IN2 Modelli di Calcolo

A.A. 2001/2002

Prof. Marco Pedicini

Macchine, Programmi e Modelli

1. Computabilità, complessità e rappresentabilità

- Introduzione ai problemi di decisione, procedure algoritmiche e non algoritmiche, computazioni deterministiche, procedure discrete, nozione di alfabeto, di parola. Decidibilità e semidecidibilità di un insieme. Computazioni deterministiche, finitarie e discrete. Algoritmi formali. Esempio di formalizzazione di un algoritmo. Decidibilità per automa finito. Computazioni deterministiche, finitarie e discrete. Algoritmi formali. definizione formale di algoritmo: configurazioni di input, di output, di transizione. Esempio di formalizzazione di un algoritmo. Decidibilità per automa finito.
- Macchine di Turing: definizione, decidibilità per macchina di Turing, tempo di arresto, spazio di arresto. Costo della computazione. Indipendenza del tempo di decisione da un numero finito di configurazioni di input. Funzioni di complessità, classi di complessità DTIME e DSPACE (deterministic time e space). Inclusione $DTIME(T(n)) \subset DSPACE(T(n)) \subset DTIME(2^{cT(n)})$. Simulazione di algoritmi, simulazione della macchina di Turing a seminastro, simulazione di una macchina multinatro. Macchine di Turing speciali. Teorema di Speedup lineare per macchine di Turing con alfabeto esteso. Valutazione del coefficente di accelerazione in relazione agli alfabeti.
- Turing calcolabilità: definizione di funzione Turing calcolabile, funzioni caratteristiche di insiemi Turing decidibili, la classe delle funzioni Turing calcolabili è chiusa per composizione, coppia, ricorsione e minimizzazione. Esempi di funzioni Turing calcolabili. Funzioni Ricorsive: equivalenza tra Turing computabilità e funzioni ricorsive. Macchina Universale e indecidibilità. Il problema dell'arresto.
  ([1] capp. 1, 2, 3, 4, 5)

2. Lambda calcolo e programmazione funzionale

- Programmazione dichiarativa: cenni storici sul lambda calcolo, definizioni di base, i termini del lambda calcolo, la sostituzione semplice Relazioni sui lambda termini. Congruenze, passaggio al contesto Alpha equivalenza. L'alpha equivalenza passa al contesto. Chiusura transitiva di una relazione, proprietà di Church-Rosser. Quozientamento dei lambda-termini rispetto all'alpha equivalenza.
- Definizione di beta-redesso e di beta-riduzione. Teorema di Chuch-Rosser per la beta-riduzione. Forme normali per beta-riduzione. Strategie di beta-riduzione. Strategia normalizzante: riduzione di sinistra (left most- outer most). Riduzione

di testa. Termini Risolubili. Forme Normali di Testa. Teorema di caratterizzazione della risolubilità.

- Rappresentazione delle funzioni ricorsive: teorema di lambda definibilità Esistenza del punto fisso per il lambda termini. Punto Fisso di Church ed punto fisso di Curry.
- Teorema di Böhm o teorema del modello sintattico, separabilità tra lambda termini. Trasformazioni di Böhm: applicative e sostitutive. Rappresentazione delle trasformazioni di Böhm. Böhm-out lemma.
  ([2] capp. 1, 2, 5)

3. Modelli del lambda calcolo.

Modello funzionale dei lambda termini. Modello beta funzionale del lambda calcolo. Compatibilità tra beta-riduzione e interpretazione dei termini.

([2] cap. 7 sez. 1)

4. Paradigmi di programmazione: linguaggi funzionali ed object-oriented

- Programmazione funzionale e object-oriented.
- Il linguaggio ocaml. Tipi di base: int, float. Tipi polimorfi: ['a] list. Dichiarazione di funzioni e di funzioni ricorsive. Pattern Matching.
- Programmazione object oriented in ocaml. Dichiarazioni di classi in ocaml. Classi funzionali. Classi virtuali. Ereditarietà tra Classi. Metodi privati. Late Binding di Metodi. Invocazione di metodi.
  ([3] cap. 6, 7)

Testi consigliati

[1] Dehornoy, P., *Complexité et Decidabilité*. Springer-Verlag, (1993).
[2] Krivine, J.-L., *Lambda Calculus: Types and Models*. Masson,
[3] Sethi, R., *Programming Languages: concepts and constructs*. Addison-Wesley (ed. italiana Zanichelli),

Bibliografia supplementare

[4] Ausiello, G., Gambosi, G., d'Amore F., *Linguaggi, Modelli, Complessità*. (draft scaricabile in rete: `http://www.dis.uniroma1.it/~ausiello/InfoTeoRM/main.pdf`),
[5] Aho, Hopcroft, Ullman, *Design and Analysis of Computer Programming*..
[6] Hermes, H., *Enumerability, Decidability, Computability*. Die Grundlehren der Mathematichen Wissenshaften in Einzeldarstellungen, n. 127, Springer-Verlag, ().
[7] Darnell, P. A. and Margolis, P. E., *C A Software Engineereeing Approach*. Springer-Verlag, (1996).

Modalità d'esame

- valutazione in itinere ("esoneri")		■ SI	☐ NO
- esame finale	scritto	■ SI	☐ NO
	orale	■ SI	☐ NO
- altre prove di valutazione del profitto (meglio descritte sotto)		■ SI	☐ NO

L'esame consiste di due parti: un esame scritto e un progetto di programmazione.

La prova orale é prevista per riparare le insufficenze lievi.

Il soggetto del progetto di programmazione deve essere concordato con il docente e deve essere eseguito in ocaml, facendo uso dei costrutti funzionali ed object oriented di tale linguaggio. Il progetto può essere presentato dopo il superamento dello scritto.

Le prove di esonero (2) unitamente alla soluzione dei fogli di esercizi (3) proposti durante il corso sostituiscono la prova scritta.