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 la specifica dei sistemi reattivi, sistemi software 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'opera!

Cosa riguarda il corso ?

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

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*** **(1994)** - 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 P5

- ***Il baco del Pentium*** (**1994**) - Il Professor Thomas Nicely del Lynchburg College in Virginia scoprì che il chip Pentium dava risposte scorrette per certe complesse equazioni.
- 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*** (**1990**) - 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 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***”.

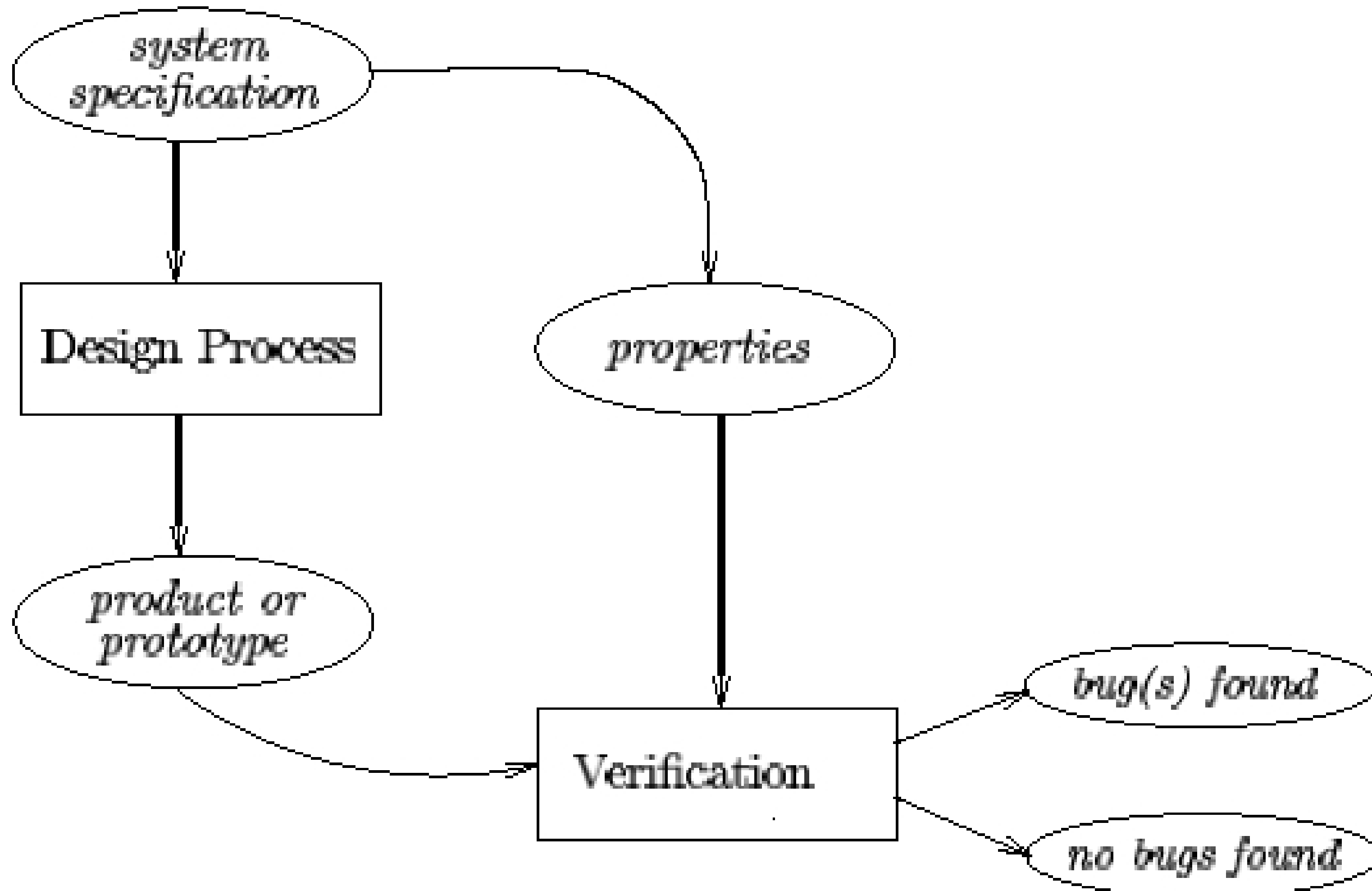
Necessità di correttezza

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

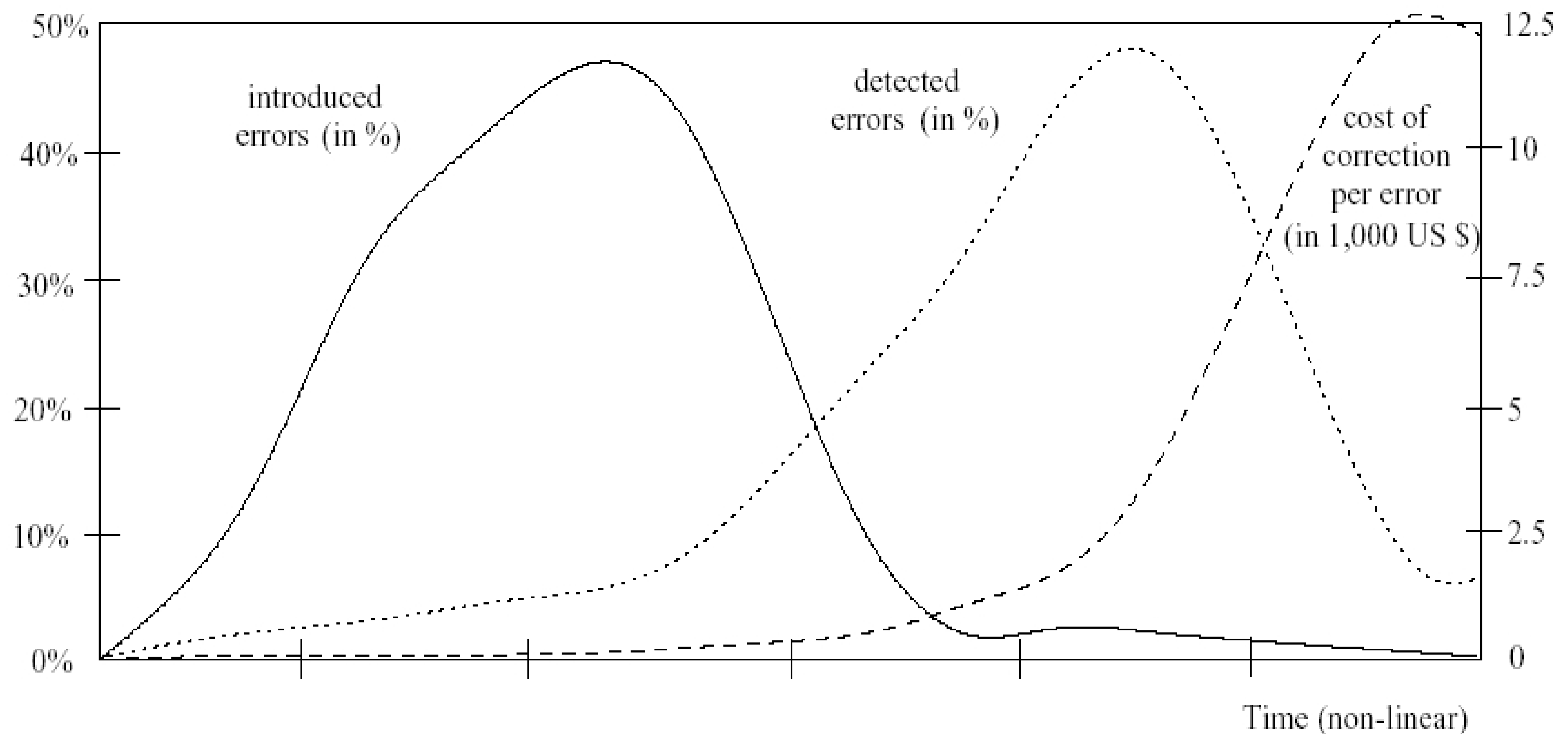
Società che usano i Metodi Formali

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

Verifica a posteriori



Errori: introduzione, individuazione e costi
(Liggesmeyer et al. 1998)



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 molto 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 errori ma non possono mai stabilire l'***assenza*** di errori (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 tool informatici per aiutare e spesso automatizzare questo ragionamento logico

Ragionamento logico e verifica

- Il **ragionamento logico** permette di dedurre proprietà dei sistemi dandoci un modo di "**verificare**" il nostro lavoro.
- Nella verifica formale le proprietà vengono controllate per "**tutti**" i possibili comportamenti del sistema. L'analisi è ***esaustiva***, anche se 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:

$$\mathcal{M} \models \phi$$

- $\mathcal{M}$ è un modello che descrive i possibili comportamenti del sistema
- ϕ è la proprietà o specifica che il sistema deve soddisfare
- $\models$ è una relazione che deve sussistere tra $\mathcal{M}$ e ϕ.
- 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 ***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 inferenza per modellare il sistema (***sistema formale***).
  - La proprieta da verificare viene espressa da un ***teorema*** del sistema formale.
  - Si dimostra il ***teorema*** con l'aiuto di un ***dimostratore automatico*** (***Theorem-Prover***)

