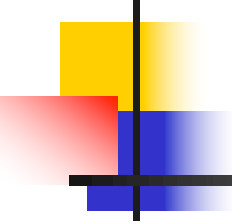




Modelli e Sistemi Concorrenti (MSC)

Laurea Magistrale in Informatica
I anno, Semestrale di 2^a ciclo, 6 cfu
Roberto Gorrieri



Programmazione (Sequenziale e/o Concorrente)

“Programming is a good medium for expressing poorly-understood and sloppily formulated ideas.” (M. Minsky)

- Ma programmare può essere particolarmente difficile nel caso di concorrenza e distribuzione: molto facile fare ragionamenti erronei
- Servono “modelli” su cui ragionare, a partire dai quali costruire i programmi “corretti”



Quali modelli?

Visione Classica = sequenziale

- Program transforms inputs into outputs.

(Denotational/Operational) Semantics:

- the meaning of a program is a partial function on stores (*assignment of values to program variables*): $stores \rightarrow stores$
- Non-termination is bad! (**while** true **do** skip)
- In case of termination, the result is unique.

Is this all we need?



Programmi sequenziali come funzioni (1)

$\llbracket - \rrbracket$: Programmi \rightarrow Dominio di funzioni (**Store** \rightarrow **Store**)

Ad esempio: $P_1 = x := 0$ $P_2 = x := x+1$ $Q = x := 1$

$\llbracket P_1 \rrbracket = \lambda s. s[x/0]$ $\llbracket P_2 \rrbracket = \lambda s. s[x/s(x)+1]$ $\llbracket Q \rrbracket = \lambda s. s[x/1]$

Composizionalità: per ogni operatore **sintattico** deve esistere un corrispondente operatore funzionale **semantico**

Ad esempio, $P = P_1 ; P_2$ (**composizione sequenziale**)

$\llbracket P \rrbracket = \llbracket P_1 ; P_2 \rrbracket = \llbracket P_2 \rrbracket \circ \llbracket P_1 \rrbracket = \lambda s. s[x/1]$

(**composizione di funzioni**)



Programmi sequenziali come funzioni (2)

Equivalenza tra programmi: stessa funzione calcolata

$$Q_1 \simeq Q_2 \Leftrightarrow \llbracket Q_1 \rrbracket = \llbracket Q_2 \rrbracket$$

$$\text{ad esempio } \llbracket P \rrbracket = \llbracket Q \rrbracket = \lambda s. s[x/1]$$

Congruenza: equivalenza preservata dagli operatori

se $Q_1 \simeq Q_2$, allora per ogni operatore, ad esempio $- ; -$, e per ogni altro programma P , deve valere

$$Q_1 ; P \simeq Q_2 ; P \quad \text{ed anche} \quad P ; Q_1 \simeq P ; Q_2$$



Sistemi Reattivi

Caratterizzazione di Sistemi Reattivi

Un sistema che computa reagendo a stimoli provenienti dal suo ambiente

- ❑ Macchina distributrice di caffè
- ❑ Sistema Operativo
- ❑ Protocolli di comunicazione
- ❑ Sistema di controllo di centrale nucleare
- ❑ Programma di gestione del traffico ferroviario
- ❑ Software embedded nei telefoni mobili ETC....



Sistemi Reattivi (2)

- Esempio astratto di programma di controllo:

loop

read the sensors at regular intervals

depending on the sensors' values, trigger the relevant actuators

forever



Sistemi Reattivi (3)

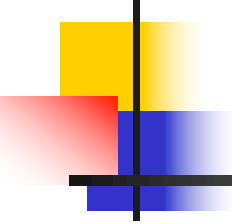
Key Issues (differenti dal caso sequenziale):

- ❖ Communication / interaction / synchronization / timing / parallelism
- ❖ Non-termination is good!
- ❖ Non-determinism is good! (The result (if any) has not to be unique)



Classical vs. Reactive Computing

	Classical / sequential	Reactive / parallel
interaction	no	yes
nontermination	undesirable	Often desirable
Unique result	yes	no
semantics	stores \rightarrow stores	????



