Linguaggi, Compilatori e Modelli Computazionali

Anno accademico 2025-2026

Docente: Prof. Mario Bravetti
E-mail: `mario.bravetti@unibo.it`

Assistente: Dott. Lorenzo Bacchiani
E-mail: `lorenzo.bacchiani2@unibo.it`

Orario lezioni:

- Martedi' 9 – 12, Aula 2.8
- Mercoledi' 9 – 12, Aula 2.4
- Giovedi' 13 – 17, **Laboratorio 3.3** (inizio effettivo ore 13)

Ricevimento
Su appuntamento da richiedere via e-mail

Comunicazioni e Materiale didattico (su Virtuale)

Motivazione

- **Linguaggio di programmazione**: strumento per descrivere in modo rigoroso come risolvere "problemi", in modo tale che questi possano poi essere risolti da un esecutore automatico

- Ma ...
  - ... cos'è un esecutore automatico?
  - ... come progettare e realizzare un linguaggio?

- Perchè studiare queste cose?
  - non rimanere dei "passivi" utilizzatori
  - capire cosa sta dietro all'implementazione di un linguaggio
  - domain-specific languages
  - model-driven software development
  - model checking
  - ...

Linguaggi

- Linguaggi naturali/di programmazione: **lessico** e **sintassi** (Noam Chomsky)

Linguaggi

- Linguaggi naturali/di programmazione: **lessico** e **sintassi** (Noam Chomsky)

- **Compilatore**: parser + generazione codice oggetto

Linguaggi

- Linguaggi naturali/di programmazione: **lessico** e **sintassi** (Noam Chomsky)
- **Compilatore**: parser + generazione codice oggetto
- Quanto e' complesso scrivere il codice di un compilatore?

Linguaggi

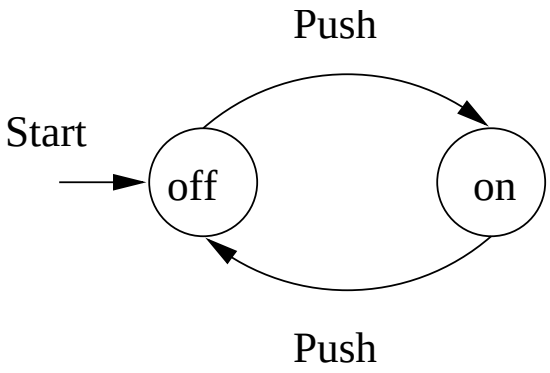
- Linguaggi naturali/di programmazione: **lessico** e **sintassi** (Noam Chomsky)
- **Compilatore**: parser + generazione codice oggetto
- Quanto e' complesso scrivere il codice di un compilatore?
- **Generazione automatica** del codice del parser/compilatore per un nuovo linguaggio tramite **approccio dichiarativo**
  - lessico: *espressioni regolari* (o automi a stati finiti)
  - sintassi: *grammatiche*, es. EBNF (o automi a pila)

Esempio di automi: Automi a stati finiti

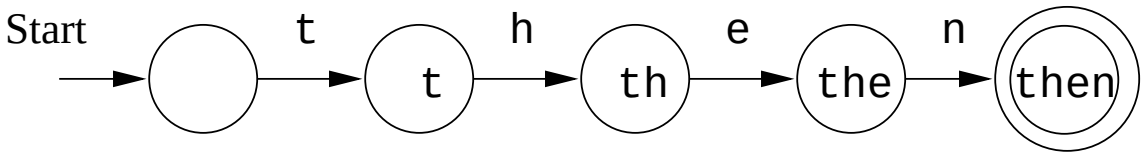
Modello per descrivere situazioni di calcolo in cui si **consumano informazioni una alla volta**, e durante la consumazione si devono memorizzare **quantità finite di informazioni**. Usati in:

- Software per la progettazione di circuiti digitali.
- Analizzatori lessicali di un compilatore.
- Ricerca di parole chiave in un file o sul web.
- Software per verificare sistemi a stati finiti, come protocolli di comunicazione.

