Tecniche di Specifica e di Verifica

Introduzione

Esrcitazioni

- Esercizi per casa
  - Durante la prima metà
- Esercizi di verifica
  - Durante la seconda metà
- Prevedono l’uso del tool **NuSMV**.

Materiale per il corso

- Libro di testo:

  Clarke, Grumberg and **Peled**. "***Model Checking***"

  MIT Press

- Slides delle lezioni

- Altro materiale, ove necessario, fornito durante il corso.

Prerequisiti

- Basi solide di Matematica (Algebra).
- ***Logica***, ***Semantica dei Linguaggi***, Ingegneria del Software, Linguaggi Formali.
- Consigliati: Logiche per la Rappresentazione della Conoscenza
- Interesse per sistemi reattivi e ... per la loro verifica .

Obiettivi

- Comprensione del problema della verifica di sistemi.
- Introduzione alle tecniche e metodologie della verifica formale.
- Esperienza con tool per la verifica formale.
- Cenni alle recenti tendenze della ricerca in verifica formale.

Lezione di oggi

- Introduzione alla verifica formale.
- Schema di massima del corso….
- … soggetto a modifiche in corso d’opra!

Cosa riguarda il corso ?

- L’argomento generale è:
  - la ***Verifica formale***.
- Per alcuni sistemi (di computazione) è cruciale verificare la loro ***correttezza*** prima che vengano utilizzati.
- Sistema di computazione:
  - Circuiti, unità aritmetiche, CPU, programmi, compilatori…..

Proprietà dei sistemi

- ***Availability***: la probabilità che un sistema sia funzionante e in grado di fornire i servizi nel tempo
- ***Reliability***: la probabilità, in un dato intervallo di tempo, che un sistema fornisca correttamente i servizi come atteso dagli utenti
- ***Safety***: è una valutazione di quanto sia verosimile che il sistema causi danni a persone o all'ambiente.
- ***Security***: è una valutazione di quanto sia verosimile che il sistema possa resistere ad intrusioni accidentali o premeditate.

Tipi di sistemi critici

- ***Safety-critical***: sistemi il cui fallimento (***failure***) può causare ferimenti, ***perdite di vita***, o seri danni ambientali (e.g. sistema di controllo per un impianto chimico).
- ***Mission-critical***: sistemi il cui fallimento (failure) può causare il fallimento di ***attività guidate da obiettivo***. (e.g. sistema di navigazione di una navetta spaziale)
- ***Business-critical***: sistemi il cui fallimento (failure) può causare ***ingenti perdite di denaro*** (e.g. sistema di gestione dei conti correnti in una banca).

Il caso Therac-25

- Il fatto: ***Canada-USA 1985-1987***. 3 persone uccise e 3 gravemente ferite (di cui una paralizzata) per i malfunzionamenti di una macchina per l'irradiazione dei tumori, il ***Therac-25***.
- Il problema: "***overdose***" di radiazioni.
- Le cause: **errori nel sistema SW**, e di **interfacciamento SW/HW** (erronea integrazione di componenti SW preesistenti nel Therac-20).

Il caso di Denver

- ***Aeroporto di Denver*** - Progettato per essere un aeroporto all'avanguardia, provvisto di un complesso sistema di gestione bagagli computerizzato e 5.300 miglia di cablaggio in fibre ottiche.
- Sfortunatamente, ***errori nel sistema gestione bagagli*** causavano lo schiacciamento delle valige e guidavano i carrelli automatici contro i muri.
- L'aeroporto inaugurò con ***16 mesi di ritardo***, con una perdita di ***3.2 miliardi di dollari***, e con un sistema bagagli essenzialmente manuale.

