Le Algebre Evolventi per la validazione di hardware

Giampaolo Bella ed Elvinia Riccobene

Dipartimento di Matematica
Viale A. Doria, 6
I-95125 CATANIA
e-mail: {giamp, riccobene}@dipmat.unict.it

Sommario La complessità dei moderni circuiti integrati cresce in continuazione. La necessità di commercializzare prodotti perfettamente funzionanti è ovvia, tuttavia ancora oggi non esiste un metodo sicuro, a costi ragionevoli, per convincersi categoricamente del corretto funzionamento di un dispositivo elettronico. Diversi sono stati i tentativi di superare i limiti della *Simulazione* con la *Verifica Formale*. L'articolo descrive rigorosamente quest'ultima metodologia, proponendo l'utilizzo del potente ed elegante formalismo *Algebre Evolventi*. Quindi presenta un primo originale risultato ottenuto, evidenziandone la superiore intuitività rispetto ad un analogo tentativo facente uso di un altro formalismo. Un accenno ai risultati più recenti conclude il lavoro.

KEYWORDS: affidabilità della comunicazione, validazione di hardware, verifica formale.

1 Introduzione

Il concetto di *circuito integrato* nasce solo agli inizi degli anni '60, tuttavia attualmente è già stato raggiunto un *livello d'integrazione* astronomico (qualche milione di *transistor* per *chip*). Questa straordinaria evoluzione è certamente frutto dei progressi nel campo dell'ingegneria dell'hardware, ma anche del miglioramento delle tecniche produttive dell'industria chimico-meccanica.

La complessità raggiunta crea pesanti perplessità sul corretto funzionamento di un moderno dispositivo elettronico, per cui nasce l'esigenza di testare i prodotti prima di commercializzarli, ossia di provarne in maniera rigorosa la *correttezza*. Questa prova in verità non è per niente banale, anzi attualmente non esiste una metodologia generale, sicura e a costi ragionevoli [Bel95].

Gli stimoli a continuare la ricerca in questa direzione sono molto forti: basti pensare alle grosse cifre perse dalla *Intel* col suo ultimo prodotto, il *pentium*, che ha rivelato degli errori solo dopo la commercializzazione.

L'articolo è organizzato come segue: la **sezione 2** affronta il concetto di validazione di dispositivi elettronici, presentando le metodologie attualmente conosciute – *simulazione* e *verifica formale* –, con un'accurata formalizzazione della seconda, l'unica che può portare ad una dimostrazione matematica della

correttezza. Essa contiene anche una breve descrizione del formalismo *Algebre Evolventi*, che viene poi utilizzato per la validazione di un modello semplificato di *shift register*, presentata nella **sezione 3**. La **sezione 4** conclude il lavoro con una panoramica delle successive applicazioni del metodo.

2 Validazione di dispositivi elettronici

Validazione dell'implementazione di un dispositivo elettronico, o piú semplicemante validazione di un dispositivo, è il processo che consiste nello stabilire che il dispositivo funziona correttamente – come il progettista desidera che funzioni – su ogni possibile input [Dav90].
Le due sottosezioni seguenti presentano rispettivamente le due metodologie oggi note per la realizzazione di tale processo.

2.1 Simulazione

Si tratta della tecnica più vecchia di validazione. Quando i dispositivi elettronici erano ancora a bassi livelli di integrazione (*SSI*), era possibile dimostrarne la correttezza semplicemente "testandoli", ossia facendo vedere che il dispositivo forniva gli output desiderati, su ogni possibile input.

Il metodo si rivelò in principio esaustivo, perchè non lasciava alcun dubbio sul corretto funzionamento, e soprattutto molto economico, in quanto non richiedeva nessun ulteriore investimento di capitali dopo la realizzazione del dispositivo.

Tuttavia, a mano a mano che il livello di integrazione è aumentato (*MSI*, *LSI*) grazie ai progressi nelle tecnologie di costruzione, la tecnica di simulazione ha cominciato a mostrare il suo grosso limite: la necessità di testare tutti i possibili input. Infatti, l'aumento del numero di porte logiche del dispositivo ha fatto crescere in maniera esponenziale il numero di input possibili, cosicchè, attualmente, simulare il funzionamento di un circuito *VLSI* sarebbe un procedimento lunghissimo (misurabile in anni-uomo!) e quindi incredibilmente costoso: di fatto, impossibile.

Nasce anche un altro problema: scoprire un errore dopo lungo tempo dall'inizio della simulazione significa aver speso ingenti capitali fino allo stato attuale, spenderne ancora per riproggettare i settori del dispositivo con l'errore, continuare a spendere in una nuova simulazione del progetto modificato. È chiaro che questa prospettiva, peraltro non improbabile, rappresenta una grossa incognita per qualsiasi azienda produttrice di hardware che, invece, vorrebbe assicurarsi investimenti sicuri.

2.2 Verifica formale

La *verifica formale* di un circuito – unica alternativa alla *simulazione* – è un procedimento di concezione recente, simile ad una dimostrazione matematica. Infatti, così come la veridicità di un teorema dimostrato matematicamente è indipendente dai particolari valori delle entità trattate, la correttezza di un dispositivo verificato formalmente è indipendente dai valori di input. In altre parole, il

prendere in considerazione tutti i casi possibili è implicito al concetto di verifica formale. Per questo motivo tale metodologia ha tutti i requisiti per soppiantare definitivamente la simulazione: sicurezza della correttezza a costi ragionevoli.

