Walter Ruggeri (wruggeri)

FORMA CUM LAUDE: INTRODUZIONE

28 March 2018

PRESENTAZIONE

Benvenuto, caro lettore, a questo primo articolo sui metodi formali!
Se sei qui, ovviamente non conosci i metodi formali e vuoi capire di che si tratta, o magari ne hai sentito parlare e vuoi saperne di più, o ancora li conosci già e sei qui per giudicarmi; quale che sia il caso, ritengo giusto iniziare presentandoti brevemente la serie di articoli cui questo da inizio, e fatto questo entreremo finalmente nel merito.

Perché questi articoli?

La domanda sorge spontanea: di tutti gli argomenti disponibili, perché proprio i metodi formali? Perché non trattare piuttosto la costruzione di un robot che va in giro a cantare ossessivamente "The day I tried to live" dei Soundgarden o analizzare le possibili tecniche realizzative di una spazzola per calvi a forma di sega circolare? Le motivazioni che mi spingono sono sostanzialmente tre:

- Ho notato, seguendo un corso che trattava le basi minime dei metodi formali, tanta gente fissare il vuoto attendendo l'illuminazione che non arrivava: i metodi formali, pur se trattati in modo più che elementare, queste persone proprio non riuscivano a capirli: non ne capivano l'intima natura, non ne capivano il senso, non ne capivano l'utilità.
- Moltissime università italiane trascurano vergognosamente i metodi formali e la verifica in generale.
- La didattica dei metodi formali è attualmente molto involuta: non c'è - per quanto ne so - un approccio condiviso, non ci sono testi introduttivi adeguati, le notazioni adottate dai vari ricercatori spesso differiscono enormemente, non è sempre facile capire quali ricerche si siano tradotte in sviluppi della materia e quali no, un novellino non è in grado di giudicare l'affidabilità dei vari autori (spiegherò meglio cosa intendo in una lezione successiva)...

Partendo da qui, mi son detto: perché non scrivere degli articoli che aiutino chi vuole iniziare a capirci qualcosa e chi ha già iniziato a capirci di più? Detto, fatto: ho aperto un sondaggio qui sul forum, l'idea è stata approvata (con alcuni interessanti suggerimenti) ed eccoci qui.

Di cosa parleremo?

I metodi formali sono un argomento vastissimo, che abbraccia un quantità enorme di regioni della matematica e ha applicazioni in praticamente qualsiasi ambito i cui attori siano modellabili. Di conseguenza, pretendere di trattare esaustivamente i metodi formali è assurdo, ed egualmente è assurdo pretendere di procedere a vista: occorre stabilire degli argomenti e un piano per trattarli, ed attenervisi rigidamente.
In questo ciclo di articoli, tratteremo i seguenti argomenti (si noti che quelle citate sono macro-aree, per cui se la leggibilità ne trarrà giovamento il contenuto di alcune di esse sarà suddiviso in più articoli):

1. Introduzione ai metodi formali
2. Algebra di commutazione
3. Logica proposizionale e del primo ordine
4. Logiche temporali
5. Equivalence checking
6. Model checking

Non tratteremo, invece (o almeno: non tratteremo in questo ciclo di articoli), tanti altri argomenti, come ad esempio:

1. Logiche superiori
2. Theorem proving
3. Language containment
4. Linguaggi di specifica/modellazione quali Z, VDM-SL, CSP…

La nostra discussione sarà orientata prevalentemente alle tecniche di verifica dell'hardware, ma senz'altro non rifuggiremo riferimenti alla verifica del software quando potremo inserirli senza appesantire la trattazione o quando risulteranno più adatti. Ogni articolo sarà accompagnato, dove possibile, da esempi ed esercizi che rendano chiaro quanto spiegato.