Il Pentium 5

- ***Il baco del Pentium*** - Il Professor Thomas Nicely del Lynchburg College in Virginia scoprì che il chip Pentium dava risposte scorrette per certe complesse equazioni (**1994**).
- L'errore era dovuto a un difetto di progettazione dell'algoritmo di divisione in floating point (***FDIV***) del processore.
- Intel fu costretta a ritirare il chip e a sostituirlo. Il tutto costò alla Intel ***450 milioni*** di dollari.
- **Dal 1994 Intel usa le Tecniche di Verifica Formale!**

Il caso AT&T

- ***AT&T*** - errori nel software dei computer che gestivano le chiamate telefoniche, causarono uno scollegamento di 9 ore per tutte le reti a lunga distanza della compagnia.
- Fu il peggior black-out, tra i tanti subiti nella storia del sistema. Coinvolse centinaia di servizi.
- Alla fine la causa venne individuata in una ***singola linea di codice errata*****.**

L'Ariane 5

- Il fatto: **4 giugno 1996**, il volo di prova dell'***Ariane 5*** risultò un fallimento. Dopo solo 40 secondi dall'inizio della sequenza di volo, ad un'altitudine di 3700 m, il razzo deviò dalla rotta, si separò ed esplose (***500M$***).
- La causa: gli ***Inertial Reference Systems*** cessarono di funzionare simultaneamente dopo circa 36.7 secondi. L'eccezione del software ***SRI*** si verificò durante la ***conversione di dati da floating point a 64-bit a signed integer a 16-bit***. Il numero floating point da convertire aveva un valore superiore a quello rappresentabile con un signed integer di 16-bit.
- Il risultato fu una eccezione di "**Operand Error**"….

L’Ariane 5 (cont’d)

- … il pacchetto di navigazione era stato ***ereditato dall’Ariane 4 senza eseguire un testing accurato***. Il nuovo razzo volava più velocemente, fornendo al il software di navigazione valori maggiori per alcune variabili.
- La sfortuna fu che il codice che generò l’eccezione forniva ***informazioni utili solo nella fase precedente alla partenza***; se fosse stato disabilitato al momento del lancio, non ci sarebbero stati problemi.

Verifica di correttezza

- È una pratica ingegneristica fondamentale.
- Gli errori possono essere molto costosi
  - il “baco” del Pentium.
  - il Razzo Ariane 5
- Ci sono molti sistemi la cui correttezza è critica, ad esempio i sistemi “***safety-critical***”.