I passi di una verifica formale si possono enucleare [Gup92] come segue:

1. *scelta del formalismo*,
2. *specifica*,
3. *implementazione*,
4. *processo di validazione*.

Esaminiamoli nei dettagli.

2.2.1 Il formalismo. *Formalismo*, è uno strumento matematico capace di rappresentare correttamente il sistema da validare. Un formalismo deve essere anche *versatile*, cioè capace di modellare le situazioni più svariate, ed *espressivo*, ossia capace di rappresentare qualunque situazione ad un qualsiasi livello di astrazione. Attualmente i formalismi utilizzati sono molteplici ma possono essere raggruppati come segue:

- Formalismi basati sulla Logica.
  - Logica del primo ordine [Hat82, Men64]
  - Logica di Boyer-Moore [BM84, BM88]
  - Logica di ordine superiore [Rei83, Gor87]
  - Logica Temporale [Eme90, RU71]
  - Logica Temporale Lineare [MP82]
  - Logica Temporale ad Intervalli [HKP82]
  - Mu-Calcolo [Koz83]

  con molteplici varianti.
- Formalismi basati sulla Teoria degli Automi.
  - Macchine di Moore [HU79]
  - Macchine di Dill [Dil88]
- Formalismi ibridi.
  - Logica temporale lineare ed Automi a stati finiti [Wol85]
  - Logica temporale ed automi di Büchi [VW86, Buc60]

Va osservato che la validazione di hardware necessita di un formalismo capace di modellare il concetto di tempo in quanto i valori delle porte di un circuito sono soggetti ad evoluzione temporale. Anche per questa ragione i formalismi basati sulla logica mostrano una notevole pesantezza sintattica, mentre quelli basati sulla teoria degli automi sono troppo poco flessibili, sebbene più espressivi.

Noi abbiamo scelto un formalismo – le *Algebre Evolventi* [Gur93a, Gur93b] – classificabile come ibrido, che ha rivelato alcuni grossi pregi sconosciuti ad altri:

1. una grande espressività che ci ha permesso di descrivere funzionamenti complicati in maniera molto breve;
2. una straordinaria semplicità sintattica, sconosciuta ai formalismi di tipo logico, che ha reso molto intuitive tutte le nostre descrizioni;

3. la facilità nel rappresentare il concetto di tempo, del tutto implicito nel formalismo.

Tali requisiti rivestono grande importanza perchè, come sarà più chiaro in seguito, proprio dalla bontà del formalismo dipende la possibilità di estendere il processo di validazione a dispositivi sempre più complicati.

La nozione di *Algebra Evolvente* fu definita nel 1988 da Yuri Gurevich, nel tentativo di affinare la tesi di Turing in base a considerazioni teoriche di complessità computazionale. Formalmente, un'*algebra evolvente* è un coppia

$$\mathcal{A} = (S, P)$$

dove S è un'algebra statica (secondo la definizione della logica classica) detta *stato iniziale* e P è un programma, ossia un insieme finito di *regole di transizione* della forma

`if` *condition* `then` *updates* `endif`

in cui *condition* è una condizione booleana ed *updates* è una sequenza di aggiornamenti di funzioni del tipo $f(t_1, \ldots, t_n) := t$. Un'*esecuzione* di P su S è una sequenza, finita o infinita, di algebre (stati)

$$S_0 = S, S_1, S_2, \ldots, S_n, S_{n+1}, \ldots$$

tale che ogni S_{i+1} è ottenuto applicando P ad S_i, ossia eseguendo tutti gli aggiornamenti di funzioni indotti dalle regole la cui condizione booleana è vera nell'algebra S_i. In generale, il simbolo $[t]^i$ indica l'interpretazione del termine t nello stato S_i.

Il trascorrere di un intervallo di tempo viene formalizzato mediante la transizione da uno stato all'altro. (Per una formalizzazione esaustiva si veda [Gur93a]).

2.2.2 La specifica. *Specifica* di un dispositivo è una descrizione macroscopica del funzionamento del dispositivo. Essa non dà alcuna informazione sulla struttura interna dell'oggetto, il quale viene visto semplicemente come una specie di scatola nera capace di ricevere certi valori in input e di restituirne altri in output.

2.2.3 L'implementazione. *Implementazione* è una descrizione microscopica del dispositivo. Essa spiega dettagliatamente come il dispositivo va costruito nella pratica, ossia come va implementato. Le operazioni che nella specifica venivano viste come atomiche, adesso vengono descritte in tutti i loro singoli passi.

2.2.4 Il processo di validazione. Consiste nello stabilire che l'implementazione data "soddisfa" la specifica, ossia nel dimostrare l'esistenza di una certa relazione fra implementazione e specifica che ne esprima l'equivalenza. I metodi comunemente usati per ottenere tale equivalenza sono basati sui seguenti strumenti [Gup92]:

- *Funzioni.* L'equivalenza dei due livelli descrittivi è ricondotta all'equivalenza di funzioni (*Equivalence checking*).
- *Modelli.* La specifica è costituita da una formula logica di cui provare la veridicità rispetto al modello semantico fornito dall'implementazione (*Model checking*).
- *Teoremi.* La relazione fra specifica ed implementazione viene espressa attraverso un teorema che va provato mediante dimostrazione automatica (*Theorem proving*).
- *Linguaggi.* Viene dimostrato che il linguaggio che rappresenta l'implementazione è contenuto nel linguaggio che descrive la specifica (*Language containment*).