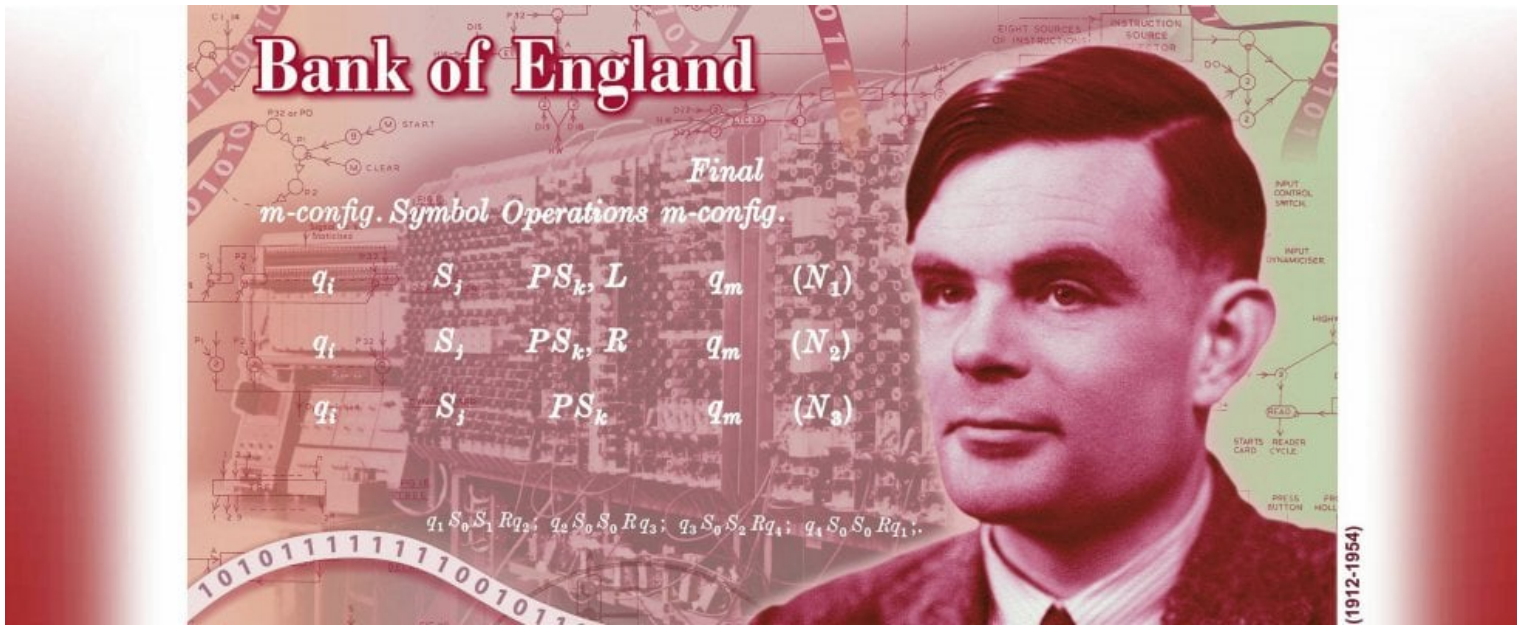
- Esempio: automa a stati finiti per un interruttore on/off



- Esempio: automa a stati finiti che riconosce la stringa `then`



Alan Turing 1912-1954
(nuova banconota da 50 sterline)



Algoritmi e Calcolabilità

- Secondo voi gli algoritmi che posso scrivere **dipendono dal linguaggio che uso**/un nuovo linguaggio che mi posso creare? (suggerimento: pensare ai compilatori)

Algoritmi e Calcolabilità

- Secondo voi gli algoritmi che posso scrivere **dipendono dal linguaggio che uso**/un nuovo linguaggio che mi posso creare? (suggerimento: pensare ai compilatori)

- **Macchina di Turing**: modello matematico di un semplice esecutore automatico/CPU (automa con memoria ausiliaria)