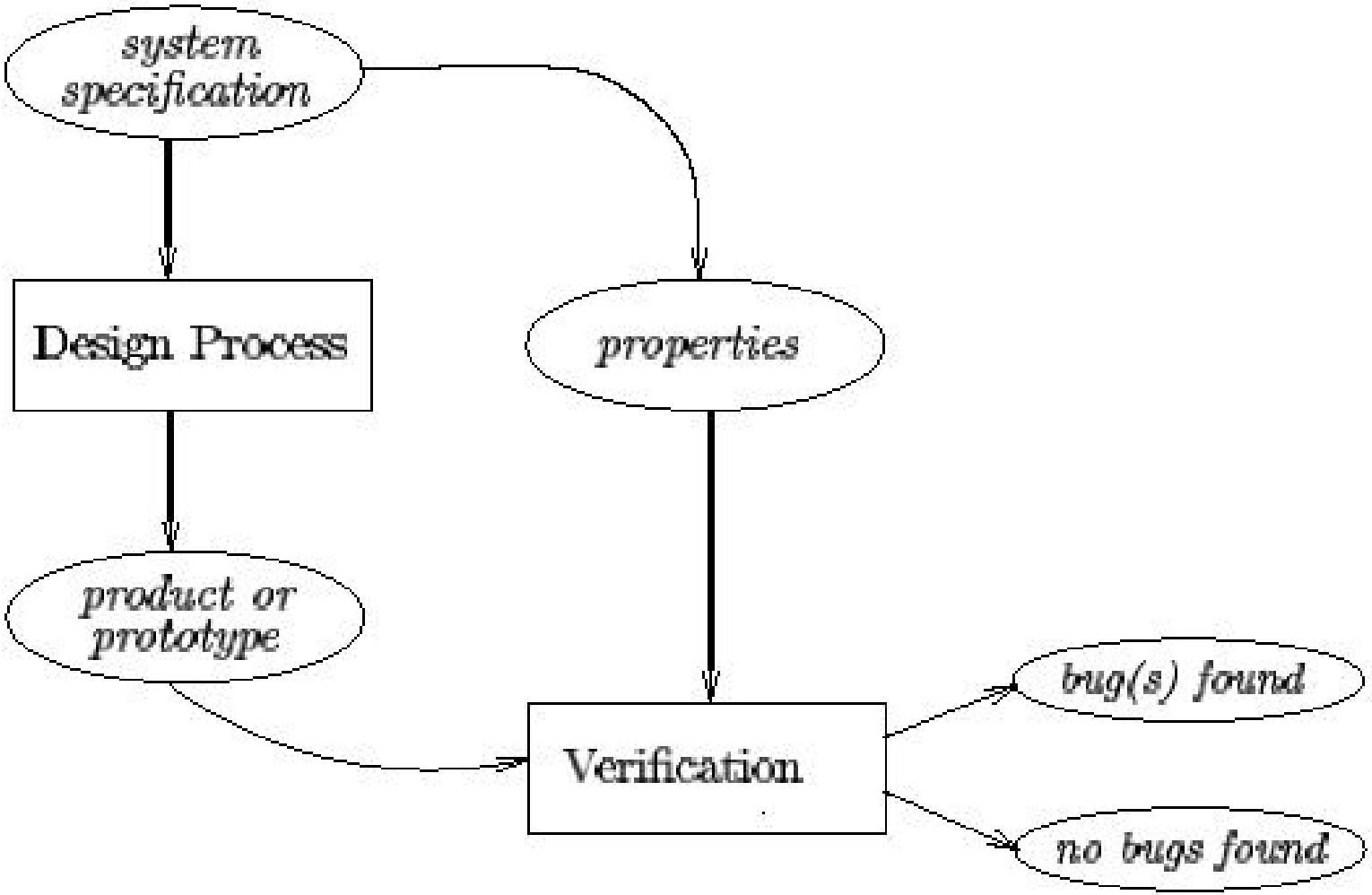
Necessità di correttezza

- Sistemi safety-critical:
  - Medici
  - Controllo di reattori
  - Controllo ferroviario
  - Controllo aereo
  - ……