Nelle nostre applicazioni abbiamo utilizzato un approccio misto: mediante simulazione e note tecniche algebriche, abbiamo dimostrato l'equivalenza dei due livelli descrittivi considerati, ossia correttezza e completezza dell'implementazione rispetto alla specifica.

2.2.5 Un'estensione: la validazione gerarchica. I moderni circuiti integrati a vasta scala di integrazione (*VLSI*) sono progettati con struttura gerarchica, la quale facilita la realizzazione di progetti molto complicati mediante suddivisione di un problema in più sotto-problemi di difficoltà minore.

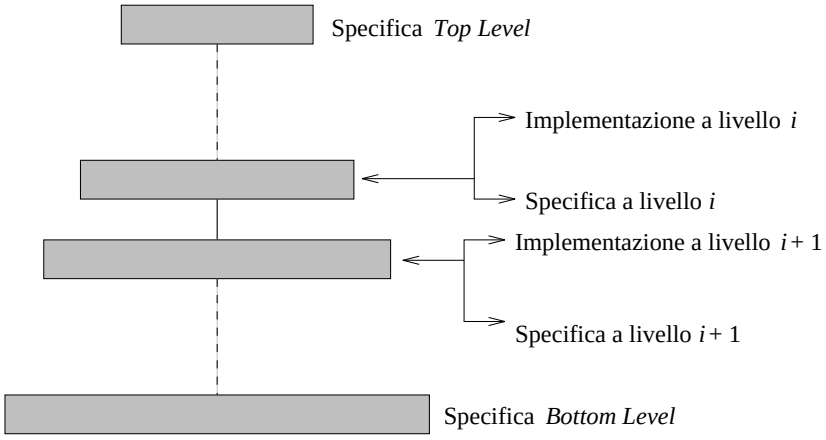

Figura1. Gerarchia di Validazione

I livelli di questa gerarchia di progetti vengono chiamati *livelli di astrazione*. Il processo di verifica formale esposto sopra può essere esteso descrivendo ciascun livello di astrazione del circuito: nasce allora il concetto di *verifica formale gerarchica* [CP88], schematizzato in fig. 1.

Si osservi che ogni livello descrittivo rappresenta contemporaneamente l'implementazione per il livello immediatamente più alto e la specifica per il livello immediatamente più basso, per cui, dopo aver dimostrato la correttezza del livello più astratto (*top level*) rispetto al modello di funzionamento, basterà dimostrare che, scendendo nella gerarchia, ogni livello implementa correttamente il precedente. Continuando così, giunti al livello più basso (*bottom level*), avremo dimostrato per transitività la correttezza di quest'ultimo livello di descrizione – molto concreto, un'effettiva implementazione – rispetto al modello[1].

Inoltre tale processo gerarchico è abbastanza semplice perchè si tratta semplicemente di stabilire un'opportuna relazione (par. 2.2.4) fra due livelli di descrizione molto vicini per astrazione.

La validazione dello *shift register* che presentiamo nella prossima sezione necessita di due soli livelli descrittivi grazie alla semplicità del dispositivo. Tuttavia la verifica formale gerarchica è già stata applicata con successo dagli autori a dispositivi più complicati quali sommatori binari e moltiplicatori (vedi par. 4).

3 La validazione di uno *shift register*

Lo *shift register* è un dispositivo abbastanza semplice, capace di ritardare la trasmissione di un'informazione binaria nel tempo: preso un bit ad un certo istante di tempo, il dispositivo lo ridà in output, inalterato, trascorso un certo numero di intervalli di *clock*.

Esso viene generalmente realizzato collegando due *inverter* [MC80] in serie, oppure tramite *flip-flop* [Flo86]. Noi faremo riferimento alla prima implementazione.

Si tenga presente che stiamo considerando un modello semplificato di *shift register*, che intendiamo validare mediante verifica formale con Algebre Evolventi [Bel95], allo scopo di confrontare il nostro risultato all'analogo tentativo presentato in [Dav90].

3.1 La specifica

Per farne la specifica, l'oggetto va pensato in maniera macroscopica e descritto così come viene visto dall'ambiente circostante (par. 2.2.2).

Dal punto di vista dello shift register (vedi fig. 2), l'input rappresenta un'entità incontrollabile, a priori non determinabile. Quindi, pensando a una formulazione mediante Algebre Evolventi, l'input va visto come una *funzione esterna* [Gur93a] il cui valore viene fornito da un oracolo senza alcuna legge predeterminata: tale acquisizione è peraltro molto vicina alla realtà. Detta funzione esterna assume valore *undef* quando il circuito, non avendo momentaneamente

[1] La validazione gerarchica può essere realizzata anche procedendo dal basso verso l'alto nella gerarchia di descrizioni. Tuttavia generalmente viene seguita l'altra direzione.

bisogno di utilizzare il dispositivo, non gli fornisce alcun valore di input. Analogamente, se lo *shift register* dà in output *undef*, vuol dire che, se k è il suo numero di *delay*, k intervalli di clock prima, il circuito non gli aveva fornito alcun input.