Un paio di note sulla bibliografia

Per quanto riguarda la bibliografia, l'obiettivo è di garantire la possibilità di comprendere e approfondire quanto spiegato, per cui saranno citate molte pubblicazioni: è sempre meglio ascoltare più voci, perché ciascuna enfatizza specifiche parti dell'argomento.
È importante tenere presente che la bibliografia non ha solo il ruolo di riportare le fonti, bensì costituisce parte integrante del relativo articolo, alla quale si collega soprattutto come lista di approfondimenti consigliati; per questo motivo, essa includerà anche testi non esplicitamente citati nel corso degli articoli o addirittura non utilizzati affatto per la loro preparazione. Cosa leggere di questi testi è una scelta individuale; il mio consiglio è di limitarsi inizialmente ai capitoli che trattano esattamente quanto discusso negli articoli, così da poter sentire più voci sugli stessi argomenti senza strafare, e solo dopo aver metabolizzato bene tutto procedere con l'approfondimento personale.
Molti dei libri presenti in bibliografia sono recensiti o lo saranno a breve nell'apposita sezione della biblioteca di EY.

IL CONTESTO

Il problema della verifica

Come ben evidenziato dall'informatico Les Hatton nel primo capitolo di [Hatton95], chiunque realizzi un progetto (Hatton si rivolge ai progettisti software, ma il discorso vale per tutti) ha non solo il dovere di curarne la qualità, ma anche e soprattutto il dovere di verificarne la correttezza: un progetto scorretto causa ingenti perdite di tempo e denaro (lo sanno bene i progettisti della Intel, che per un bug di certe procedure di basso livello ha dovuto ritirare milioni di processori e fornire soluzioni alternative ai clienti), e nei casi più gravi può persino mettere in pericolo delle persone (si pensi, ad esempio, a cosa succederebbe se il software che gestisce i dispositivi di sicurezza di un'autovettura smettesse di funzionare). Segue l'assoluta necessità di un metodo quanto più possibile esaustivo di verifica *a priori* (ovvero, precedente all'effettiva produzione del sistema progettato).
Banalmente, il progettista potrebbe voler far ricorso alla simulazione delle operazioni che il sistema progettato deve svolgere; si tratta di una soluzione apparentemente semplice, economica ed efficace, quindi non è raro che sia la preferita. Tuttavia, la simulazione risulta meno banale di quanto ci si aspetterebbe, per molti motivi:

- Non è quasi mai possibile esplorare l'intero spazio dei casi possibili: per fare un esempio, verificare un banale circuito digitale con un ingresso da 32 bit richiederebbe almeno di valutare tutte le possibili configurazioni di questi 32 bit, ovvero più di quattro miliardi di casi.
- Non è sempre facile identificare i *test cases* meritevoli di particolare attenzione.
- La simulazione richiede, in generale, un non indifferente sforzo per la modellazione (che comunque introduce delle approssimazioni) del sistema progettato.

Oltre alla simulazione, sono disponibili varie altre tecniche di verifica: nell'ambito dei sistemi digitali programmabili, ad esempio, esistono tecniche *correct-by-construction* volte alla sintesi di progetti necessariamente corretti. Si tratta tuttavia di tecniche ad oggi immature o applicabili solo a casi specifici. Esempi di tecniche di vario genere per la verifica dei sistemi software sono disponibili nel capitolo 5 del già citato [Hatton95].

Perché i metodi formali?

È nel panorama sopra delineato che si inseriscono i metodi formali: si tratta di metodi dalla forte base matematica che permettono, partendo da un modello (anche parziale) del sistema sotto esame e da un'opportuna formulazione delle proprietà desiderate, di ottenere una dimostrazione di correttezza, e cioè verificare inequivocabilmente se il sistema rispetta le proprietà.
Va notato subito, a scanso di equivoci, che i metodi formali non sono la soluzione a tutti i problemi:

la loro appartenenza al dominio della matematica li rende a volte difficili da usare, e inoltre non eliminano il problema, già evidenziato nel caso della simulazione, di dover lavorare su un modello del sistema. Sono problemi di non poco conto: se non è possibile elaborare un'opportuna formulazione delle proprietà desiderate, per esempio, diventa sostanzialmente impossibile verificarle pur disponendo di un ottimo modello del sistema, e simmetricamente anche verificare la proprietà più banale diventa difficile se il modello del sistema è troppo impreciso o addirittura scorretto. Tuttavia, i metodi formali mantengono una validità enorme, e sono infatti usati (insieme alla simulazione, rispetto alla quale non si pongono solo come alternativa, bensì anche come complemento) in molti ambiti e addirittura obbligatori per progetti safety-critical.
Una spinta consistente all'utilizzo dei metodi formali è stata data, recentemente, dallo sviluppo di nuove tecniche di verifica in grado di nascondere le complicazioni teoriche e permettere al progettista di lavorare su front-end amichevoli e accessibili: si tratta dell'equivalence checking e del model checking, ovvero le due tecniche di verifica formale di cui parleremo in questo ciclo di articoli.

ANDIAMO NEL MERITO

Varchiamo la soglia

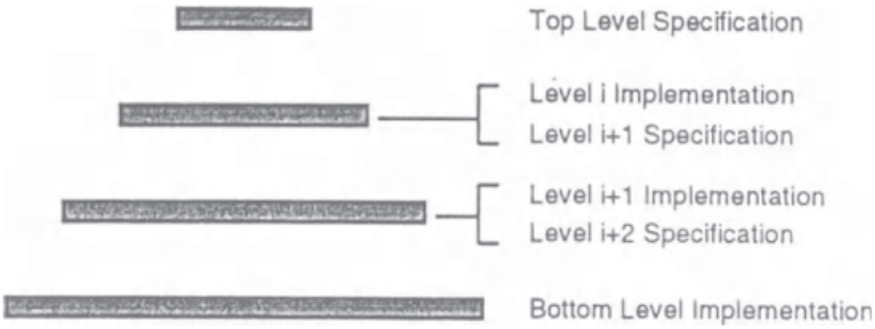
Delineato il contesto, possiamo finalmente iniziare a capire di cosa parliamo. L'introduzione ai metodi formali è un argomento spinoso, perché è grande il rischio di confondere il lettore e far germogliare nella sua mente convinzioni errate e dannose. Praticamente ogni autore che abbia scritto di metodi formali ha un suo metodo per introdurli; personalmente, intendo ispirarmi a Camurati e Prinetto [CamPri88], nonché a Kropf [Kropf99] e a Gupta [Gupta93], che peraltro si riferiscono direttamente alle applicazioni in ambito hardware.
Abbiamo già accennato all'utilità dei metodi formali: dati un modello e delle proprietà, essi servono a dimostrare la validità delle proprietà entro il modello. Ora saremo più precisi: i metodi formali, partendo da:

- Una **specifica** $\mathcal{S}$, che modella il comportamento desiderato del sistema.
- Un'**implementazione** $\mathcal{I}$, che modella la struttura del sistema.

Le quali sono espresse in opportuni linguaggi formali, si occupano di dimostrare opportune relazioni tra di esse, come ad esempio:

$$\mathcal{I} \rightarrow \mathcal{S}$$
$$\mathcal{I} = \mathcal{S}$$
$$\mathcal{I} \models \mathcal{S}$$

Particolarmente importante è la terza relazione, detta **model relation**: la sua interpretazione fa riferimento alla teoria dei linguaggi formali e dei modelli, ed è sostanzialmente "l'implementazione è un **modello semantico** entro cui la specifica è vera"; al momento, probabilmente, questa frase risulta un po' criptica, ma sarà più chiara quando - nelle prossime lezioni - vedremo le logiche formali. Per il momento, possiamo interpretare intuitivamente quella relazione come "l'implementazione modella un sistema che rispetta la specifica", così da rendere evidente che essa esprime proprio ciò che ci interessa dimostrare.
Come ottimamente evidenziato sia in [CamPri88] che in [Gupta93], un aspetto da non trascurare della formulazione che abbiamo appena visto per il problema della verifica è che si presta alla risoluzione gerarchica: partendo da una specifica di alto livello, che si pone corretta, la si può usare per verificare una certa implementazione; fatto questo, l'implementazione verificata diventa sua volta una specifica corretta, che può essere usata per verificare un'implementazione di più basso livello. Procedendo in questo modo, è possibile ottenere una verifica profonda del sistema, obiettivo altrimenti irraggiungibile o comunque molto difficile da perseguire.