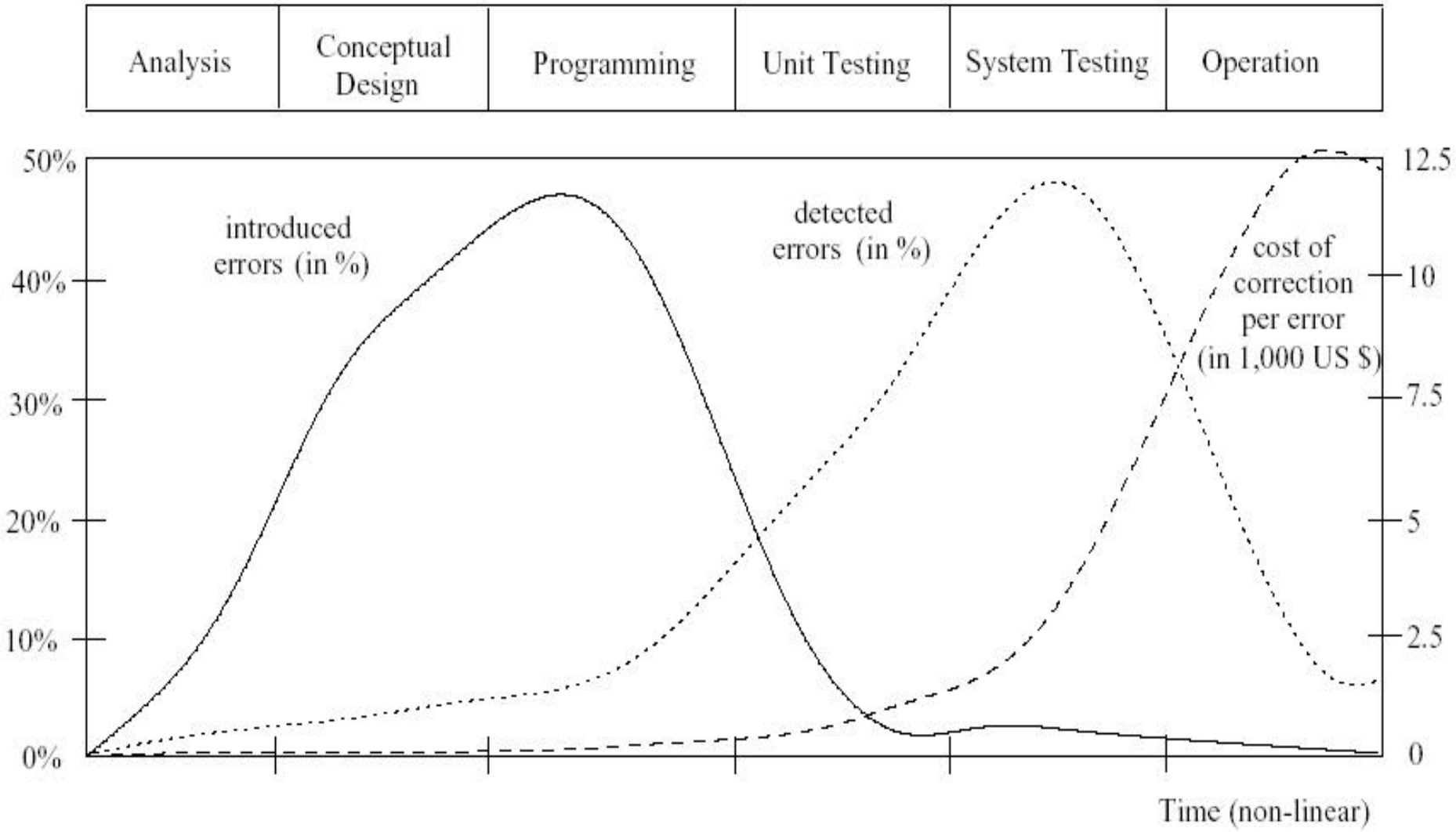
Società che usano i Metodi Formali

- Intel
- Siemens
- AT&T
- Lucent
- AMD
- IBM
- Motorola
- Nasa
- British Telecom (BT)
- etc….

Verifica a posteriori



Errori: introduzione, individuazione e costi



Tecniche correnti

- Simulazione
- Testing
- Sempre più frequente e diffusa:
  - ***Verifica formale***
- In pratica le più utilizzate sono però la simulazione e il testing.

Simulazione e testing

- ***Simulazione*** : Un modello ***astratto*** del sistema viene simulato inclusi :
  - Dati di input
  - Eventi esterni
- ***Testing*** : Il sistema implementato viene eseguito su insiemi selezionati di input e eventi esterni, e verificato.

Svantaggi

- La simulazione non può essere eseguita per sempre.
- La simulazione è molte volte più lenta del sistema reale.
- Può essere motlo costosa.
- Nessuna garanzia che ***tutte le esecuzioni possibili*** vengano simulate.

Svantaggi

- Anche il testing soffre di svantaggi simili.
- In pratica, non tutte le configurazioni di input possono essere presentate al sistema.
- Gli insiemi di input devono essere generati automaticamente.
- Nessuna garanzia che gli input “cattivi” vengano mai presentati.
- Molto difficile soprattutto per sistemi ***concorrenti***
  - Sistemi ***multi-componente***.

Svantaggi

- La simulazione e il testing possono rivelare la ***presenza*** di bachi ma non possono mai stabilire l'***assenza*** di bachi (Dijkstra negli anni '70).
- Questa è una limitazione fondamentale nel caso di sistemi "safety-critical".
- Ma in pratica …..

Metodi Formali

I ***Metodi Formali*** hanno come obiettivo fornire:

- la capacità di esprimere specifiche in maniera precisa, e
- la capacità di definire in modo chiaro quando un'implementazione soddisfa le specifiche ("correttezza")

La Verifica Automatica

La ***verifica automatica***:

- l'applicazione del ***ragionamento logico*** allo sviluppo di sistemi computerizzati (software, hardware, applicazioni web distribuite,….)
- l'uso di tools informatici per aiutare questo ragionamento logico

Ragionamento logico e verifica

- Il ragionamento logico permette di dedurre proprietà dei sistemi dandoci un modo di "verificare" il nostro lavoro, e fornisce un punto di vista diverso sul sistema.
- Nella verifica formale le proprietà vengono controllate per "tutti" i possibili comportamenti del sistema. L'analisi è ***esaustiva*** ma eseguita su un ***modello formale*** del sistema, non sul sistema reale.
- Il ragionamento simbolico rende possibile l'analisi esaustiva.