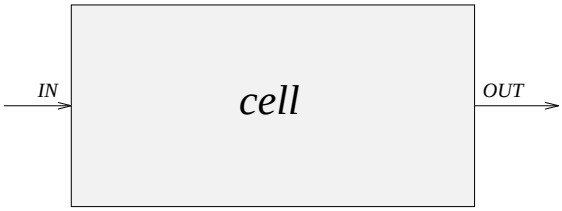

Figura2. Visione *Macroscopica* del Dispositivo

3.1.1 L'Algebra Cella ad Alto Livello. Introduciamo un'algebra evolvente[2] che formalizzi il dispositivo ad un alto livello di astrazione.

La segnatura. Lo *shift register* è dotato di una porta di ingresso dei dati e di una porta di uscita, che vengono formalizzate definendo l'universo:

$$Gates = \{IN, OUT\}$$

in cui i simboli IN e OUT vanno interpretati rispettivamente come porta di input e porta di output. Il valore della seconda è determinato da una funzione interna [Gur93a], mentre il valore della prima è dato da una funzione esterna, ossia acquista i valori scelti da un oracolo del tutto indipendente dal dispositivo.

Per accedere ai valori sulle porte definiamo un'opportuna funzione:

$$val : Gates \longrightarrow Bits \cup \{undef\}$$

essendo $Bits = \{0, 1\}$ l'universo dei possibili valori (esso coincide con l'universo base *Bool* presente in ogni algebra).

Per rappresentare gli stati di funzionamento e non funzionamento del sistema, introduciamo la funzione esterna nullaria *work* a valori booleani. Essa vale 1 se il circuito di cui fa parte il dispositivo è acceso, altrimenti vale 0.

Infine, per rappresentare la cella, utilizziamo una costante (di stato) [Gur93a] *cell*, sulla quale estendiamo la definizione della funzione *val*. Identificare tutta la cella unicamente e semplicemente con una costante, determina l'alto livello di astrazione di questa specifica.

Le interpretazioni di tutti gli oggetti dichiarati sopra, sono funzioni effettive sul superuniverso X dell'algebra evolvente [Gur93a].

[2] Il nome "Algebra Cella" deriva dal fatto che il nostro dispositivo viene per lo più utilizzato come cella, ossia come componente di dispositivi più complessi.

La regola di transizione. L'aggiornamento delle funzioni interne, in base ai criteri di funzionamento del dispositivo, è garantito dall'applicazione della seguente regola R di multi-aggiornamento condizionato:

```
if work then
    val(cell) := val(IN);
    val(OUT) := val(cell);
endif
```

Si osservi che l'applicazione della regola è controllata solo dalla funzione esterna *work*. Se il circuito è in funzione, *work* sarà vera per cui la regola R verrà applicata ripetutamente. Ad ogni applicazione, il valore in input viene conservato nella costante di stato *cell* e, nello stesso tempo, il suo valore attuale viene dato in output.

3.1.2 La correttezza della specifica. Per dimostrare che la specifica è corretta rispetto all'idea di funzionamento (ossia simula correttamente il funzionamento del dispositivo), dobbiamo dimostrare che essa può garantire che il bit sulla porta di input ad un certo istante (quindi in una certa algebra) si ripresenta sulla porta di output dopo due intervalli di tempo (cioè dopo due algebre).

Osservazione 1. *Si ricordi che il nostro formalismo rappresenta il trascorrere di un intervallo di tempo mediante passaggio da uno stato al successivo.*

Diamo allora il seguente teorema.

Teorema 1 (Correttezza della Specifica). *L'Algebra Cella ad Alto Livello specifica correttamente lo* shift register *con* delay *di due intervalli di* clock.

Proof. Dobbiamo provare che l'applicazione ripetuta della regola R induce una sequenza di algebre statiche che simulano il funzionamento dello *shift register*. Basta dimostrare che, sotto l'ipotesi

$$[work]^j = 1,\ j \geq 0$$

vale la seguente relazione di correttezza:

$$[val(IN)]^j = [val(OUT)]^{j+2},\ j \geq 0 \tag{1}$$

1) Sia $j = 0$. La tesi diventa:

$$[val(IN)]^0 = [val(OUT)]^2$$

L'algebra S_0 rappresenti la configurazione iniziale del dispositivo. In essa, l'interpretazione delle funzioni della segnatura è la seguente:
- $[work]^0 = 1$, per ipotesi
- $[val(IN)]^0$, valore in input
- $[val(cell)\,]^0 = undef$

– $[val(OUT)]^0 = undef$

L'applicazione della regola R induce la transizione:

$$S_0 \xrightarrow{R} S_1$$

Nella nuova algebra S_1 l'interpretazione delle funzioni risulta:
– $[work]^1 = 1$, per ipotesi
– $[val(IN)]^1$, nuovo valore in input
– $[val(cell)\,]^1 = [val(IN)]^0$
– $[val(OUT)]^1 = [val(cell)\,]^0 = undef$

Viene applicata la regola R, la quale provoca la transizione:

$$S_1 \xrightarrow{R} S_2$$

L'interpretazione delle funzioni nella nuova algebra S_2 è:
– $[work]^2 = 1$, per ipotesi
– $[val(IN)]^2$, nuovo valore in input
– $[val(cell)\,]^2 = [val(IN)]^1$
– $[val(OUT)]^2 = [val(cell)\,]^1 = [val(IN)]^0$

Come volevasi dimostrare.