Perchè la semantica di un sistema concorrente non è una funzione? (1)

$P = x:=0; x:=x+1$ $Q = x:=1$ $\llbracket P \rrbracket = \llbracket Q \rrbracket = \lambda s. s[x/1]$

Qual è la semantica di $\llbracket Q|Q \rrbracket$?

Forse la funzione da store a store $\lambda s. s[x/1]$? Sembra così.

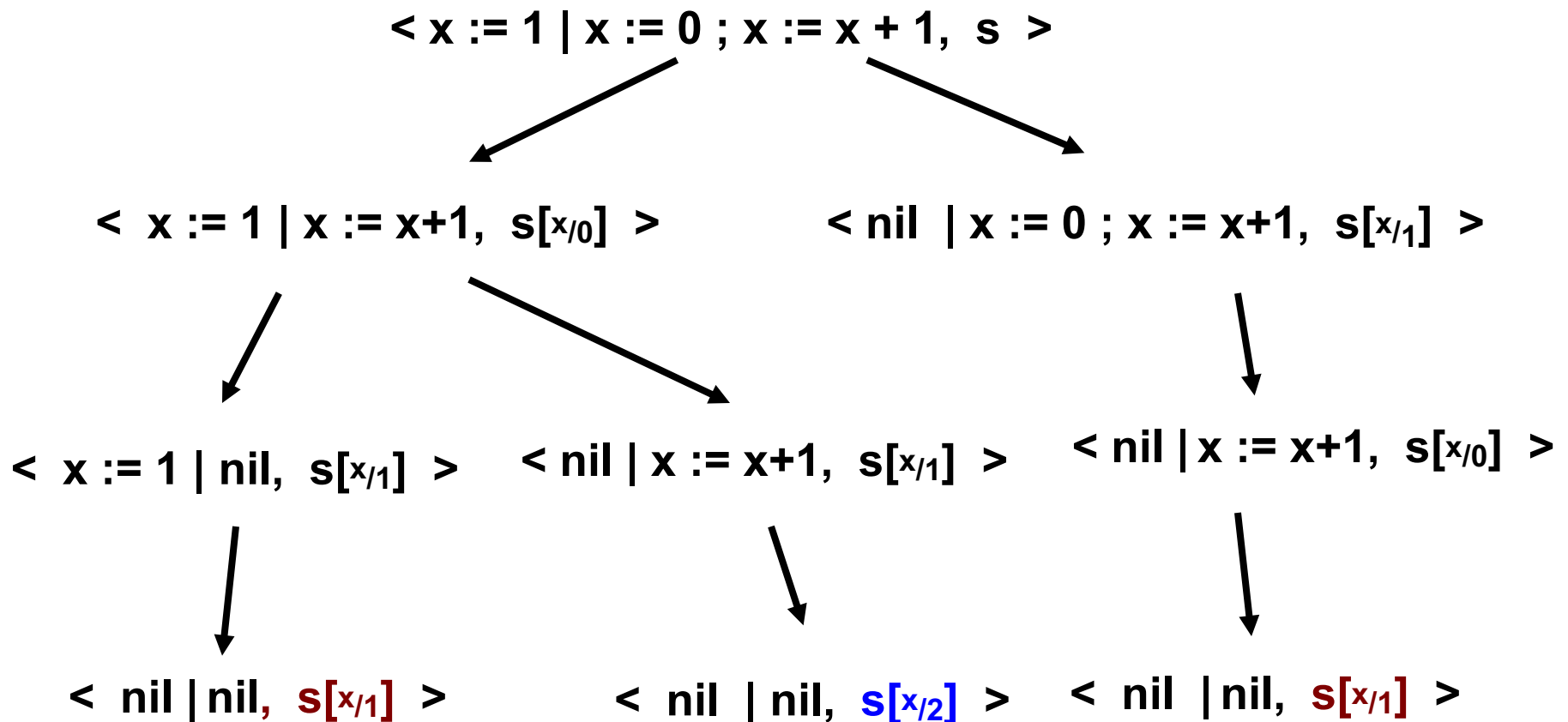
Considera $Q|P$ e guardiamo il suo tracciato operativo
(prossimo lucido):

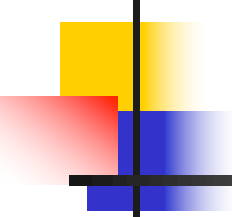
Qual è la semantica di $\llbracket Q|P \rrbracket$?