Svantaggi

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

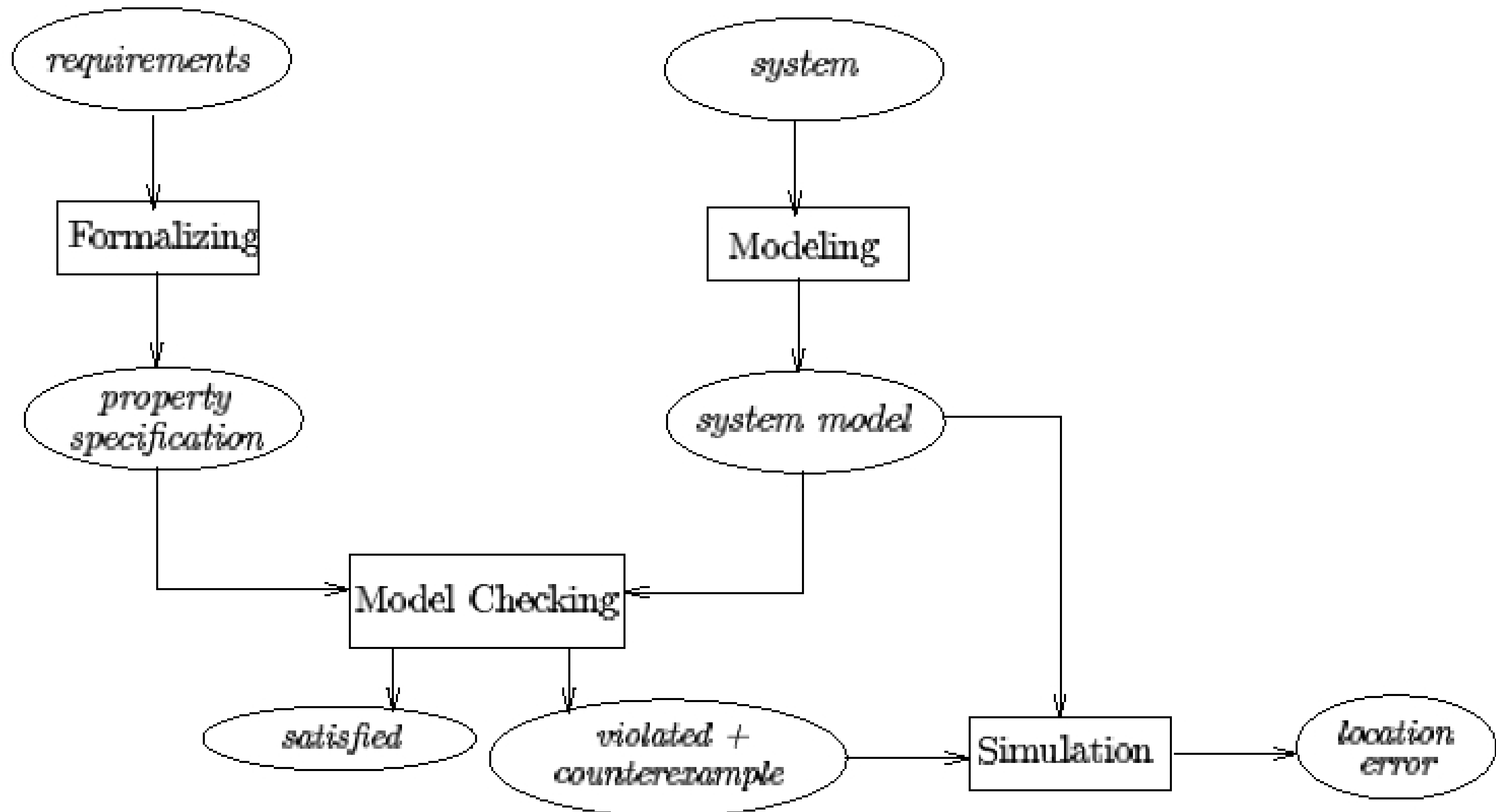
Model checking

- Usa ***sistemi a transizioni*** (***LTS***) o ***automi*** 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 componente significativa del processo di sviluppo.
- Progettazione, verifica, riprogettazione.
- Model checking (come il testing) funziona al meglio quando ci sono "errori"!
- Le verifiche di basso livello vengono sempre svolte tramite testing …
- … ma forse meccanismi di sintesi automatica potrebbero 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 modellazione non è facile.
    - Eliminare i giusti dettagli.
- Non tutti gli informatici sono 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

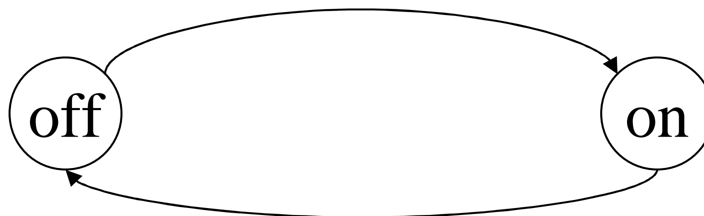
- Integrazione con il theorem-proving (**PVS**, ***Abstraction-Refinement*** per verifica SW).
- Verifica parametrica.
  - Verificare sistemi ad **n**-componenti per ogni **n**.
  - Pipelines con **n** stadi.
  - Protocolli di cache con **n** unità di memoria.
  - …..
- Sistemi a stati infiniti (ricorsione,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

Vedremo come...

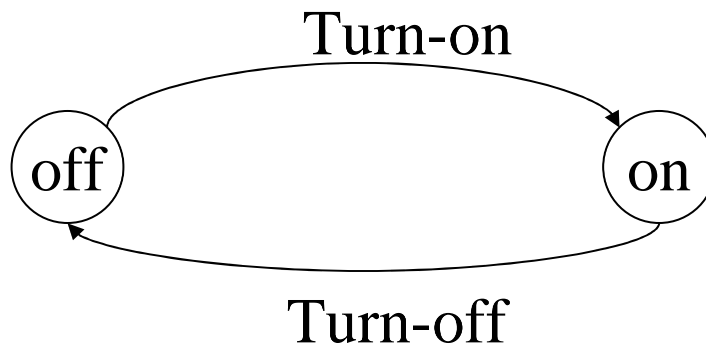
- Definire il modello di un sistema usando il formalismo dei ***sistemi di transizioni*** : **TS**
- Esprimere proprietà interessanti del sistema usando una ***logica temporale*** : Ψ
- ***Verificare*** che il sistema soddisfa la proprietà desiderata tramite ***model checking***:
- $\mathbf{TS} \models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

