COMP 409 Logic in Computer Science

April 3, 2003

1 Isomorphism

Let A, B be isomorphic structures, and φ a sentence, then $A \models \varphi$ iff $B \models \varphi$.

$\simeq$ is:

- Reflexive
- Symmetric
- Transitive

Therefore, $\simeq$ is an equivalence relation.

2 Elementarily equivalence

Definition 1 *Let A, B be structures on some vocabulary, then A and B are* elementarily equivalent*, denoted $A \equiv B$ if for all sentences φ we have $A \models \varphi iff B \models \varphi$.*

$A \simeq B \Rightarrow A \equiv B$. In other words, $\simeq \subseteq \equiv$.

Theorem 1 *Let Φ be a set of sentences. Let models(Φ) = $\ A : A \models \Phi\ $ = C. Then C is a union of equivalence classes of $\equiv$.*

3 Elementary and extended elementary

Definition 2 *A class C of structures is* elementary*, denoted EC, if $C = models(\varphi)$ fo some sentence φ. (Examples include triangle-freedom, transitivity). C is* extended elementary*, denoted EC_Δ, if $C = models(\Phi)$ for some set Φ of sentences.*

Definition 3 *$G = \{V, E\}$ is k-colorable iff there is a mapping $c : V \mapsto \{1, \ldots, k\}$ such that $\langle u, v\rangle \in E \rightarrow c(u) \neq c(v)$.*

Question 1 *Is 2C EC? Or EC_Δ?*

Lemma 1 *A graph is 2C iff it has no cycle of odd length.*

Theorem 2 *2C is EC_Δ.*

Proof: Known result from graph theory. We can define a set of first-order sentences which model 2-colorable graphs.

Define $p_n(u,v)$ recursively as follows:

$$p_1(x,y) = \langle x,y\rangle \in E \tag{1}$$
$$p_n(x,y) = \exists z\,(\langle x,z\rangle \in E \wedge p_{n-1}(z,y)) \tag{2}$$

That is, $p_n(x,y)$ iff there is a path of length n from vertex x to vertex y. Likewise, $p_n(x,x)$ iff there is a cycle of length n which includes the vertex x.

Define $cycle_n$ as follows:

$$cycle_n = \exists x(p_n(x,x))$$

To restate the lemma, a graph is two colorable iff no vertex in the graph is in an odd cycle. This can be translated directly into a set of first-order sentences:

$$\Phi = \{\neg cycle_1, \neg cycle_3, \neg cycle_5, \neg cycle_7, \ldots\}$$

By construction, $models(\Phi) = 2C$, so $2C \in EC_\Delta$.

Definition 4 *Let Σ be a vocabulary and C a class of structures on Σ, then C' is the class of structures not in C.*

Lemma 2 *Properties of EC:*

- *$EC \Rightarrow EC_\Delta$*
- *If C is EC, then C' (complement of C) is also EC.*

Now that we have two notions of classes of structures, elementary and extended elementary, we wish to determine what distinguishes them. We first need a compactness theorem, equivalent to the compactness theorem for propositional logic.

Definition 5 *A set Φ of formulas is* finitely satisfiable *iff every finite subset Φ' of Φ is satisfiable.*

Theorem 3 Compactness Theorem: *A set of formulas Φ is satisfiable iff it is finitely satisfiable.*

Using this theorem, we will start by showing that graph 2-colorability is not in EC, even though, as we have shown above, it is in EC_Δ.

Theorem 4 $2C \notin EC$

Proof: We prove that $2C \notin EC$ by contradiction.

Suppose $2C \in EC$. Then $2C = models\,(\varphi)$ for some sentence φ. $\overline{2C}$ defines all graphs not in $2C$; that is, all graphs that are not 2-colorable. $\overline{2C} = models\,((\neg\varphi))$, so $\overline{2C} \in EC$. This implies that $\overline{2C} = models\,(\Psi)$ for some set of sentences Ψ (consider $\Psi = \{(\neg\varphi)\}$), and therefore that $\overline{2C} \in EC_\Delta$.

We already know that $2C \in EC_\Delta$, so for some set Φ, $2C = models\,(\Phi)$. Since $2C$ and $\overline{2C}$ are mutually exclusive, $\Phi \cup \Psi = models\,(\emptyset) = \emptyset$. That is, $\Phi \cup \Psi$ is unsatisfiable.

By the compactness theorem, there are finite subsets $\Phi' \subseteq \Phi$ and $\Psi' \subseteq \Psi$ such that $\Phi' \cup \Psi'$ is unsatisfiable. Let $|\Phi'| = j$. Based on the definition of Φ above, we know

$$\Phi' = \{\neg cycle_{i1}, \neg cycle_{i2}, \neg cycle_{i3}, \ldots, \neg cycle_{ij}\}$$

Let A be a graph containing only an odd cycle of length $k > max\,\{i_1, \ldots i_j\}$.

$$\begin{array}{lcll} A & \models & \Psi & \text{Since } A \text{ is not } 2C \\ A & \models & \Phi' & \text{Since } A \text{ is bigger than the largest cycle disallowed in } \Phi' \end{array}$$

So $\Phi' \cup \Psi$ is satisfiable. This contradicts the compactness theorem. Therefore $2C$ is not EC (and $\overline{2C}$ is not EC_Δ).