Forse $\lambda s. (s[x/1] + s[x/2])$? (dove $+$ rappresenta una forma di
nondeterminismo = insieme di possibili stores)



Tracciato Operazionale





Perchè la semantica di un sistema concorrente non è una funzione? (2)

$$P = x:=0; x:=x+1 \quad Q = x:=1 \quad \llbracket P \rrbracket = \llbracket Q \rrbracket = \lambda s. s[x/1]$$

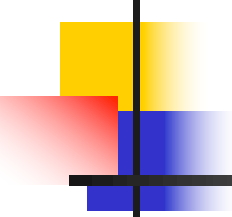
Ma allora \simeq **non** è una **congruenza**! $Q \simeq P$, ma $Q|Q$ non è equivalente a $Q|P$.

Composizionalità: siamo in gradi di definire l'operatore semantico \parallel di parallelo di funzioni? NO:

$$\llbracket Q|Q \rrbracket = \llbracket Q \rrbracket \parallel \llbracket Q \rrbracket = \lambda s. s[x/1]$$

$$\llbracket Q|P \rrbracket = \llbracket Q \rrbracket \parallel \llbracket P \rrbracket = \lambda s. (s[x/1] + s[x/2])$$

Ma $\llbracket P \rrbracket = \llbracket Q \rrbracket$ e quindi \parallel non è definibile come operatore su funzioni (semantica non composizionale).



Perchè la semantica di un sistema concorrente non è una funzione? (3)

$P = x:=0; x:=x+1 \quad Q = x:=1 \quad \llbracket P \rrbracket = \llbracket Q \rrbracket = \lambda s. s[x/1]$

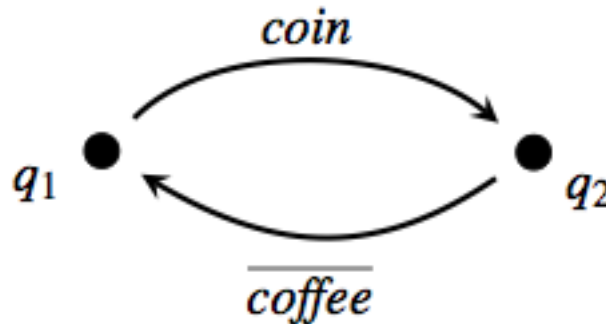
P e Q non dovrebbero essere considerati equivalenti, perchè diverso è il modo in cui operano/interagiscono sulla memoria: gli stati intermedi sono rilevanti!

Programmi che non terminano: operating system, vending machine, railway control system, ... non producono risultati finali, ma offrono servizi per sempre. Una semantica funzionale li considererebbe tutti equivalenti!

Per tutti questi motivi la semantica di sistemi concorrenti reattivi non può essere descritta da funzioni.

Quali modelli per sistemi reattivi? (1)

- Azioni, interazioni, cambiamento di stato come mattoni primitivi con cui costruire il modello
- Modelli a stati/transizioni simili agli **automi** studiati in teoria dei linguaggi formali

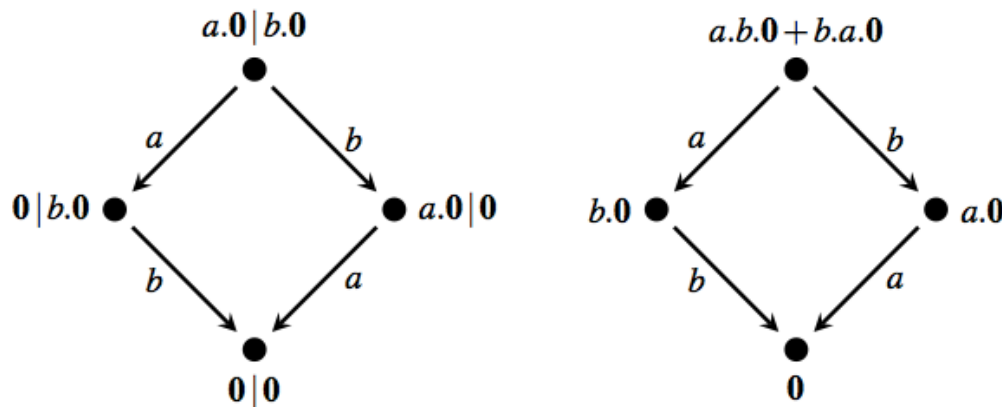


- La **semantica** di un sistema reattivo è l'automa, modulo un'opportuna nozione di equivalenza \sim , ovvero **la classe d'equivalenza dell'automa rispetto a \sim**