Sistemi a transizioni

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



Sistemi a transizioni



Sistemi a transizioni

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

  S un insieme di stati

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

  $\rightarrow \subseteq S \times S$ la relazione di transizione

  $(s, s') \in \rightarrow$ **$s \rightarrow s'$**

Varianti

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

$\longrightarrow \subseteq \mathbf{S} \times \mathbf{Act} \times \mathbf{S}$ $\quad \mathbf{s} \xrightarrow{\mathbf{a}} \mathbf{s'}$

$(\mathbf{s}, \mathbf{a}, \mathbf{s'}) \in \longrightarrow$

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 particolare, 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***, **μ-Calculus** (logiche ramificate)

Il framework di base

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

Linear time

(*Lungo la computazione che stiamo analizzando*)

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

    "**lo stato del controllo è "End"**"
  - "**P1-status = wait**" vero ***finchè (until)*** "**P1-granted-access**" diviene ***infine (eventually)*** vero.

Branching time

- **CTL** (**C**omputation **T**ree **L**ogic)
- Esempi:
  - ***In ogni stato lungo ogni computazione*** la proprietà "**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 atomiche

  $P = \{p_1, p_2, \ldots p_n\}$.
- **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 **Ψ** (***specifica***, ***specification***) a partire da

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

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)** --------- $A_{\mathbf{K}}$
- **Ψ** ------ A_{Ψ}
- **"K *meets the specification* Ψ"** iff

  $\mathbf{L}(A_{\mathbf{K}}) \subseteq \mathbf{L}(A_{\Psi})$.
- Questo si riduce ad un problema algoritmico su grafi!

Tecnica di etichettatura degli stati

- ***Branching time***
- **K = (TS, V)**   **Ψ**
- Partiamo da **$P = \{p_1, p_2,..p_n\}$** ed etichettiamo induttivamente gli stati di **TS** con le sottofor-mule di **Ψ** che essi soddisfano.
- Il passo base è fornito dalla valutazione **V**!
- **"K meets Ψ"** iff, alla fine, **s_0** è risulta essere etichettato con la proprietà **Ψ**.

Esplosione dello spazio degli stati

- Sistemi concorrenti.
- $\mathbf{TS = TS_1 \parallel TS_2 \parallel \ldots \parallel TS_m}$
- **m** processi sequenziali comunicanti (communicating sequential processes).
- Lo spazio degli stati di **TS** è esponenziale rispetto al numero di sistemi (approx. $|TS_i|^m$).
- **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 **OBDD*s*** (**O**rdered **B**oolean **D**ecision **D**iagrams); Randy Bryant, Ken McMillan…

Altre tecniche di riduzione

- ***Partial order reduction***
- ***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 in ***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!

Logic

A formal logic is defined by its *syntax* and *semantics*.

Syntax

- An *alphabet* is a set of symbols.
- A finite sequence of these symbols is called an *expression*.
- A set of rules defines the *well-formed* expressions (*well-formed femulae* or *wff*'s).

Semantics

- Gives meaning to well-formed expressions
- Formal notions of induction and recursion are required to provide a rigorous semantics.

Propositional (Boolean) Logic

Propositional logic is simple but extremely important in Computer Science

1. It is the basis for day-to-day reasoning (e.g., in programming)
2. It is the theory behind digital circuits.
3. Many problems can be translated into propositional logic.
4. It is an important part of more complex logics, such as: *First-Order Logic (*also called *Predicate Logic)*, *Modal and Temporal logic*, which we will discuss later.

Propositional Logic: Syntax

Alphabet

(	Left parenthesis	Begin group
)	Right parenthesis	End group
$\neg$	Negation symbol	English: not
$\wedge$	Conjunction symbol	English: and
$\vee$	Disjunction symbol	English: or (inclusive)
$\rightarrow$	Conditional symbol	English: if, then
$\leftrightarrow$	Bi-conditional symbol	English: if and only if
$\mathbf{A_1}$	First propositional symbol	
$\mathbf{A_2}$	Second propositional symbol	
…		
$\mathbf{A_N}$	N-th propositional symbol	
…		

We are assuming a *countable* alphabet, but most of our conclusions hold equally well for an *uncountable* alphabet.

Propositional Logic: Syntax

Alphabet

Propositional connective symbols: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$

Logical symbols: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, (,).

Parameters or *nonlogical symbols*: $\mathbf{A_1}$, $\mathbf{A_2}$, $\mathbf{A_3}$, . . .

The *meaning* of *logical symbols* is always the same.

The *meaning* of *nonlogical symbols* depends on the context.

From now on, let **AP** be the set $\{\mathbf{A_1}, \mathbf{A_2}, \mathbf{A_3}, \ldots\}$, called the set of *atomic propositions*.

Propositional Logic: Syntax

A *propositional expression* is a sequence of symbols. A sequence is denoted explicitly by a comma separated list enclosed in angle brackets:

Examples

(, A_1 , ∧ , A_3 ,)	**(A_1 ∧ A_3)**
(, (, ¬ , A_1,) , → , A_2 ,)	**((¬A_1) → A_2)**
) ,) , ↔ , A_1, ¬ , A_5	**)) ↔ A_1¬ A_5**

We will write these sequences as simple strings of symbols, with the understanding that the ***formal structure*** represented is a sequence containing exactly the symbols in the string.