Ragionamento logico e verifica

- La verifica corrisponde a decidere la ***relazione di soddisfacimento*** tra un modello e una formula:

$$\mathcal{M} \models f$$

- $\mathcal{M}$ è un modello che descrive i possibili comportamenti del sistema
- f è la proprietà o specifica che il sistema deve soddisfare
- $\models$ è una relazione che deve sussistere tra $\mathcal{M}$ e f.
- Il ragionamento logico ci permette di decidere se la relazione vale o meno tra il modello e la proprietà.

Verifica Formale

La ***verifica*** è:

- ***formale***: il proprietà di correttezza è una proposizione matematica precisa - sia il modello che la proprietà sono non ambigue
- ***definitiva***: la verifica o dimostra oppure refuta la proprietà di correttezza

opposta a

- ***testing*** del sistema reale su input selezionati
- ***simulazione*** di un modello su input selezionati
- ***ispezione manualmente*** del codice o di un modello

Approcci alla Verifica Formale

- ***Verifica deduttiva*** e/o ***Model Checking***.
- ***Verifica deduttiva*** (o ***Theorem proving***):
  - Usa assiomi e regole di dimostrazione per modellare il sistema (***sistema formale***).
  - La proprieta da verificare viene espressa da un ***teorema*** del sistema formale.
  - Derivare il teorema con l'aiuto di un ***dimostratore automatico*** (***Theorem-prover***)

Svantaggi

- Molto difficile da automatizzare (***spesso impossibile in teoria***).
- Richiede l'intervento dell'utente.
- Derivare il sistema formale può essere piuttosto scomodo.
- Raramente ***effettiva*** (fattibile).
- Richiede un esperto per usare il theorem-prover.