Quali modelli per sistemi reattivi? (2)

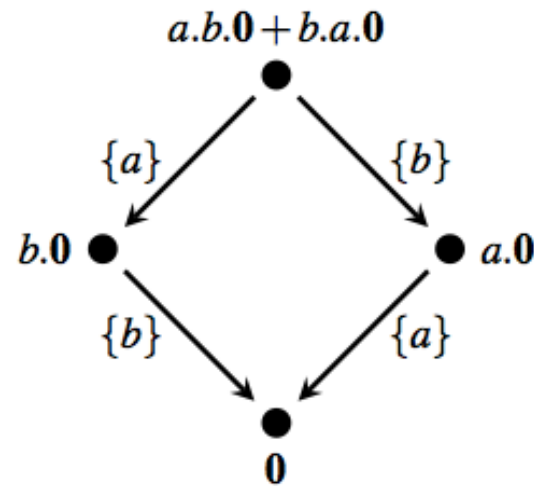
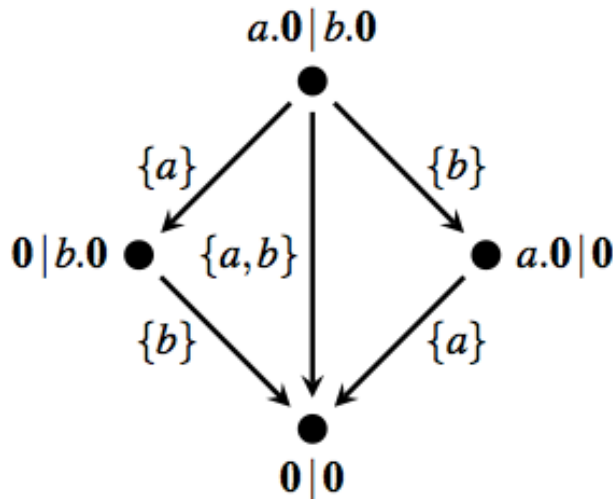
- Automi (sistemi di transizioni etichettate): **modello sequenziale/interleaving**
 - una sola cosa può essere fatta in ogni passo.
 - Lo stato è monolitico (non vedo i componenti di un sistema distribuito)

Conseguenza: parallelismo non primitivo (interleaving law)



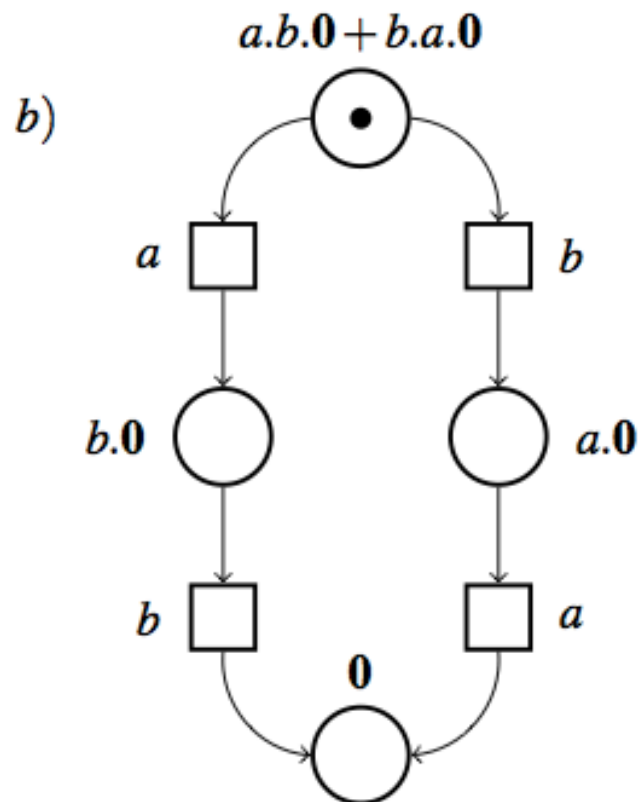
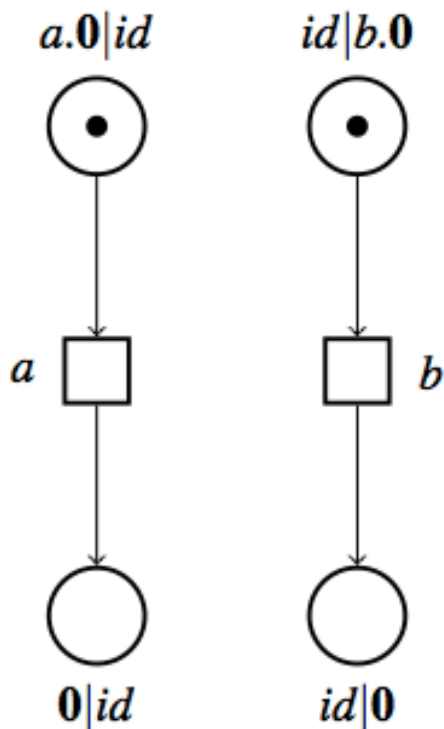
Quali modelli per sistemi reattivi? (3)