The formal meaning becomes important when trying to prove things about expressions.

We want to restrict the kinds of expressions that will be allowed.

Propositional Logic: Syntax

Let us define the set **W** of *well-formed formulas* (*wff*'s).

(a) Every expression consisting of a single propositional symbol is in **W** (**AP** $\subseteq$ **W**);

(b) If α and β are in **W**, then so are $(\neg \alpha)$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \rightarrow \beta)$ and $(\alpha \leftrightarrow \beta)$;

(c) No other expression is in **W**.

This definition is *inductive*: the set being defined is used as part of the definition.

How would you use this definition to prove that the expression $))\leftrightarrow A_1 \neg A_5$ is not a *wff*?

Propositional Logic: Semantics

Intuitively, given a *wff* α and the truth value (either *true* or *false*) for each propositional symbol in α (the atomic propositions), we should be able to determine the truth value of α.

How do we make this precise?

Let υ be a function from **AP** to {0,1}, where **0** represents *false* and **1** represents *true* . Recall that in the inductive definition of *wff* 's, **AP** contains the propositional symbols.

Any function υ defined as above is called ***truth assignment***, and represent a possible ***propositional model***.

Now, we define the ***satisfaction relation*** $\models$ between υ and elements of W.

Propositional Logic: Semantics

Let υ be a function from **AP** to {0,1}, where **0** represents *false* and **1** represents *true* .

The ***satisfaction relation*** $\models$ between υ and elements of W is defined inductively as follows:

- $\upsilon \models \mathbf{A_i}$ if and only if $\upsilon\mathbf{(A_i) = 1}$
- $\upsilon \models (\neg\, \alpha)$ if and only if $\upsilon \not\models \alpha$
- $\upsilon \models (\alpha \wedge \beta)$ if and only if $\upsilon \models \alpha$ and $\upsilon \models \beta$
- $\upsilon \models (\alpha \vee \beta)$ if and only if $\upsilon \models \alpha$ or $\upsilon \models \beta$
- $\upsilon \models (\alpha \rightarrow \beta)$ if and only if $\upsilon \not\models \alpha$ or $\upsilon \models \beta$
- $\upsilon \models (\alpha \leftrightarrow \beta)$ if and only if $\upsilon \models \alpha$ if and only if $\upsilon \models \beta$

Truth Tables

There are other ways to present the semantics which are less formal but perhaps more intuitive.

α	$\neg\alpha$
1	0
0	1

α	β	$\alpha \wedge \beta$
1	1	1
1	0	0
0	1	0
0	0	0

α	β	$\alpha \vee \beta$
1	1	1
1	0	1
0	1	1
0	0	0

α	β	$\alpha \rightarrow \beta$
1	1	1
1	0	0
0	1	1
0	0	1

α	β	$\alpha \leftrightarrow \beta$
1	1	1
1	0	0
0	1	0
0	0	1

Truth Tables: Examples

Truth tables can be used to calculate all possible truth values for a given *wff* with respect to any possible assignment υ

There is a row for each possible truth assignment υ to the propositional atoms and connectives.

A_1	A_2	A_3	$(A_1$	$\vee$	$(A_2$	$\wedge$	$\neg A_3))$
1	1	1	1	1	1	0	0
1	1	0	1	1	1	1	1
1	0	1	1	1	0	0	0
1	0	0	1	1	0	0	1
0	1	1	0	0	1	0	0
0	1	0	0	1	1	1	1
0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1

Satisfiability and Validity

A *wff* α is *satisfiable* if there exists some truth assignment υ which satisfies α.

Suppose Σ is a set of *wff*'s. Then Σ *tautologically implies* α, or $\Sigma \models \alpha$, if ***every truth assignment*** which satisfies each formula in Σ also satisfies α.

Particular cases:

- If $\emptyset \models \alpha$, then we say is a *tautology* or is *valid* and we write $\models \alpha$
- If Σ is *unsatisfiable*, then $\Sigma \models \alpha$ for every *wff* α
- If $\alpha \models \beta$ (shorthand for $\{\alpha\} \models \beta$) and $\beta \models \alpha$, then α and β are *tautologically equivalent*.
- $\Sigma \models \alpha$ if and only if the *wff* $\wedge \Sigma \rightarrow \alpha$ is ***valid*** ($\models \wedge \Sigma \rightarrow \alpha$).

Some Tautologies

Associative and Commutative laws for $\wedge, \vee, \leftrightarrow$

Distributive Laws

- $(A \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$.
- $(A \vee (B \wedge C)) \leftrightarrow ((A \vee B) \wedge (A \vee C))$.