Rappresentazione della verifica gerarchica.

Quali sono i metodi formali?

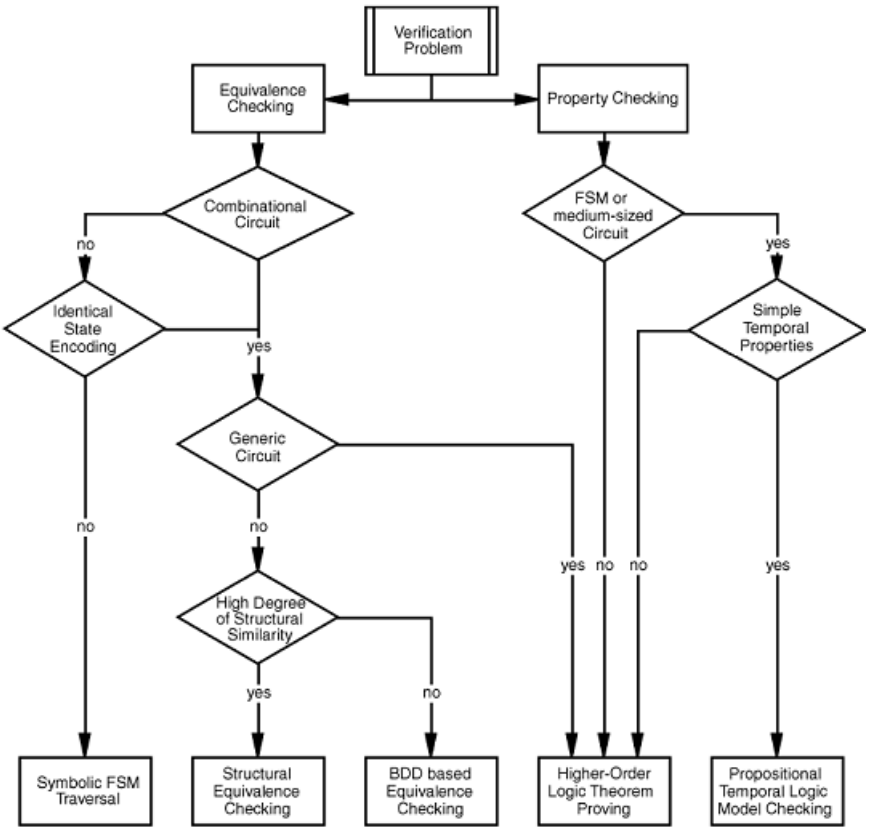
Ora che abbiamo definito formalmente (ahahahah) cosa sono i metodi formali, non ci resta che capire quali sono. Il panorama dei metodi formali è vastissimo, perché date la specifica e l'implementazione qualsiasi tecnica che permetta di verificare le relazioni tra di esse è teoricamente eleggibile a metodo formale (in realtà non è vero, e tra poco vedremo perché... ma facciamo un passo alla volta, e per il momento accontentiamoci di questa generalizzazione).
Alcuni esempi di metodi formali sono quelli che abbiamo già citato discutendo il programma di questo ciclo di articoli, e che ora definiamo brevemente:

- Il **theorem proving**, ovvero la dimostrazione "classica", come si farebbe per un teorema, di una qualsiasi relazione tra implementazione e specifica. Si tratta, ovviamente, di un metodo molto potente, ma l'elevata consapevolezza dei mezzi matematici richiesta lo rende poco utilizzabile dal progettista quadratico medio (il quale è già tanto se ha

davvero capito cosa sta progettando, soprattutto in certi ambiti legati allo sviluppo software...).