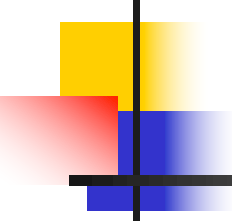
- **Modello parallelo:** automi con transizioni etichettate da multinsiemi di azioni
 - Vedo il parallelismo delle attività
 - Non vedo la distribuzione del sistema



Quali modelli per sistemi reattivi? (4)

- Modello distribuito: reti di Petri





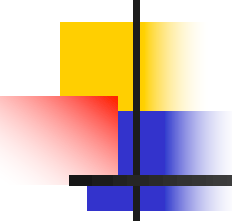
Progetto, analisi ed implementazione di sistemi reattivi (1)

Confronta cosa avviene in Architettura/Ingegneria edile

Prima il progetto del ponte, poi la prova di correttezza del progetto (e.g., rispetta la portata), infine la realizzazione del ponte.

Still disasters may happen:

- London Millennium Bridge, oscillation/resonance: Simulation fails because of wrong estimates for pedestrian forces, 2000
- Tacoma Bridge – aeroelastic flutter due to wrong estimates of wind forces, 1940



Progetto, analisi ed implementazione di sistemi reattivi (2)

What happens in Computer Science

- Common practice: Go directly to the implementation!
- Too often the design phase is missing or only very sketchy and informal
- Correctness by testing a posteriori: “testing can be used to show the presence of bugs, but never to show their absence” (Dijkstra 1969)



Fact of Life

Even short parallel programs may show unexpected behavior and may be hard to analyze



Necessità di una **teoria** di **progettazione** di sistemi reattivi (critici!)

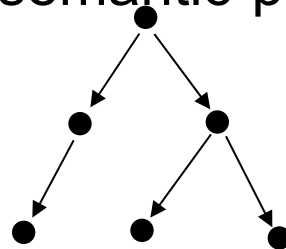
Abbiamo bisogno di metodi e strumenti sistematici, basati su solide teorie matematiche, altrimenti
“Failure Stories”☹

- Intel’ s Pentium-II bug in floating-point division unit (1994)
- Ariane-5 crash due to a conversion of 64-bit real to 16-bit integer (1996)
- Mars Pathfinder problems with the real-time OS (1997)
- ... Many more (airplane disasters!) ... See, e.g.,
<http://www5.in.tum.de/~huckle/bugse.html>
<http://www.cs.tau.ac.il/~nachumd/verify/horror.html>

Perchè è utile una solida teoria di base?

Esempio: **compilatore** = semantic-preserving translator

Programma
sorgente



Albero di Sintassi Astratta
(Struttura semantica)

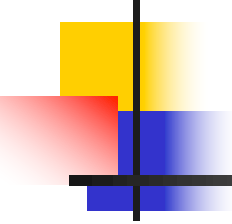


Programma
eseguibile

- Il primo compilatore FORTRAN fu scritto impiegando **18 anni-uomo!** (prima della teoria!) (Backus)
- Oggi: progetto per studenti di Linguaggi! Perché?