Negation

- $\neg\neg A \leftrightarrow A$
- $\neg(A \rightarrow B) \leftrightarrow (A \wedge \neg B)$
- $\neg(A \leftrightarrow B) \leftrightarrow ((A \wedge \neg B) \vee (\neg A \wedge B))$

De Morgan's Laws

- $\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$
- $\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$

More Tautologies

Implication

- $(A \rightarrow B) \leftrightarrow (\neg A \vee B)$

Excluded Middle

- $A \vee \neg A$

Contradiction

- $\neg(A \wedge \neg A)$

Contraposition

- $(A \rightarrow B) \leftrightarrow (\neg B \rightarrow \neg A)$

Exportation

- $((A \wedge B) \rightarrow C) \leftrightarrow (A \rightarrow (B \rightarrow C))$

Examples

- $(A \vee B) \wedge (\neg A \vee \neg B)$ is satisfiable, but not valid.
- $(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A \leftrightarrow B)$ is unsatisfiable.
- $\{A, A \rightarrow B\} \models B$ $\quad (A \wedge (A \rightarrow B) \wedge (\neg B))$
- $\{A, \neg A\} \models (A \wedge \neg A)$ $\quad (A \wedge (\neg A) \wedge \neg(A \wedge \neg A))$
- $\neg(A \wedge B)$ is tautologically equivalent to $\neg A \vee \neg B$

$\neg(\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B))$

Suppose you have an algorithm *SAT* which would take a *wff* α as input and return *true* if α is satisfiable and *false* otherwise.

How would you use this algorithm to verify each of the claims made above?

Examples

- $(A \vee B) \wedge (\neg A \vee \neg B)$ is satisfiable, but not valid.
- $(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A \leftrightarrow B)$ is unsatisfiable.
- $\{A, A \rightarrow B\} \models B$ $\quad (A \wedge (A \rightarrow B) \wedge (\neg B))$
- $\{A, \neg A\} \models (A \wedge \neg A)$ $\quad (A \wedge (\neg A) \wedge \neg(A \wedge \neg A))$
- $\neg(A \wedge B)$ is tautologically equivalent to $\neg A \vee \neg B$

  $\neg(\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B))$

Now suppose you had an algorithm *CHECKVALID* which returns *true* when α is valid and *false* otherwise.

How would you verify the claims given this algorithm?

Satisfiability and ***validity*** are dual notions: α ***is unsatisfiable*** if and only if $\neg\alpha$ ***is valid***.

Satisfiability with Truth Tables

An Algorithm for Satisfiability

To check whether α is satisfiable, form the truth table for α. If there is a row in which **1** appears as the value for α, then α is satisfiable. Otherwise, α is unsatisfiable.

Notice that this algorithm has ***exponential complexity***, since the number of different rows in a truth table is exponential (**2^n**) in the number **n** of atomic propositions occurring in α.

An Algorithm for Tautological Implication

To check whether $\{\alpha_1,\ldots,\alpha_k\} \models \beta$, check the satisfiability of the ***wff*** $(\alpha_1 \wedge \ldots \wedge \alpha_k) \wedge (\neg \beta)$. If it is unsatisfiable, then $\{\alpha_1,\ldots,\alpha_k\} \models \beta$, otherwise $\{\alpha_1,\ldots,\alpha_k\} \not\models \beta$.

Notice also that the computational ***complexity*** of the ***propositional satisfiability*** is **NP-Complete**!

Boolean Functions

- **f : Domain $\rightarrow$ Range**
- Boolean function:
  - **Domain = {0, 1}^{n} = {0,1} $\times$ …. $\times$ {0,1}.**
  - **Range = {0, 1}**
  - **f** is a function of **n** boolean variables.
- How many boolean functions of 3 variables are there?

Boolean Functions

- **f : Domain $\rightarrow$ Range**
- Boolean function:
  - **Domain = $\{0, 1\}^n$ = $\{0,1\} \times \ldots\ldots \times \{0,1\}$.**
  - **Range = $\{0, 1\}$**
  - **f** is a function of **n** boolean variables.
- How many boolean functions of 3 variables are there?
  - Answer : **$2^{2^3} = 2^8$!**

There are **2^3** different input points and **2** possible output values for each input point. **2^{2^3}** is also the number of different ***n***-ary propositional connectives

Boolean Functions & Truth Tables

x	y	z	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$g : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$$

Boolean Expressions

- Given a set of ***Boolean variables x,y,…*** and the constants **1** (true) and **0** (false):

  $t ::= x \mid 0 \mid 1 \mid \neg t \mid t \wedge t \mid t \vee t \mid t \rightarrow t \mid t \leftrightarrow t$

- The semantics of ***Boolean Expressions*** is defined by means of ***truth tables*** as usual.
- Given an ordering of Boolean variables, ***Boolean expressions*** can be used to express ***Boolean functions***.

Boolean expressions

- Boolean functions can also be represented as boolean (propositional) expressions.
- **$x \wedge y$** represents the function:
  - **$f : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$**
    - **f(0, 0) =**
    - **f(0, 1) =**
    - **f(1, 0) =**
    - **f(1, 1) =**

Boolean expressions

- Boolean functions can also be represented as boolean (propositional) expressions.
- $x \wedge y$ represents the function:
  - $f : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$
    - $f(0, 0) = 0$
    - $f(0, 1) = 0$
    - $f(1, 0) = 0$
    - $f(1, 1) = 1$

Boolean functions and expressions

x	y	z	g
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$g : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$$

$$g = ((x \Leftrightarrow y) \wedge z) \vee ((x \Leftrightarrow \neg y) \wedge \neg z)$$

Boolean expressions and functions

x	y	z	g
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

$$g = (x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (\neg x \wedge y)$$

Boolean expressions and functions

x	y	z	g
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$g = (x \wedge y \wedge \neg z) \vee (x \wedge \neg y \wedge z) \vee (\neg x \wedge y)$$

$$g : \{0, 1\} \times \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$$

Three Representations

- ***Boolean functions***
- ***Truth tables***
- ***Propositional formulas***.
- Three ***equivalent*** representations.
- We will look at a a ***fourth one*** later in the course.