Algoritmi e Calcolabilità

- Secondo voi gli algoritmi che posso scrivere **dipendono dal linguaggio che uso**/un nuovo linguaggio che mi posso creare? (suggerimento: pensare ai compilatori)

- **Macchina di Turing**: modello matematico di un semplice esecutore automatico/CPU (automa con memoria ausiliaria)

- **Tesi di Turing-Church**:
  ogni problema calcolabile da un algorimo (o meglio, più in generale, tramite una procedura) può essere risolto da una Macchina di Turing

Algoritmi e Calcolabilità

- Secondo voi gli algoritmi che posso scrivere **dipendono dal linguaggio che uso**/un nuovo linguaggio che mi posso creare? (suggerimento: pensare ai compilatori)

- **Macchina di Turing**: modello matematico di un semplice esecutore automatico/CPU (automa con memoria ausiliaria)

- **Tesi di Turing-Church**:
  ogni problema calcolabile da un algorimo (o meglio, più in generale, tramite una procedura) può essere risolto da una Macchina di Turing

- Secondo voi **tutti i problemi** (es. le funzioni $\mathbb{N} \to \mathbb{N}$) sono calcolabili da un algoritmo/Macchina di Turing?

Algoritmi e Calcolabilità

- Secondo voi gli algoritmi che posso scrivere **dipendono dal linguaggio che uso**/un nuovo linguaggio che mi posso creare? (suggerimento: pensare ai compilatori)

- **Macchina di Turing**: modello matematico di un semplice esecutore automatico/CPU (automa con memoria ausiliaria)

- **Tesi di Turing-Church**:
  ogni problema calcolabile da un algorimo (o meglio, più in generale, tramite una procedura) può essere risolto da una Macchina di Turing

- Secondo voi **tutti i problemi** (es. le funzioni $\mathbb{N} \to \mathbb{N}$) sono calcolabili da un algoritmo/Macchina di Turing?

- Es. problemi **non calcolabili** (non risolvibili algoritmicamente):
  - Raggiungibilita' di una certa istruzione in un qualsiasi programma dato
  - Soddisfacibilita' di formule della logica dei predicati

Modelli Computazionali

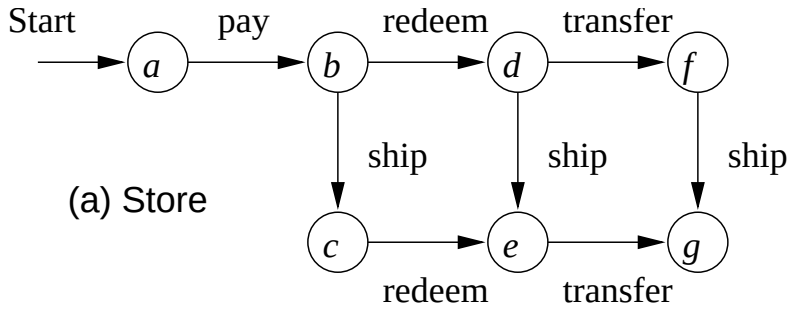
- Superamento del modello **puramente funzionale** (Macchine di Turing)
- Rappresentazione di sistemi distribuiti/concorrenti: **multipli programmi comunicanti tra loro**
- Esempio: **protocollo** per commercio elettronico

  Eventi permessi:

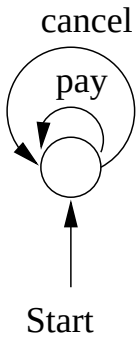
  1. Il cliente puo' *pagare (pay)* il negozio (= spedire soldi "elettronici" al negozio)
  2. Il cliente puo' *cancellare (cancel)* i soldi (come bloccare un assegno)
  3. Il negozio puo' *spedire (ship)* la merce al cliente
  4. Il negozio puo' *prendere (redeem)* i soldi (come riscuotere un assegno)
  5. La banca puo' *trasferire (transfer)* i soldi al negozio