- 1) Comprensione di come organizzare e modularizzare il processo di compilazione (scanner, parser, ...)
- 2) Scoperta di tecniche sistematiche per gestire specifici aspetti (minimizzazione, ...)
- 3) Sviluppo di software tools per facilitare la realizzazione di compilatori (Lex, YACC)

Teoria dei linguaggi formali (ormai stabile)

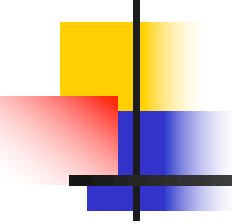


Progetto, analisi ed implementazione di sistemi reattivi (3)

Perché in informatica spesso non si pretende un progetto verificato del sistema che si intende realizzare?

- Come progettare un sistema che funzioni? (enfasi su progetto, non su implementazione)
- Come possiamo definire i requisiti che il sistema deve soddisfare?
- Come possiamo verificare che tale progetto è corretto? (a dispetto di limiti teorici intrinseci)

Quali strumenti matematici si possono usare?



Progetto, analisi ed implementazione di sistemi reattivi (4)

- How to design a reactive system?

Build a mathematical model
(automata-based representation)

- How to define the requirements that the system has to satisfy?

Use a specification logic

- How can we verify that the design is correct?

Model-checking: check that the
logical formula is satisfied by the model



Formally Verified Hardware/Software?

- Once the design is proved correct (and this can be done for **finite-state** systems), implement the system
- **Conformance** of the implementation to the design. This is not always formally verifiable: e.g., a design can be finite-state but the implementation can be infinite-state. Also there may be a big abstraction level gap between the design and the implementation (specific techniques needed)!



“Success Stories” ... 😊

- Formal verification is used by most leading hardware companies (Intel was the first after its Pentium bug in 1994).

Nowadays hardware is formally verified!

- Formal verification in the software industry is still languishing. For many reasons (including the explicit handling of data), the state space describing a system implementation may be infinite!

So, in some cases, we are forced to live with partially unverified software.



Obiettivi del corso (1)

Presentare una teoria generale dei sistemi reattivi e relative applicazioni.

- Progetto del sistema (possibilmente un prototipo eseguibile)
- Specifica delle proprietà di interesse (formule logiche)
- Verifica (possibly automatic & compositional) che il progetto soddisfa la specifica (problema della correttezza!)



Obiettivi del corso (2)

- Illustrare alcuni principi e tecniche per la **progettazione** e la **verifica** di sistemi concorrenti.
- Utilizzo di **linguaggi di specifica** molto semplici e paradigmatici:
CCS e, parzialmente, sue estensioni
- Specifica e verifica di proprietà con opportune **logiche**: HML e mu-calcolo
- Esercitazioni pratiche con **tool semi-automatici di verifica**: CWB



Inoltre accenneremo a

- Studiare il problema dell'**espressività** di vari linguaggi per descrivere sistemi reattivi
- Sottoclassi di CCS e confronto con la **gerarchia di Chomsky**
- Estensioni di CCS con operatori addizionali (derivati e non)
- **Limitazioni di CCS:** non è in grado di **risolvere** problemi ben noti (**filosofi a cena**, in modo fully-distributed e simmetrico), per mancanza di sincronizzazione multi-party. Estensione a Multi-CCS



Inoltre accenneremo a ... (2)

- Per la **tesi di Church-Turing**, due linguaggi sequenziali sono equivalenti e massimamente espressivi se e soltanto se sono entrambi **Turing-completi** (ovvero calcolano tutte le funzioni calcolabili con MdT)
- **Last Man Standing problem**: problema “distribuito” non risolvibile in CCS (che è Turing-completo) ma lo è in altri linguaggi, anche non Turing-completi.
- **Oltre la Turing-completezza**: quale criteri usare per confrontare l'**espressività** di linguaggi concorrenti?