Boolean Functions and Connectives

For each ***n***, there are $\mathbf{2^{2^n}}$ different ***n***-place boolean functions.

There are $\mathbf{2^n}$ different input points and **2** possible output values for each input point. $\mathbf{2^{2^n}}$ is also the number of different ***n***-ary propositional connectives.

0-ary connectives

There are two 0-place Boolean functions: the constants 0 and 1. We can construct corresponding 0-ary connectives $\bot$ and $\top$ with the meaning that $\upsilon \not\models \bot$ and $\upsilon \models \top$ regardless of the truth assignment\

Unary connectives

There are four 1-place functions, but these include the two constant functions mentioned above and the identity function. Thus the only additional connective of interest is negation: $\neg$.

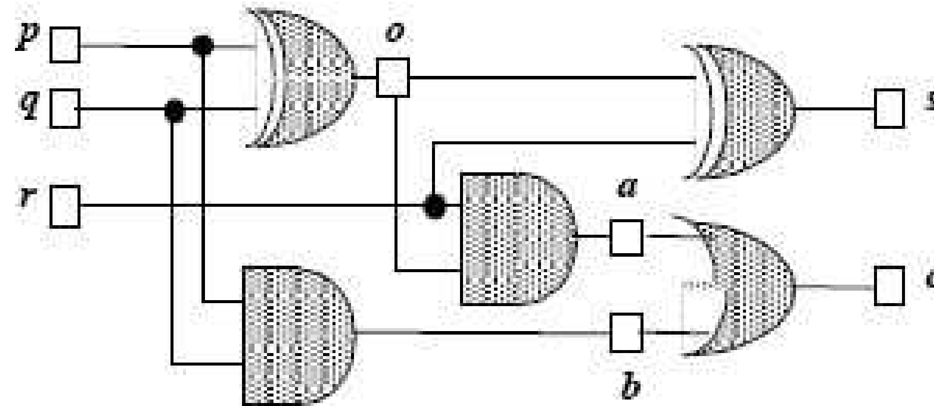
Binary connectives

There are sixteen 2-place Boolean functions. They are cataloged in the following table. Note that the first six correspond to 0-ary and unary connectives.

Binary Connectives

Symbol	Equivalent	Description
	$\bot$	constant **0**
	$\top$	constant **1**
	A	projection of first argument
	B	projection of second argument
	$\neg A$	negation of first argument
	$\neg B$	negation of second argument
$\wedge$	$A \wedge B$	and
$\vee$	$A \vee B$	or
$\rightarrow$	$A \rightarrow B$	conditional
$\leftrightarrow$	$A \leftrightarrow B$	bi-conditional
$\leftarrow$	$B \rightarrow A$	reverse conditional
$\oplus$	$(A \wedge \neg B) \vee (\neg A \wedge B)$	exclusive or
$\downarrow$	$\neg(A \vee B)$	nor (or Nicod stroke)
$\mid$	$\neg(A \wedge B)$	nand (or Sheffer stroke)
$<$	$\neg A \wedge B$	less than
$>$	$A \wedge \neg B$	greater than

Example: Curcuits and PL



$$o \Leftrightarrow (p \wedge \neg q) \vee (\neg p \wedge q)$$

$$a \Leftrightarrow r \wedge o$$

$$b \Leftrightarrow p \wedge q$$

$$s \Leftrightarrow (o \wedge \neg r) \vee (\neg o \wedge r)$$

$$c \Leftrightarrow a \vee b$$

Normal Forms: DNF

Normal forms in mathematics are canonical representations (i.e. all equivalent objects result in the same representation).

Definition: A formula α with $\mathbf{A_1, A_2, \ldots, A_n}$ propositional variables is in ***Disjunctive Normal Form*** (DNF) if it is has the structure:

$$(x^1_1 \wedge x^1_2 \wedge \ldots \wedge x^1_n) \vee \ldots \vee (x^m_1 \wedge x^m_2 \wedge \ldots \wedge x^m_n)$$

where $m \leq 2^n$ and for $i = 1,\ldots,n$ and $j = 1,\ldots,m$, $\mathbf{x^j_i}$ is either $\mathbf{A_i}$ or $\neg\mathbf{A_i}$ (both $\mathbf{A_i}$ and $\neg\mathbf{A_i}$ are called ***literals***).

E.g. $(\neg A_1 \wedge \neg A_2 \wedge A_3) \vee (A_1 \wedge \neg A_2 \wedge \neg A_3)$ is in DNF

$(\neg(A_1 \vee A_2) \wedge A_3)$ is not.

Each of the series of conjunctions picks out a row of the truth table where formula is true. DNF ORs together the ANDs for the true rows.

