Lecture 26

November 4, 2003

1 Review

Last time we looked at the distinction between EC and EC_Δ, and showed that $\overline{2C}$ is a class of formulas that does not fit inside EC (and, in fact, not even in EC_Δ). This led to the corollary

Corollary 1 *$EC \neq EC_\Delta$. Also, $\overline{EC_\Delta} \notin EC_\Delta$ (here $\overline{A}$ denotes the complement of A).*

2 Relating elementary and extended elementary

We can use analogies from previously studied topics to obtain a better grasp on the relationship between elementary and extended elementary. Let us compare EC/EC_Δ with P/NP and R/RE. We know little of P/NP...

But on the other hand, consider R/RE. Recall that RE, the set of recursively enumerable languages, contains all languages accepted by some Turing machine in finite time. *Recursive* is the set of languages accepted or rejected by a Turing machine in finite time. Clearly, therefore, $RE \cap co - RE = recursive$. It is also possible to generate something that is provably in R and not in RE.

So the questions we must consider:

1. Is there something in EC_Δ and not in EC?
   Yes, we have already considered 2C.

P/NP
$P \overset{?}{\subset} NP$
$NP \overset{?}{\neq} co - NP$
$NP \overset{?}{\cap} co - NP = P$

Table 1: NP vs P

recursive/RE
$R \subset RE$
$RE \neq co - RE$
$RE \cap co - RE = R$

Table 2: RE vs R

2. Is everything in EC in EC_Δ?
   Yes, a single formula is a singleton set of formulas.

3. Is $EC_\Delta \cap co - EC_\Delta = EC$?

Theorem 1 $EC_\Delta \cap co\text{-}EC_\Delta = EC$

In order to prove this result, we need to use the Compactness Theorem, which we repeat below:

Theorem 2 (Compactness Theorem) *A set of formulas Φ is satisfiable iff it is finitely satisfiable.*

Proof: We prove this by showing that any class of structures C in both EC_Δ and $co\text{-}EC_\Delta$ must be in EC.

Suppose that C is a class of structures such that $C \in EC_\Delta$ and $C \in co\text{-}EC_\Delta$. This means that $\overline{C} \in EC_\Delta$.

Thus, for sets Φ and Ψ of sentences, $C = models\,(\Phi)$ and $\overline{C} = models\,(\Psi)$. Since C and $\overline{C}$ are mutually exclusive, $\Phi \cup \Psi$ is unsatisfiable.

By the compactness theorem, there exist finite $\Phi' \subseteq \Phi$ and $\Psi' \subseteq \Psi$ such that $\Phi' \cup \Psi'$ is unsatisfiable.

Let $C' = models\,(\Phi')$. Since $\Phi' \subseteq \Phi$, $C \subseteq C'$ (remember that the smaller set of formulas is less restrictive, therefore it has more models). Suppose $C \neq C'$, and let $A \in C' - C = C' \cap \overline{C}$ (set-theoretic equality). This implies that $A \in models\,(\Phi') \cap models\,(\Psi) = models\,(\Phi' \cup \Psi)$, which means that A satisfies $\Phi' \cup \Psi$.

Since this contradicts the compactness theorem, $C' = C = models\,(\Phi')$. Since Φ' is finite, we can construct a single sentence φ which is the disjunction of the elements of Φ' and is therefore equivalent to Φ'. As a result, $C = models\,(\varphi)$, and therefore, $C \in EC$.

3 Tower of Abstractions

One of the most fundamental characteristics of any scientific field is the ability to abstract away unnecessary details. A systems programmer rarely needs to worry about the digital circuits on which the operating system runs. A C++ programmer need not know the operating system on which his program will run. This is possible thanks to a concept called "Tower of Abstraction".

a. Analog circuits

b. Digital circuits

c. Microarchitecture

d. Architecture level

e. OS kernel

f. OS

g. Assembly level programming

h. High-level programming languages (Fortran, Cobol)

i. Middleware

Computer science starts at b.