4 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. Our analogies are P/NP and recursive and recursively enumerable (RE).

(RE is the set of recursively enumerable languages. That is, the set of languages accepted by some Turing machine in finite time. The Turing machine may diverge if the language is rejected. Likewise, co-RE is the set of languages rejected by some turning machine in finite time. *recursive* is the set of languages accepted or rejected by a Turing machine in finite time. Clearly, therefore, $RE \cap co - RE = recursive$.)

Table 1: Analogies

EC/EC_Δ	P/NP	recursive/RE
$EC \subset EC_\Delta$	$P \subset ?NP$	$Rec \subset RE$
$EC_\Delta \neq co - EC_\Delta$	$NP \neq ?co - NP$	$RE \neq co - RE$
$EC_\Delta \wedge co - EC_\Delta = EC$	NP ∩ co-NP = P	$RE \cap co - RE = rec$

Theorem 5 $EC_\Delta \cap co\text{-}EC_\Delta = EC$

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 $\Phi' \subseteq \Phi$, $C \subseteq C'$. Suppose $C \neq C'$, and let $A \in C' - C = C' \cap \overline{C}$. 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, $C \in EC$.

5 Application of First Order Logic to Databases

In our last lectures we have seen how we can model sequential circuits using first order logic. Another application of first order logic which came up in 1970's is modelling Relational Databases. What is a database? One answer could be 'Database is a set of facts and relations'. If that is the case then why don't we put all the information in a single file. The answer to it is if we put all information in a single file there is no **structure**. People felt that it is nice to keep the information in a particular structure so that it is easier for many people to use the same data. From this came up the concept of Central Database in which an application program would get services from the Database and OS and the Database would get services from the OS. Now people started to discuss about how to give structure to data. In the 1960's people came up with the idea of organizing data in some sort of hierarchy. This came to be known as the Hierarchical model. A Hierarchical database stores its data in a series of records each of which has a set of field values attached to it. It collects all instances of record together as a record type. To create links between these record types, the hierarchical model uses Parent-Child relationships which creates tree like structures. After that people found out that it would be useful if cycles were allowed in such structures. This resulted in a more general model called the Network Model for Databases.

In 1970, Ted Codd told that this model was too complicated since easy retrieval was not possible in this model. He wanted easy query instead of going through all links as in the previous models. So he came up with the idea of **"Tables"**. He found out that there is a connection between Tables and First Order Logic. Table is a set of rows and a relation is a set of tuples. So the idea of a row in a table is analogous to a tuple in a relation. From this he devised the "Relational Model" of databases. This consisted of centralized data, with a set $\sigma = R_1...R_k, f_1...f_l$ of relations, and functions. So databases could be viewed as structures $A = (D, R_1^A...R_k^A, f_1^A...f_l^A)$. This was his first big idea.

His next big idea was that a query can be expressed as a formula $\varphi(x_1, \ldots, x_k)$.

We can view a variable assignment α, where $\alpha(x_i) = a_i$ as a tuple and look at it as $\langle a_1, \ldots, a_k \rangle$ and say that $A, \langle a_1, \ldots, a_k \rangle \models \varphi(x_1, \ldots, x_k)$. Thus, we define $\varphi(x_1 ... x_k)(A) = \{\langle a_1 ... a_k \rangle | A, < a_1 ... a_k > \models \varphi\}$. So, a response to a query will be the set tuples of elements of A that satisfy it. Let us see an example. Suppose we have a database having the following relations $Employee - Dept^{(2)}, Manager - Department^{(2)}, Manager - Secretary^{(2)}, Employee - Salary^{(2)} and >^{(2)}$ and we have the following query: 'Find all employees whose manager's secretary makes more money than the manager'. To answer this query we need one free variable x for Employee and the query can be $(\exists y)(\exists z)(\exists u)(\exists s1)(\exists s2) Employee - Department(x, y) \wedge Manager - Department(z, y) \wedge Manager - Secretary(z, u) \wedge Employee - Salary(z, s1) \wedge Employee - Salary(u, s2) \wedge > (s2, s1)$. These two ideas were fundamental enough for Ted Codd to be awarded the Turing Award. IBM's System R was the first to implement these two ideas, out of which came out SQL (standard query language).

Typically, the formulas for a database are expressed as statements in the special language SQL that is essentialy equivalent in expressive power to first-order logic. SQL is user friendly first order logic.