DNF

Consider the truth tables for the formulas **¬p ∧ ¬q ∧ r** and **¬p ∧ q ∧ ¬r**

	p	q	r	$\neg p \wedge \neg q \wedge r$	$\neg p \wedge q \wedge r$	$(p \wedge q \wedge \neg r)$
0	F	F	F	F	F	F
1	F	F	T	T	F	F
2	F	T	F	F	F	F
3	F	T	T	F	T	F
4	T	F	F	F	F	F
5	T	F	T	F	F	F
6	T	T	F	F	F	T
7	T	T	T	F	F	F

for **¬p ∧ ¬q ∧ r** only row 1 is true; for **¬p ∧ q ∧ r** only row 3 is true; for **p ∧ q ∧ ¬ r** only row 6 is true.

DNF

Consider the truth tables for the formulas **¬p ∧ ¬q ∧ r** and **¬p ∧ q ∧ ¬r**

	p	q	r	$\neg p \wedge \neg q \wedge r$	$\neg p \wedge q \wedge r$	$(p \wedge q \wedge \neg r)$
0	F	F	F	F	F	F
1	F	F	T	T	F	F
2	F	T	F	F	F	F
3	F	T	T	F	T	F
4	T	F	F	F	F	F
5	T	F	T	F	F	F
6	T	T	F	F	F	T
7	T	T	T	F	F	F

(**¬p ∧ ¬q ∧ r**) ∨ (**¬p ∧ q ∧ r**) ∨ (**p ∧ q ∧ ¬ r**) is true on rows 1, 3 and 6

DNF

Theorem: Every ***propositional formula*** that is not a contradiction is a ***logically equivalent*** to a DNF formula.

Corollary: For α, β not contradictions, $\alpha \leftrightarrow \beta$ if and only if α and β have the same DNF representation.

Proof: Two formulas are logically equivalent if and only if they have the same truth table (i.e. same true rows) and, thus, the same DNF.

DNF and Satisfiability

Theorem: Satisfiability of propositional ***formula*** in **DNF** can be checked on **Polynomial Time**.

Proof: Every formula in **DNF** is a disjunction of clauses. Therefore, the only possibility for the formula to be unsatisfiable is if every clause in isolation is unsatisfiable.

Since every clause is a conjunction of literals, for a clause of a DNF formula to be unsatisfiable, it must contain both a literal (**p**) and its complement ($\neg$**p**).

Therefore, every **DFN** ***formula*** is ***satisfiable*** unless every clause contains a pair of ***complementary*** literals.

And this can easily be checked in **Polynomial Time**.

CNF

Definition: A formula α with $A_1, A_2, \ldots, A_n$ propositional variables is in ***Conjunctive Normal Form*** (CNF) if it is has the structure:

$$(x^1_1 \vee x^1_2 \vee \ldots \vee x^1_n) \wedge \ldots \wedge (x^m_1 \vee x^m_2 \vee \ldots \vee x^m_n)$$

where $m \leq 2^n$, for $i = 1,\ldots,n$ and $j = 1,\ldots,m$, x^j_i is either A_i or $\neg A_i$.

E.g. $(\neg A_1 \vee \neg A_2 \vee A_3) \wedge (A_1 \vee \neg A_2 \vee \neg A_3)$ is in CNF

$(\neg(A_1 \vee A_2) \wedge A_3)$ is not.

Each of the series of disjunctions represents the negation of a row of the truth table where formula is false. CNF ANDs together the ORs corresponding to the negation of the flase rows.

One way to obtain the CNF form of a formula α is to write down the DNF for $\neg\alpha$, then negate it and apply De Morgan's lows as much as possilbe.

CNF and Validity

Using CNF to Check $\models \alpha$ (trivial)

$$\models (x^1_1 \vee x^1_2 \vee \ldots \vee x^1_n) \wedge \ldots \wedge (x^m_1 \vee x^m_2 \vee \ldots \vee x^m_n)$$

if and only if

$$\models (x^1_1 \vee x^1_2 \vee \ldots \vee x^1_n)$$
$$\models (x^2_1 \vee x^2_2 \vee \ldots \vee x^2_n)$$
$$\ldots$$
$$\models (x^m_1 \vee x^m_2 \vee \ldots \vee x^m_n)$$

If each x^j_i is a literal (e.g., p or its negation $\neg p$) then $\models (x^j_1 \vee x^j_2 \vee \ldots \vee x^j_n)$ iff there exists k and l s.t. $x^j_k = p$ and $x^j_l = \neg p$.

And this can easily be checked in **Polynomial Time**.

SAT complexity revisited

Question: So why are not ***validity*** and ***satisfiability*** polynomial problems?

Answer: **No!** Since converting a formula into an ***equivalent*** DNF or CNF can be exponential in the number of literals.

Example:

CNF: $(A_1 \vee B_1) \wedge (A_2 \vee B_2) \wedge \ldots \wedge (A_n \vee B_n)$

DNF: $(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \vee (A_1 \wedge A_2 \wedge \ldots \wedge B_n) \vee$

$\vee (A_1 \wedge A_2 \wedge \ldots B_{n-1} \wedge A_n) \vee (A_1 \wedge A_2 \wedge \ldots B_{n-1} \wedge B_n) \vee \ldots \vee$

$\vee (B_1 \wedge B_2 \wedge \ldots \wedge B_n)$

In worlds, when a CNF formula contains n clauses, the equivalent DNF formula may contain up to 2^n clauses, where each clause contains, for each i, either A_i or B_i.