Inoltre accenneremo a ... (3)

- Come confrontare l'espressività di linguaggi concorrenti? Problema aperto.
- Esempio di **gerarchia di modelli distribuiti** (reti di Petri) e **corrispondente gerarchia di linguaggi associati** (analogo, per sistemi distribuiti, della gerarchia di Chomsky, proposta nel 1959 su sistemi sequenziali).
- Altre limitazioni di CCS: non è in grado di modellare sistemi la cui architettura si riconfigura dinamicamente nel tempo.
- **Mobilità: π -calcolo** (solo accennato)



Struttura e contenuti del corso (1)

- Teoria classica dei sistemi reattivi

Sistemi di transizione, Equivalenze, Bisimulazione, CCS, Semantica Operazionale, proprietà algebriche, assiomatizzazioni (circa 22 ore in tutto)

- Verifica Formale

Logiche (Hennessy-Milner Logic), anche con ricorsione (teoria del punto fisso), Tool di verifica (Concurrency Workbench). Analisi dell'algoritmo di mutua esclusione di Peterson (circa 8)

- Cenni ad espressività ed estensioni

- Sotto-calcoli di CCS, BPA e varianti, Multi-CCS, CCS value-passing, π -calcolo (circa 6 ore)



Struttura e contenuti del corso (2)

- **Teoria classica dei sistemi distribuiti**
Reti di Petri, equivalenze, proprietà decidibili (circa 6 ore)
- **Cenni a linguaggi per reti di Petri**
Definire la gerarchia di reti e di linguaggi, semantica distribuita dei linguaggi, calcolabilità distribuita. (circa 4 ore)
- **Caso speciale: semantica distribuita per reti BPP**
Team bisimulation, Logica TML, assiomatizzazione (circa 4 ore)



Calculus of Communicating Systems (CCS)

Teoria degli Automi

Comunicazione



CCS

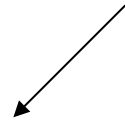
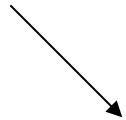
- Robin Milner, Edinburgo, fine anni ' 70 — **Turing Award 1991**
- Milner, “Communication and Concurrency”, Prentice-Hall, 1989.



CCS distribuito

Reti di Petri finite

Comunicazione



Sotto-calcoli di CCS
“distribuito” (con estensioni)

- Roberto Gorrieri, “Process Algebras for Petri Nets --- The Alphabetization of Distributed Systems”, Springer, 2017.



Materiale didattico

Testi di studio

- ***Introduction to Concurrency Theory: Transition Systems and CCS***, R. Gorrieri, C. Versari, EATCS texts in Theoretical Computer Science, Springer, 2015
- ***Reactive Systems: Modelling, Specification and Verification***

Luca Aceto, Anna Ingolfsdottir, Kim Guldstrand Larsen, Jiri Srba
Cambridge University Press, August 2007

Testo di consultazione

- ***Process Algebras for Petri Nets --- The Alphabetization of Distributed Systems***, R. Gorrieri, EATCS monographs in Theoretical Computer Science, Springer, 2017



Altro materiale didattico

- Pagina di riferimento per il corso:
<https://virtuale.unibo.it/course/view.php?id=46126>
- Tutto il materiale didattico verrà qui reso disponibile (lucidi delle lezioni in pdf)
- Concurrency Workbench
<http://homepages.inf.ed.ac.uk/perdita/cwb/>



A Lezione:

- Fate domande e/o rispondete alle mie domande: Be active!
- Prendete appunti! (anche sui lucidi, che vi distribuisco in anticipo)
- Fate a casa gli esercizi del libro e/o quelli che vi lascio di volta in volta!



Orario delle lezioni

Ora inizio	lun	mar	mer	gio	ven
11:00-12:00	E1				
12:00-13:00	E1				
13:00-14:00				E2	
14:00-15:00				E2	
15:00-16:00				E2	
16:00-17:00					
17:00-18:00					



Prova d'esame

- **Scritto = progetto** (anche di **gruppo**) + **Orale** (**individuale**)
- Appelli su appuntamento, da concordare per email
- Scritto/progetto: verrà assegnato a fine lezioni, da svolgere a casa e da presentare (via email) almeno 3 giorni prima dell'orale.



Info utili

- Email
 - roberto.gorrieri@unibo.it
- Pagina web
- <https://virtuale.unibo.it/course/view.php?id=46126>
- Controllare regolarmente la pagina web del corso
- Ricevimento
 - Da concordare (previa richiesta per email)