2) Fissiamo $j > 0$. Dobbiamo dimostrare che:

$$[val(IN)]^j = [val(OUT)]^{j+2}$$

Sia S_j l'algebra che fotografa la configurazione del sistema all'istante k. L'interpretazione delle funzioni è:
– $[work]^j = 1$, per ipotesi
– $[val(IN)]^j$, nuovo valore in input
– $[val(cell)\,]^j = [val(IN)]^{j-1}$
– $[val(OUT)]^j = [val(cell)\,]^{j-1}$

L'applicazione dalla regola R induce la transizione:

$$S_j \xrightarrow{R} S_{j+1}$$

Nell'algebra S_{j+1} le funzioni della segnatura assumono i seguenti valori:
– $[work]^{j+1} = 1$, per ipotesi
– $[val(IN)]^{j+1}$, nuovo valore in input
– $[val(cell)\,]^{j+1} = [val(IN)]^j$
– $[val(OUT)]^{j+1} = [val(cell)\,]^j$

L'ulteriore applicazione di R induce:

$$S_{j+1} \xrightarrow{R} S_{j+2}$$

Nell'algebra S_{j+2} abbiamo:
– $[tiwork]^{j+2} = 1$, per ipotesi
– $[val(IN)]^{j+2}$, nuovo valore in input
– $[val(cell)\,]^{j+2} = [val(IN)]^{j+1}$
– $[val(OUT)]^{j+2} = [val(cell)\,]^{j+1} = [val(IN)]^j$

Da cui l'asserto.

Per quanto dimostrato ai punti (1) e (2), resta provata la tesi. □

Osservazione 2. *L'ipotesi del teorema 1,* $[\text{work}]^j = 1$ *per* $j \geq 0$ *si giustifica perchè nostro scopo è dimostrare la correttezza della specifica nel caso di funzionamento del dispositivo. Infatti, se indichiamo con* k *il primo istante in cui il circuito viene spento, e quindi il dispositivo cessa di funzionare, vale:*

$$[\text{work}]^j = 0$$

per cui l'algebra S_j *rappresenta l'ultima e definitiva configurazione istantanea del dispositivo, in quanto la regola* R *non è più applicabile.*

3.2 L'implementazione

Descrivere un dispositivo ad un basso livello di astrazione, ossia descriverne un'implementazione, vuol dire attenzionarlo in maniera microscopica, mettendo in luce il suo funzionamento interno (par. 2.2.3).

Prima di spiegare il funzionamento dell'implementazione di uno *shift register* tramite due *inverter* in serie, ricordiamo brevemente che un *inverter* è un dispositivo base per i VLSI [MC80, Dig89], che realizza la funzione logica *not*[3] e di cui stiamo dando per scontata la correttezza del funzionamento.

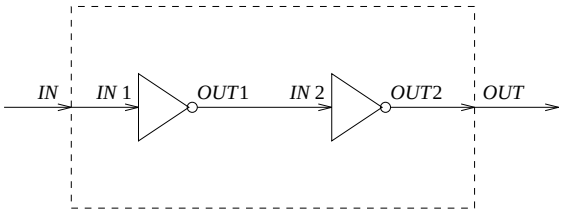

Figura3. Visione *Microscopica* del Dispositivo

L'idea di base di questa implementazione dello *shift register* è quella di invertire il bit per due volte, nel senso che il bit ottenuto come output dal primo inverter (quindi invertito una volta) viene dato in input al secondo che lo inverte una seconda volta (vedi fig. 3).

[3] Una decrizione di livello ancora più basso potrebbe prendere in considerazione il funzionamento interno dell'*inverter*, tuttavia perderebbe d'interesse per la semplicità del dispositivo.

3.2.1 L'Algebra Cella a Basso Livello. Introduciamo una nuova e più concreta algebra evolvente, capace di descrivere il funzionamento e l'interazione degli *inverter* presenti all'interno dello *shift register*.

La segnatura. Osservando la composizione del nostro dispositivo (fig. 3), si rendono necessari dei nomi per le porte di ingresso e di uscita dei due *inverter*, oltre a quelli per le porte dello *shift register*. Fissiamo allora il seguente universo:

$$Gates = \{IN, OUT, IN1, OUT1, IN2, OUT2\}$$

sugli elementi del quale, imponiamo le seguenti uguaglianze semantiche:

- $IN \equiv IN1$
- $OUT1 \equiv IN2$
- $OUT2 \equiv OUT$

L'interpretazione dei simboli per le porte e le motivazioni delle uguaglianze semantiche imposte, sono facilmente comprensibili analizzando la fig. 3.

Si definiscono anche le funzioni *val* e *work* come in 3.1.1.

L'interpretazione dei simboli presentati definisce il superuniverso X' dell'algebra evolvente.

La regola di transizione. La seguente regola R' di multi-aggiornamento condizionato è sufficiente a formalizzare a basso livello il funzionamento dello *shift register*:

```
if work then
    val(OUT1) := ¬val(IN);
    val(OUT) := ¬val(OUT1);
endif
```

Si osservi ancora una volta il ruolo della funzione *work* e si ricordino le uguaglianze semantiche di cui sopra.