Commercio elettronico

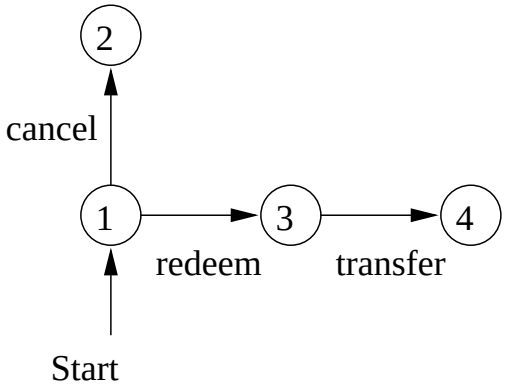
Il protocollo per **ogni partecipante**:



(a) Store

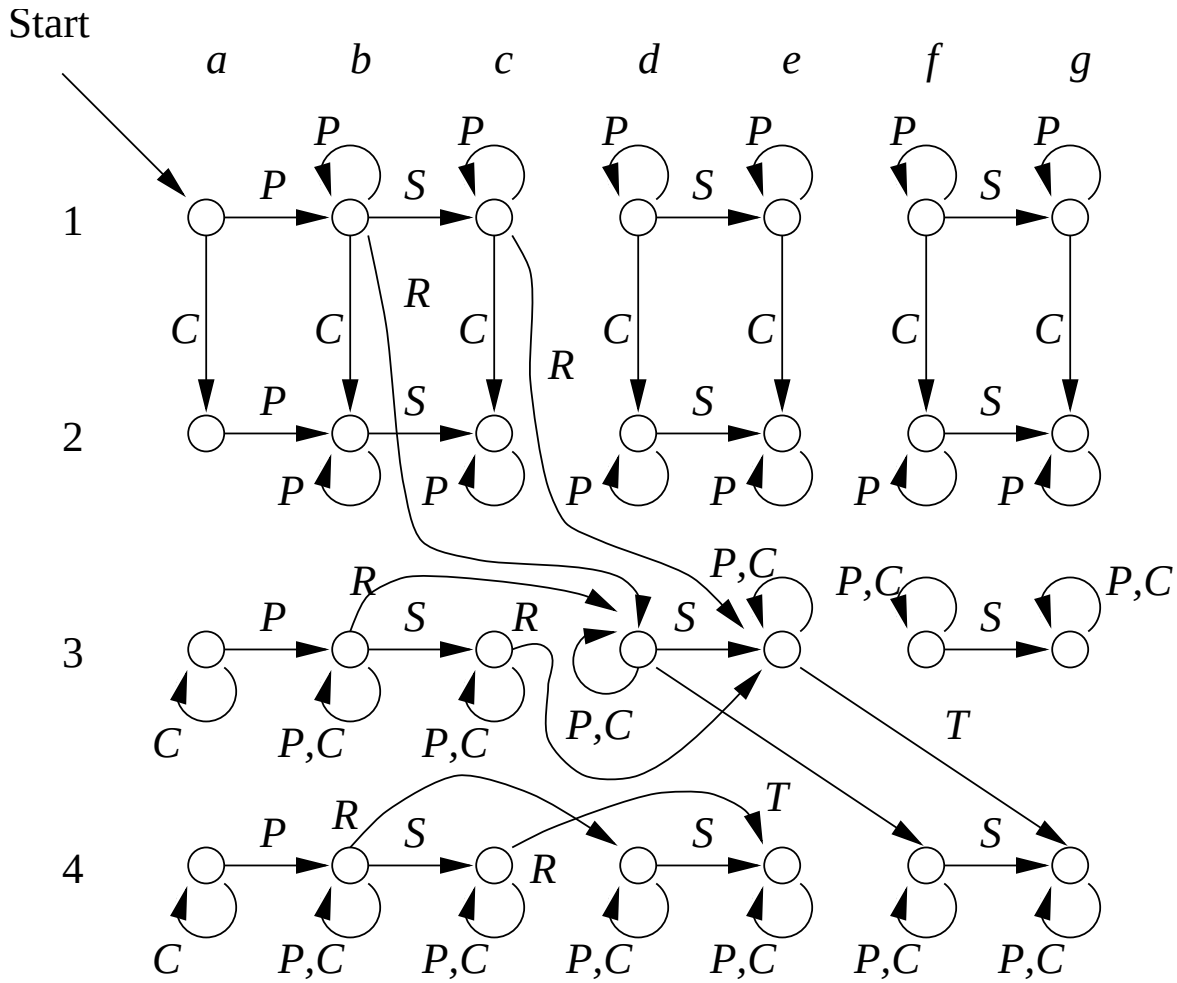


(b) Customer



(c) Bank

L'intero sistema come automa:



Logiche temporali

Permettono di esprimere **proprietà** del tipo:

> *non è possibile raggiungere uno stato in cui:*
>
> - *il negozio ha spedito (ship) e*
> - *non è possibile raggiungere il trasferimento del denaro (transfer)*

Model checker: Strumenti che verificano se **proprietà espresse tramite logiche temporali valgono oppure no**.

Se non valgono forniscono *controesempi* (computazioni che non soddisfano la proprietà)

> *pay,ship,cancel* è un controesempio per la proprietà precedente

Contenuti del corso

- **Linguaggi formali e Automi**
  - Automi a stati finiti, espressioni regolari, grammatiche libere, automi a pila, Macchine di Turing, calcolabilità
- **Compilatori**
  - Analisi lessicale, analisi sintattica, analisi semantica, generazione di codice
- **Logica di Base**
  - Logica delle proposizioni e dei predicati
- **Modelli computazionali**
  - Specifica di sistemi tramite sistemi di transizione, logiche temporali per la specifica e verifica di proprietà dei sistemi (model checking), sistemi concorrenti (algebre di processi e reti di Petri).

Laboratorio

Parte integrante del corso:

- **Supporto alla parte teorica** usando tool specifici.
  - JFLAP 7.1: http://www.jflap.org
    (automi/grammatiche/espressioni regolari), prime settimane
  - Tina 3.8.0: http://projects.laas.fr/tina