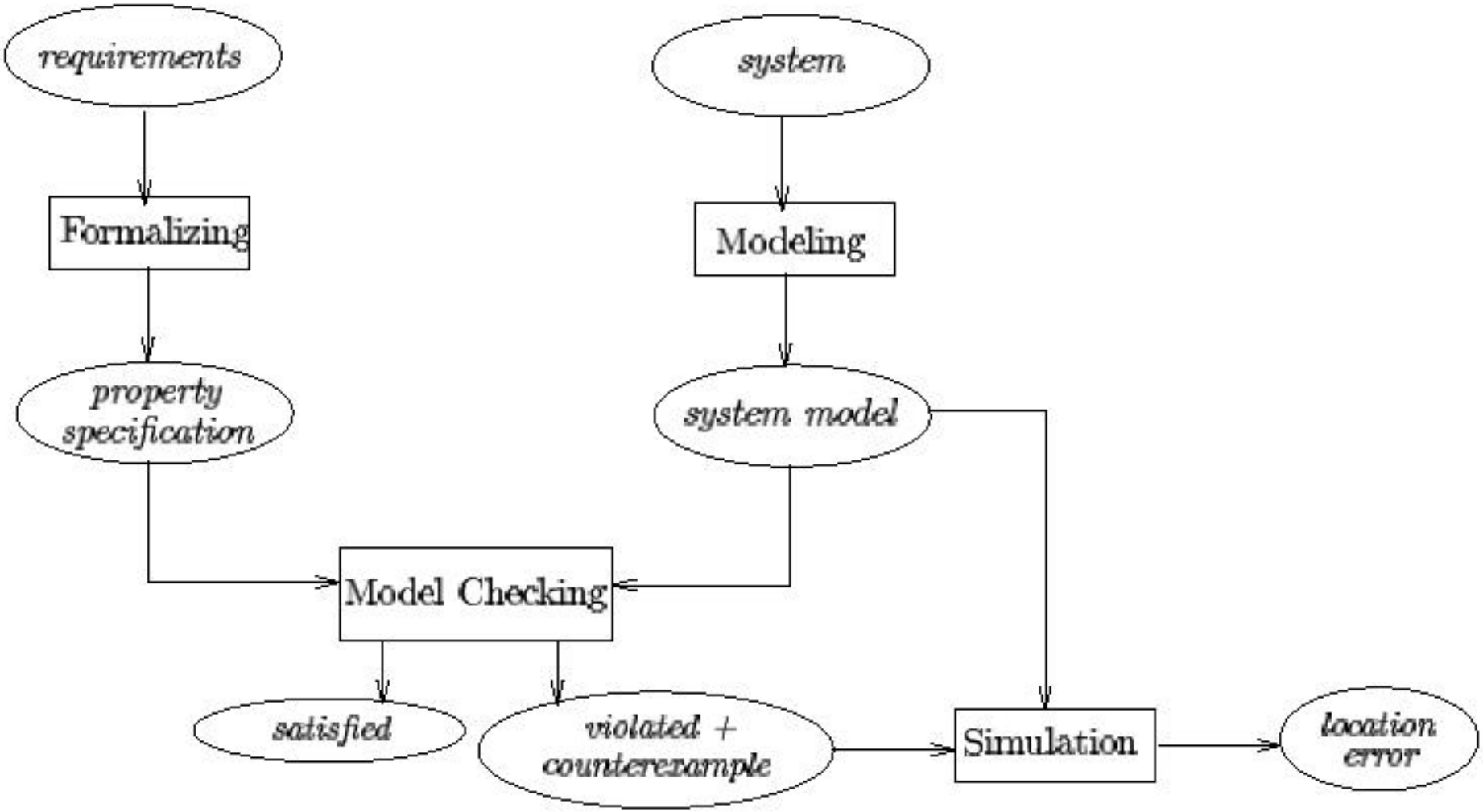
Model checking

- Usa ***sistemi a transizioni*** (***LTS***) per modellare i sistemi.
- Usa ***logica temporale*** per specificare le proprietà.
- Il problema della verifica viene spesso ridotto a problemi di ricerca su grafi.
- Può essere completamente automatizzato.
- Se la proprietà non è verificata viene generato un controesempio.
- È relativamente facile da usare.

Filosofia di progettazione

- Rendere la verifica una parte significativa del processo di sviluppo.
- Progettazione, verifica, riprogettazione.
- Model checking (come il testing) funziona al meglio quando ci sono “bachi”!
- Le verifiche di basso livello vengono sempre svolte tramite testing …
- … ma forse meccanismi di sintesi automatica potrbbero essere di aiuto …

Il processo di Model Checking



Il processo di Model Checking

Modellazione:

- descrivere il comportamento del sistema usando il linguaggio di descrizione fornito dal Model Checker (***costruzione del modello***).
- (opzionale) eseguire alcune simulazioni per controllare il comportamento del modello
- ***formalizzare la proprietà*** da verificare usando il linguaggio di specifica fornito dal Model Checker

Il processo di Model Checking

Esecuzione:

– esegue la verifica sul modello e sulla proprietà

Analisi:

– ***proprietà soddisfatta*** --- passare alla prossima

– ***proprietà violata*** ---

- analizzare il contro-esempio fornito tramite simulazione
- raffinare il modello, il progetto o la proprietà
- ripetere il procedimento

– ***out of memory*** --- tentare di ***ridurre il modello*** e riprovare

Svantaggi

- Funziona (bene) prevalentemente per sistemi a stati finiti.
- L'esplosione dello spazio degli stati è un problema serio:
  - Al crescere del numero di componenti interagenti le dimensioni del sistema crescono ***esponenzialmente***.
  - Trovare il giusto livello di astrazione per la modellizzazione non è facile.
    - Eliminare i giusti dettagli.
- Gli informatici non sono sempre a loro agio con le logiche temporali.

Stato dell'arte

- Di grande successo nella verifica di :
  - hardware "isolato" di media dimensione.
  - Protocolli di comunicazione/sicurezza
- Sempre più popolare in ambito industriale
- Sono disponibili tool robusti come **SPIN**, **SMV** (**NuSMV**), **COSPAN**, etc.
- I principi sottostanti sono ben compresi.

Tendenze della ricerca

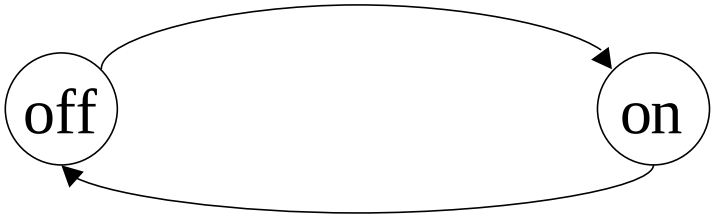
- Integrazione con il theorem-proving (**PVS**, ***Abstraction-Refinement*** per verifica SW).
- Verifica parametrica.
  - Verificare sistemi ad **n**-componenti per ogni **n**.
  - Pipelines con **n** stadi.
  - Protocolli di cache con **n** unità di memoria.
  - …..
- Sistemi a stati infiniti (ricorsione, parallelismo)
- Sistemi temporizzati (real-time).
- Sistemi ibridi.