Ad ogni applicazione di R', il valore di output del primo *inverter* diventa l'opposto del valore di input dello *shift register*, mentre il corrente valore di output del primo *inverter* viene negato ed assegnato come output allo *shift register*.

3.2.2 Validazione dell'implementazione. Avendo già dimostrato in 3.1 la correttezza della specifica rispetto al modello, per validare l'implementazione basta provare che essa soddisfa la specifica.

Nel nostro passo di raffinamento abbiamo costruito un'algebra evolvente più concreta L, che adesso vogliamo mettere in relazione con l'algebra evolvente più astratta H, attraverso una *funzione di verifica* $\mathcal{F}$ che mappa stati S' di L in stati $\mathcal{F}(S')$ di H e sequenze di regole R' di L in sequenze di regole $\mathcal{F}(R')$ di H, in modo da garantire la commutatività del diagramma in fig. 4.

La funzione di verifica $\mathcal{F}$ stabilisce la *correttezza* di L rispetto ad H se consente di ottenere in L gli stessi output che si otterrebbero in H sui medesimi

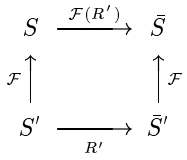

Figura4. Diagramma di Commutazione fra L ed H

input. In tal caso potremmo vedere una evoluzione nell'algebra più concreta L come corretta implementazione dell'evoluzione nell'algebra più astratta H.

Viceversa, la stessa funzione $\mathcal{F}$ stabilisce la *completezza* di L rispetto ad H se ogni computazione in H è l'immagine tramite $\mathcal{F}$ di una computazione in L, poichè in tal caso potremmo vedere ogni computazione astratta come implementata da una computazione concreta.

Se stabiliamo *correttezza e completezza* nel senso esposto sopra, avremo stabilito l'*equivalenza operazionale* delle due algebre evolventi H ed L.

La prova di correttezza e completezza necessita del seguente lemma.

Lemma 1. *Siano S_t uno stato dell'Algebra Shift ad Alto Livello ed S'_t uno stato dell'Algebra Shift a Basso Livello. Sussiste la seguente uguaglianza $\forall t \geq 0$:*

$$[val(cell)]^{S_t} = \neg[val(OUT1)]^{S'_t}$$

Proof. L'asserto è dimostrabile in maniera operativa, provando che i membri dell'uguaglianza assumono lo stesso valore ad ogni istante t, $(t \geq 0)$.

Per $t = 0$ la tesi è banalmente vera per valori *undef*.

Sia $t > 0$. Per applicazione delle regole R ed R' ad S_{t-1} ed S'_{t-1} rispettivamente, si ottengono le uguaglianze:

- $[val(cell)]^{S_t} = [val(IN)]^{S_{t-1}}$
- $[val(OUT1)]^{S'_t} = \neg[val(IN)]^{S'_{t-1}}$

da cui la tesi per l'identità tra $[val(IN)]^{S'_{t-1}}$ e $[val(IN)]^{S_{t-1}}$. □

Teorema 2 (Correttezza e Completezza dell'Implementazione).
Sia $\mathcal{F} : L \longrightarrow H$ (per ogni stato S'_t di L, $\mathcal{F}(S'_t)$ indica uno stato di H) una funzione fra algebre evolventi tale che:

a) Sussiste la seguente corrispondenza[4]

$$(IN1, OUT1 \equiv IN2, OUT2) \stackrel{\mathcal{F}}{\longmapsto} cell$$

fra una struttura composita in L e una funzione nullaria di H, sia sulla restrizione di $\mathcal{F}$ al superuniverso, che sulla restrizione al vocabolario di S'_t;

[4] Se in una corrispondenza tra algebre, una struttura composita viene associata ad un oggetto atomico, ogni elemento della struttura è mappato nell'oggetto stesso.

b) $\mathcal{F}$ *è l'identità su tutti gli altri oggetti della segnatura di* L *(tenuto conto delle uguaglianze semantiche date in 3.2.1) distinti da quelli citati citati al punto (a).*
c) $\mathcal{F}(R') = R$, R' *regola di* L, R *regola di* H;
d) $[val(cell)]^{\mathcal{F}(S'_t)} = \neg[(val(OUT1)]^{S'_t}$, *per* $t \geq 0$[5]

Tale funzione $\mathcal{F}$ *commuta il seguente diagramma:*

$$\begin{array}{ccc} S_t & \xrightarrow{\mathcal{F}(R')} & S_{t+1} \\ {\scriptstyle\mathcal{F}}\uparrow & & \uparrow{\scriptstyle\mathcal{F}} \\ S'_t & \xrightarrow[R']{} & S'_{t+1} \end{array}$$

Proof. Provare che $\mathcal{F}$ commuta il diagramma significa dimostrare che, se S'_t ed S'_{t+1}, $t \geq 0$, sono una coppia di stati consecutivi di L tramite R' – ossia S'_{t+1} è ottenuto per applicazione di R' su S'_t – allora $\mathcal{F}(S'_t)$ ed $\mathcal{F}(S'_{t+1})$ costituiscono una coppia di stati successivi di H tramite $\mathcal{F}(R') = R$ – ossia $\mathcal{F}(S'_{t+1})$ si ottiene per applicazione di R ad $\mathcal{F}(S'_t)$ –. A tal uopo basterà provare che la $\mathcal{F}$ soddisfa la seguente proprietà ad ogni istante di funzionamento t (ossia $work = 1$):

$$\mathcal{F}(S'_t) = S_t,\ t \geq 0.$$

Sia S'_t l'algebra in L che fotografa la configurazione del sistema all'istante t.