    (model checking di sistemi di transizione e reti di Petri)
  - LTSA 3.0: http://www.doc.ic.ac.uk/ltsa
    (sistemi di transizione definiti tramite algebre di processi)
- Nel resto del corso utilizzeremo un **ambiente di sviluppo per generare parser/compilatori**
  - IntelliJ esteso con plug-in ANTLRv4, ultima versione 1.24
    (generatore ANTLR: http://www.antlr.org/)

Libri di testo

- J. E. Hopcroft, R. Motwani e J. D. Ullman:
  Automi, linguaggi e calcolabilita',
  Addison-Wesley, Terza Edizione, 2009. Cap. 1–9

- A. V. Aho, M. S. Lam, R. Sethi e J. D. Ullman:
  Compilatori: principi tecniche e strumenti,
  Addison Wesley, Seconda Edizione, 2009. Cap. 1–5

- M. Huth e M. Ryan:
  Logic in Computer Science: Modelling and Reasoning about Systems,
  Cambridge University Press, Second Edition, 2004. Cap. 1–3

Esame

- **Scritto**: 27 punti

  3 appelli sessione invernale (Gennaio e Febbraio),
  3 appelli sessione estiva (Giugno, Luglio e Settembre).

  Il voto dello scritto vale solo nell'ambito della sessione.

- **Orale di progetto**: 7 punti (facoltativo)

  Realizzazione di un **compilatore** per un semplice linguaggio di programmazione funzionale object-oriented, chiamato FOOL. Estensione del **compilatore di base** realizzato in laboratorio.

  Progetto da svolgersi **singolarmente** o in **gruppo max 4 persone** e che può essere effettuato **parzialmente** (anche solo una estensione minimale della versione fatta in laboratorio).

  Discussione progetto su **appuntamento** con tutto il gruppo (dopo lo scritto, entro l'inizio della sessione successiva).