Vedremo come...

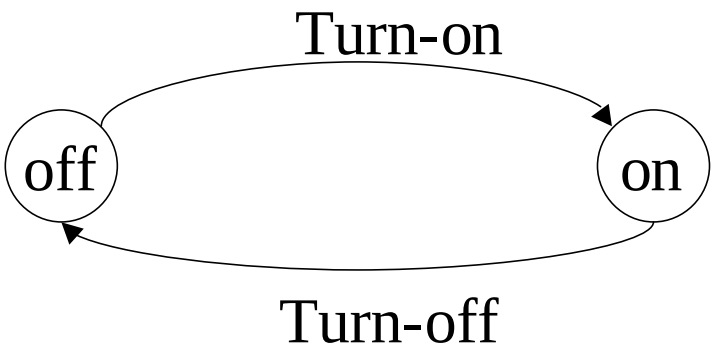
- Definire il modello di un sistema usando il formalismo dei ***sistemi di transizioni*** : **TS**
- Esprimere prorpietà interessanti del sistema usando una ***logica temporale*** : Y
- ***Verificare*** che il sistema soddisfa la proprietà desiderata tramite ***model checking***:
- **TS** $\models$ Y

Sistemi a transizioni

- Come modellare sistemi ?
- Uso di sistemi a transizioni.



Sistemi a transizioni



Sistemi a transizioni

- **TS = (S, S_{in} , $\longrightarrow$)**

  S un insieme di stati

  S_{in} un inseme di stati iniziali (un sottoinsieme di **S**)

  $\longrightarrow$ Í **S** ´ **S** la relazione di transizione

  (s, s') Î $\longrightarrow$ **s $\longrightarrow$ s'**

Varianti

- **S** deve essere finito.
- Ci deve essere ***un solo stato iniziale***.
- **Act** – un insieme di azioni **a**, **b**,…

$\rightarrow$ Í **S ´ Act ´ S** $\quad s \xrightarrow{a} s'$

$(s, a, s') \hat{I} \rightarrow$

Sistemi a transizioni

- Modello molto generale.
- Programmi, hardware, protocolli, Macchine di Turing …
- Programmi possono essere modellati tramite:
  - Stato
    - (locazione del controllo, valori correnti delle variabili)
  - Le transizioni producono nuovi stati :
    - (nuove locazioni del controllo, nuovi valori delle variabili)

Sistemi a transizioni

- Studieremo :
  - Il modello base
  - Nozioni associate
  - Esempi tramite circuiti, programmi e in particulare, alcuni sistemi concorrenti.

Logiche temporali

- Nascono nella filosofia greca antica.
- Sviluppate dai filosofi per lo studio dei costrutti linguistici che coinvolgono il flusso del tempo.
- Riprese nei tempi moderni da Prior.
- Importata in ambito informatico (per la verifica formale) nel 1977 da *Amir Pnueli*.

Logiche temporali

- Due principali famiglie:
- ***Logiche Temporali Lineari***

  (Linear-time Temporal Logics)
- ***Logiche Temporali Ramificate***

  (Branching-time Temporal Logics)
- Studieremo:
  - **LTL** (logica lineare)
  - **CTL**, **CTL***, **m-Calculus** (logiche ramificate)

Il framework di base

- **TS = (S, s_0, $\rightarrow$)**
- **$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$ ……**
- Una ***computazione*** come quella sopra è l'oggetto di base del ragionamento.
- In ***linear time*** ci interessano le ***computazioni individuali*** che hanno origine da **s_0**.
- In ***branching time*** ci interessa l'***albero delle computazioni*** che hanno origine da **s_0**.

Linear time

(***Lungo la computazione che stiamo analizzando***)

- ***Esempi***
  - Nello ***stato corrente***, "**buffer-size = 5**" è vero.
  - Nello ***stato suvvessivo (next)***

    "**lo stato del controllo è "End"**"
  - "**1-status = wait**" sarà vero ***finchè (until)***

    "**1-granted-access**" diviene ***infine (eventually)*** vera.