- Il **model checking**, ovvero la verifica, con metodi diversi dalla dimostrazione, della model relation. Si tratta di una tecnica relativamente nuova, ma molto potente e di ottime prospettive, oltre che totalmente (o quasi) automatizzabile.
- Il **language containment**, ovvero la verifica della relazione $\mathcal{L}(\mathcal{I}) \subseteq \mathcal{L}(\mathcal{S})$, dove $\mathcal{L}(x)$ è il linguaggio formale che descrive l'entità x.
- L'**equivalence checking**, ovvero la verifica, con metodi diversi dalla dimostrazione, della corrispondenza tra due modelli. Come il model checking (cui risulta intimamente legato, per motivi che vedremo a tempo debito), si tratta di una tecnica automatizzabile.

A questo punto, ha senso chiedersi: ma se esistono tante tecniche di verifica formale, come decidiamo quale usare? Cosa rende una tecnica di verifica "migliore" di un'altra?
Banalmente, non c'è una risposta univoca: stante il fatto che qualsiasi tecnica di verifica deve ovviamente rispettare la **proprietà di correttezza**, ovvero non deve permettere di dedurre la verità di un'ipotesi falsa, tutto dipende dalla specifica applicazione. In generale, sono metriche comuni la possibilità di automatizzare del tutto o in parte il processo di verifica (dettaglio che potrebbe far preferire, per esempio, il model checking all'assai più potente - ma non automatizzabile, se non entro certi limiti - theorem proving), la completezza (ovvero, la possibilità di dimostrare qualsiasi ipotesi vera; non è scontata come si potrebbe credere) e la facoltà di realizzare prove di un certo tipo (prove induttive, ad esempio, o magari prove composizionali).
In [Kropf99] viene proposto un utile diagramma che aiuta a capire come si potrebbe ragionare per identificare la tecnica più adatta per affrontare un problema di verifica; lo riportiamo qui:



Si tratta di uno schema ovviamente datato, che non può tener conto delle tantissime innovazioni che hanno caratterizzato i metodi formali negli ultimi vent'anni, tuttavia mantiene intatta la propria validità di fondo.

Fermiamoci un attimo sulle proprietà

La domanda è spontanea: ok, abbiamo il nostro metodo formale, abbiamo i linguaggi formali per descrivere la specifica e l'implementazione... ma cosa possiamo verificarci, con 'sta roba?
La risposta sembra ovvia: tutte quelle proprietà che possiamo esprimere nei linguaggi che abbiamo scelto e che dipendono solo da caratteristiche incluse nel modello del sistema. Questo punto, quantunque ovvio, merita un attimo di riflessione, perché ci permette di evidenziare l'importanza della consapevolezza dei propri mezzi.
Analizziamo un semplice esempio: se scegliamo di esprimere le nostre proprietà in logica proposizionale (sì, la logica matematica è un linguaggio formale... vedremo a tempo debito le sue caratteristiche), possiamo senz'altro dimostrare qualsiasi proprietà che riusciamo a formulare, perché la logica proposizionale è completa... tuttavia, se intendiamo esprimere proprietà del tipo "per ogni valore di una variabile, succede qualcosa" ci troviamo davanti a un bivio: o continuare a lavorare con la logica proposizionale, che ci permette di esprimere la proprietà solo nel caso in cui la variabile sia booleana, o passare alla logica predicativa. La risposta sembra ovvia: passiamo alla logica del primo ordine, ed eliminiamo il problema... tutto giusto, se non fosse che (come dimostrato da Alonzo Church, che ci ha pure formulato un teorema) non esiste un algoritmo che

possa dimostrare meccanicamente la validità di una formula predicativa!
A margine, è giusto notare che le proprietà comunemente richieste ad un sistema da verificare hanno delle denominazioni ben precise che è bene conoscere:

- Le proprietà del tipo "prima o poi il sistema deve fare qualcosa" sono dette **liveness property**; un esempio ovvio possiamo trovarlo guardando allo scheduler di un sistema operativo, al quale senz'altro richiederemmo di funzionare in modo tale che ogni processo sia posto in esecuzione in un tempo finito: questa è una liveness property!
- Le proprietà del tipo "una certa cosa non deve mai succedere" sono dette **safety properties**; esempi di questo tipo se ne trovano a volontà: basti pensare che, per esempio, si vuole che il sistema di controllo dei semafori ad un incrocio non attivi mai contemporaneamente la luce verde per gli autoveicoli e i pedoni sulla stessa carreggiata.

