.

From the above three ordering of sets, it follows that at least one in each of the following three cases must be a strict subset relation (and not subset or equal relation).

1. $LOGSPACE \subseteq PTIME \subseteq PSPACE$
2. $PTIME \subseteq PSPACE \subseteq EXPTIME$
3. $PSPACE \subseteq EXPTIME \subseteq EXPSPACE$

But even after 30 years of research it is not known which of these relations is a strict subset relations and this remains one of the fundamental questions of computer science.

4 Back to $Eval(\varphi, \tau)$

Recall that we wish to write an algorithm for evaluation: $Eval(\varphi, \tau)$.

Assuming we know how to parse the formula, an algorithm for $Eval(\varphi, \tau)$ would be:

case
  φ *is* p : *return* $\tau(p)$
  φ *is* $(\neg\varphi)$: *return* $\neg(Eval(\varphi, \tau))$
  φ *is* $(\theta \circ \psi)$: *return* $\circ\,(Eval(\theta, \tau),\ Eval(\psi, \tau))$
esac

4.1 Checking for Correctness

To check for correctness, we have the semantical definition of $\varphi(\tau)$ as:

Definition 1 $\varphi(\tau)$:

1. φ *is* p: $p(\tau) = \tau(p)$
2. φ *is* $(\neg\varphi)$: $(\neg\varphi)(\tau) = \neg(\theta(\tau))$
3. φ *is* $(\theta \circ \psi)$: $(\theta \circ \psi)(\tau) = \circ(\theta(\tau), \psi(\tau))$

We can see that the semantical definition is an algorithm itself!

Lemma 1 $Eval(\varphi, \tau) = \varphi(\tau)$

Proof: By induction on φ

We have discovered that logical definitions can really be thought of as algorithms.

4.2 Complexity of the Algorithm

We also need to consider the complexity, or running time, of the algorithm.

First, we will focus on analyzing time requirements of Eval, to ask how much time $T(n)$ do we spend on parsing a formula length n

$$T(n) = O(n) + T(k) + T(n - k - 1), 1 \leq k \leq n$$

(The formula for $T(n)$ comes from analyzing the longest-processing-time for a formula, i.e., when it has a binary connective $\circ$)

so $T(n) = O(n^2)$

The same applies to space requirements of *Eval* (n recursive steps, each requiring at most n memory locations), so we have:

Lemma 2 *Eval uses polynomial (quadratic) time and space.*

and so we have, $Eval \in PTIME$

Theorem 2 *(Nancy Lynch, MIT)*
Eval *can be implemented in LOGSPACE.*

Comment: Time is a more flexible resource than space (but this is really a matter of preference). We call this the *time-space tradeoff.*