La tesi è banale per $t = 0, 1$.

Supponendo la tesi vera fino all'istante $t-1$, proviamo l'asserto per t. L'interpretazione delle funzioni in S'_t risulta:

$$\begin{aligned} &- [work]^{S'_t} = 1, \text{per ipotesi} \\ &- [val(IN)]^{S'_t}, \text{valore in input all'istante } t \\ &- [val(OUT1)]^{S'_t} = \neg[val(IN)]^{S'_{t-1}} \text{per la regola } R' \qquad (2) \\ &- [val(OUT)]^{S'_t} = \neg[val(OUT1)]^{S'_{t-1}} \text{per la regola } R' \qquad (3) \end{aligned}$$

Applicando la $\mathcal{F}$ ad S'_t otteniamo uno stato $\mathcal{F}(S'_t)$ in cui le funzioni sono così interpretate:

– $[work]^{\mathcal{F}(S'_t)} = [work]^{S'_t} = 1$, per la (b)
– $[val(IN)]^{\mathcal{F}(S'_t)} = [val(IN)]^{S'_t}$, input all'istante t, per la (b)
– $[val(cell)]^{\mathcal{F}(S'_t)} = [val(IN)]^{\mathcal{F}(S'_{t-1})}$, per la (a), la (d), la (2) e la (b)

[5] La proprietà di $\mathcal{F}$ è conseguenza del lemma 1. Essa stabilisce la relazione tra il valore dell'elemento atomico *cell*, nello stato immagine $\mathcal{F}(S'_t)$, e il valore di un elemento della struttura associata a *cell*, nello stato di partenza S'_t.

$$= [val(IN)]^{S_{t-1}}, \text{ per ipotesi induttiva}$$

$$- [val(OUT)]^{\mathcal{F}(S'_t)} = [val(IN)]^{\mathcal{F}(S'_{t-2})}, \text{ per la (a), la (3) e la (2)}$$
$$\text{applicata a } t-1 \text{ e } t-2 \text{ e la (b)}$$

$$= [val(IN)]^{S_{t-2}}, \text{ per ipotesi induttiva}$$

Lo stato così ottenuto coincide con S_t, essendo uno stato di funzionamento del sistema in cui il valore in input è quello all'istante t, il valore della cella quello in input all'istante precedente $(t-1)$, il valore in output quello in input due istanti prima $(t-2)$. □

3.3 Confronto

Abbiamo realizzato la validazione di un'implementazione di un modello semplificato di *shift register*, utilizzando il formalismo Algebre Evolventi, l'espressività del quale si commenta da sola, essendo riusciti a catturare il funzionamento del dispositivo mediante una sola regola di aggiornamento.

Per convincerci dello straordinario risultato ottenuto, basta fare il confronto con il tentativo di specifica dello stesso dispositivo, realizzato da B.S. Davie dell'Università di Edimburgo. In [Dav90] egli descrive lo stesso modello di *shift register* da noi preso in considerazione, mediante un estensione dell'*hardware description language* "Circal" ed ottiene una specifica di queste dimensioni:

```
BUF(w,x,y) <= if eqs(x,y) then
              ( {t} BUF(w,w,x)
              + {t, in>q : noteq(w,q)} BUF(q,w,x) )
              + if noteq(x,y) then
              ( {t, out<x} BUF(w,w,x)
              + {t, in>q : noteq(w,q),out<x} BUF(q,w,x) )
```

L'implementazione è espressa dal predicato:

```
IMP <= INV[mid/out] * INV[mid/in] - mid
```

insieme ad un'ulteriore specifica per "INV":

```
INV(m,n) <= if noteq(m,n) then
            ( {t} INV(m,n)
            + {t, in>p : noteq(p,m)} INV(p,m) )
            + if eqs(m,n) then
            ( {t, out<not(m)} INV(m,not(m))
            + {t, in>p : noteq(p,m),out<not(m)} INV(p,not(m)) )
```

che noi abbiamo evitato semplicemente aggiungendo alla nostra segnatura il simbolo di funzione "¬" interpretato come la funzione logica *not*. Attraverso lunghe e complicate trasformazioni sintattiche e infine con l'indispensabile – data la complessità delle formule logiche ottenute – ausilio di un dimostratore automatico, Davie prova l'equivalenza dei predicati "BUF" e "INV", dimostrando così che l'implementazione soddisfa la specifica.

Va osservato che le complicazioni sintattiche della specifica di Davie derivano dalla necessità di formalizzare l'idea di una evoluzione temporale, caratteristica insita, invece, nell'evoluzione delle nostre algebre.

Si tenga in considerazione anche lo sforzo necessario per comprendere a fondo le strutture sintattiche utilizzate, mentre la formulazione tramite Algebre Evolventi riduce tale sforzo ad apprendere il semplice concetto di interpretazione di funzioni, tipico della logica del primo ordine. Ciò rende le nostre descrizioni molto più intuitive.