A queste proprietà si possono aggiungere di solito le **fairness properties**, che però non sono proprietà dimostrabili: si tratta più che altro di assunzioni del tipo "il sistema fa qualcosa infinitamente spesso", utili per circostanziare i casi in cui si vogliono valutare certe proprietà; accenneremo a queste cose a tempo debito.

Un esempio

Concludiamo quest'articolo introduttivo con un esempio banalissimo, ma molto carino e comunque utile per capire come si ragiona quando si tratta di metodi formali: abbiamo un sistema modellato in logica proposizionale:

$$\mathcal{I} = (\overline{x} + \overline{y} + z)(y + \overline{z})(x + \overline{z})$$

Vogliamo verificare che il suo comportamento rispetti queste specifiche:

$$\mathcal{S} = \begin{cases} \overline{z} \rightarrow \overline{x} + \overline{y} \\ \overline{x} \rightarrow \overline{z} \\ \overline{y} \rightarrow \overline{z} \end{cases} = (\overline{z} \rightarrow \overline{x} + \overline{y})(\overline{x} \rightarrow \overline{z})(\overline{y} \rightarrow \overline{z})$$

Ovvero, vogliamo verificare che $\mathcal{I} = \mathcal{S}$. Possiamo gestire la cosa in vari modi: per esempio, possiamo scegliere di vedere il tutto come un problema di equivalence checking, oppure possiamo decidere di dimostrare l'uguaglianza. Vediamo entrambi gli approcci:

- Come spiegheremo in una lezione futura, in casi come questo l'equivalence checking si può ridurre al semplice confronto delle tavole di verità:

x	y	z	$\mathcal{I}$	$\mathcal{S}$
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1

1 0 1 0 0

1 1 0 0 0

1 1 1 1 1

- Il theorem proving può essere svolto in molti modi, applicando le regole dell'algebra booleana che più ci piacciono. Per far prima, seguiremo il percorso più semplice: noteremo che $(a \to b) = \overline{a} + b$, ovvero:

$$\mathcal{S} = (\overline{z} \to \overline{x} + \overline{y})(\overline{x} \to \overline{z})(\overline{y} \to \overline{z}) = (\overline{x} + \overline{y} + z)(y + \overline{z})(x + \overline{z}) = \mathcal{I}$$

In entrambi i casi, il risultato è inequivocabile: la nostra implementazione è corretta!

BIBLIOGRAFIA

[BHJL96] P.E. Black, K.M. Hall, M.D. Jones, T.N. Larson, P.J. Windley, *A brief introduction to formal methods*, in *Proceedings of Custom Integrated Circuits Conference*, 1996, pp.377-380
[CamPri88] P. Camurati, P. Prinetto, *Formal verification of hardware correctness: introduction and survey of current research*, in *Computer*, 21(7), 1988, pp.8-19
[Gupta93] A. Gupta, *Formal hardware verification methods: a survey*, in R. Kurshan, *Computer-aided verification*, Springer, 1993, pp.5-92
[Hatton95] L. Hatton, *Safer C: Developing software for high-integrity and safety-critical systems*, McGraw-Hill, 1995
[Kropf99] T. Kropf, *Introduction to formal hardware verification*, Springer, 1999
[Monin03] J.F. Monin, *Understanding formal methods*, Springer, 2003
[ORegan17] G. O'Regan, *Concise guide to formal methods*, Springer, 2017
[Wing90] J.M. Wing, *A specifier's introduction to formal methods*, in *Computer*, 23(9), 1990, pp.8-24