Branching time

- **CTL** (**C**omputation **T**ree **L**ogic)
- Esempiç
  - ***In ogni stato lungo ogni computazione***

    “**1-granted-access** E **2-granted-access**” è ***falsa***.
  - ***Per ogni computazione***, ***esiste uno stato lungo tale computazione*** nel quale la proprietà

    “**1-idle** e **2-idle**” è ***vera***.

Model Checking

- Partiamo da **TS = (S, s_0, $\longrightarrow$)**, il modello del sistema
- Da un insieme finito di proposizioni atimoche

  P = {p_1, p_2,…p_n}.
- **p_1** = “lo stato del controllo è “**begin**””
- Fissiamo una valutazione **V** che assegna un sottoinsieme di **P** ad ogni stato **s**;

  V(s_0) = {p_1, p_2, p_5} …

Model Checking

- **(TS, V)** --- Kripke Structure
- Costruiamo una formula in logica temporale Y (***specifica***, ***specification***) a partire da

  P = {p_1, p_2, ..p_n}
- Determiniamo se **(TS, V)** ***soddisfa*** (***meets***) la specifica Y.
- **(TS, V)** $\models$ Y

Tecniche basate sulla teoria degli automi

- **L**inear time **T**emporal **L**ogic (**LTL**).
- Si associa un automa a **K** e uno alla ***spec***.
- **K = (TS, V)** --------- A_K
- Y ------ A_Y
- **"K *meets the specification* Y"** iff
  $L(A_K) \subseteq L(A_Y)$.
- Questo di riduce ad un problema algoritmico su grafi!

Tecnica di etichettatura degli stati

- ***Branching time***
- **K = (TS, V)** Y
- Partiamo da **P = {p_1, p_2,..p_n}** ed etichettiamo induttivamente gli stati di **TS** con le sottoformule di Y.
- Il passo base è fornito dalla valutazione **V**!
- **"K meets** Y**"** iff, alla fine, s_0 è risulta essere etichettato con Y.

Esplosione dello spazio degli stati

- Sistemi concorrenti.
- **$TS = TS_1 \| TS_2 \| \ldots \| TS_m$**
- **m** processi sequenziali comunicanti (communicating sequential processes).
- Lo spazio degli stati di **TS** è esponenziale (**2^m**).
- **m** = 50 ……!

Esplosione dello spazio degli stati

- Trovare modi succinti per descrivere gli stati e le transizioni di **TS**.
- Usare questa rappresentazione nella ***computazione di punto fisso*** guidata dalla procedura di model checking.
- Utilizzeremo gli **OBDDs** (**O**rdered **B**oolean **D**ecision **D**iagrams); Randy Bryant, Ken McMillan…

Altre tecniche di riduzione

- ***Partial order reductions***
- ***Simmetria***
- ***Astrazione***
- ….

Sistemi real-time

- Le logiche temporali permettono di esprimere solo asserzioni ***qualitative*** sul tempo.
- A volte è necessario fare asserzioni ***quantitative*** sul tempo.
- ***Dopo al più 5.5 unità di tempo*** dal verificarsi dell'evento

  "**gas-pressure-level = danger**"
- ***deve*** verificarsi l'evento

  "**safety-valve = open**"

Sistemi real-time

- Uso degli ***automi temporizzati*** (***timed automata***) e sistemi a transizioni temporizzati (***timed transition systems***)
  - Sistemi a transizioni + ***Variabili di clock***.
- Una variabile di clock evolve col tempo a ***tasso unitario***.
- Una transizione può essere ***guarded*** (condizionata) da ***predicati*** che coinvolgono variabili di clock:

  $x < 6.1$ **e** $y > 3$
- Una variabile di clock può essere ***resettata a 0*** da una transizione.
- Molto espressivi.
- Proprietà espresse ***Logiche Temporali Temporizzate***.

Verifica formale

- Le logiche temporali sono uno strumento fondamentale in Informatica.
- Strumenti molto potenti e interessanti di specifica di proprietà di sistemi reatrivi.
- Verifica formale basata su model checking è un “*must*” per sistemi safety-critical.
- … ma c’è ancora molto da fare!