4 Ulteriori risultati

La potenza e l'espressività delle Algebre Evolventi non sono rimaste allo stadio teorico. Tale formalismo è già stato utilizzato con successo dagli autori per la verifica formale di correttezza di dispositivi più realistici e più complicati.

Infatti è stato preso in considerazione un orologio di sistema "a due fasi che non si sovrappongono mai" [MC80], uno dei più diffusi sui moderni circuiti, nella verifica formale di uno "*shift register* bidirezionale, seriale-parallelo" [BR95]. Questo dispositivo [PE88] è costituito da un numero positivo, arbitrario e fissato di *celle*, ognuna delle quali è una versione più realistica del dispositivo presentato in questo articolo; esso può "shiftare" flussi di bit in entrambe le direzioni, così come agire da commutatore per flussi seriali in paralleli e viceversa. Anche in tal caso sono stati considerati due soli livelli di descrizione, ottenendo però quattro regole di aggiornamento nella specifica e sei nell'implementazione.

È stata quindi realizzata la verifica formale della correttezza [BR96] di un "sommatore binario parallelo" implementato tramite *full adder* e poi di un "moltiplicatore" costruito mediante sommatori [Flo86]. In questi casi è stata utilizzata la metodologia di validazione gerarchica presentata in 2.2.5, essendo stati necessari quattro livelli descrittivi nel primo caso e cinque nel secondo. È bastato studiare accuratamente i passi di raffinamento per riuscire a dimostrare la correttezza delle implementazioni considerate.

I risultati ottenuti finora ci permettono di sostenere le Algebre Evolventi come uno degli strumenti più indicati per la validazione di hardware e ci fanno nutrire un buon ottimismo sulla possibilità di estendere la metodologia di validazione presentata in questo articolo a dispositivi sempre più complessi.

References

[Bel95] G. Bella. Validazione di Hardware mediante Algebre Evolventi: I Circuiti VLSI. Tesi di Laurea in Scienze dell'Informazione, Università di Catania, 1995.

[BR95] G. Bella, E. Riccobene. A VLSI device validation by Evolving Algebras. *Da sottoporre.*

[BR96] G. Bella, E. Riccobene. Hardware hierarchical validation by Evolving Algebras. *Da sottoporre.*

[BM84] R. S. Boyer, J. S. Moore. "Proof-Checking, Theorem-Proving and Program Verification". *Contemporary Mathematics*, 1984.

[BM88] R. S. Boyer, J. S. Moore. *A Computational Logic Handbook.*

[Buc60] J. R. Büchi. "On a Decision Method in Restricted Second Order Arithmetic". *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science.* (Ed. E. Nagel et al.), Stanford University Press, 1960.

[CP88] P. Camurati, P. Prinetto. "Formal Verification of Hardware Correctness: Introduction and Survey of Current Research". *Formal Verification of Hardware Design.* (Ed. Michael Yoeli) IEEE Computer Society Press Tutorial, 1988.

[Dav90] B. S. Davie. *Formal Specification and Verification in VLSI Design.* Edinburgh University Press, 1990.

[Dig89] J. Di Giacomo. *VLSI Handbook.* McGraw-Hill, 1989. Speed-Independent Circuits". *Proceedings of the Fift MIT Conference on Advanced Research on VLSI.* (Eds. J. Allen, F. T. Leighton), Mit Press, Cambridge, 1988.

[Dil88] D. L. Dill. "Trace theory for Automatic Hierarchical Verification of Speed-Independent Circuits". *Proceedings of the Fift MIT Conference on Advanced Research on VLSI.* (Eds. J. Allen, F. T. Leighton), Mit Press, Cambridge, 1988.

[Eme90] E. A. Emerson. "Temporal and Modal Logic". *Handbook of Theoretical Computer Science.* (Ed. B. J. Van Leuween), Elsevier Science Publishers, Amsterdam, 1990.

[Flo86] T. L. Floyd. *Foundamentals of Digital Electronics.* Bell & Howell Company Press, 1986.

[GH93] Y. Gurevich, J. Huggins. "The semantics of the C programming language". In E. Börger, H. Kleine Büning, G. Jäger, S. Martini, and M. M. Richter, editors, *Computer Science Logic*, Lecture Notes in Computer Science 702, pages 274–309. Springer, 1993.

[Gor87] M. J. C. Gordon. HOL: "A Proof Generating System for Higher Order Logic". *VLSI Specification, Verification and Synthesys.* (Eds. G. Birtwistle, P. A. Subrahmanyam), Kluwer Academic Publishers, Boston, 1987.

[Gup92] A. Gupta. "Formal Hardware Verification Methods: A Survey".

[Gur93a] Y. Gurevich. "Evolving Algebras 1993: Lipari Guide". *Specification and Validation Methods*, Ed. E. Börger, Oxford University Press, 1994.

[Gur93b] Y. Gurevich. "Evolving Algebras: An Attempt to Discover Semantics". *Formal Methods in System Design*, 1992.

[Hat82] W. S. Hatcher. *The Logical Foundations of Mathematics.* Pergamon Press, Oxford, 1982.

[HKP82] D. Harel, D. Kozen, R. Parikh. "Process Logic: Expressiveness, Decidability and Completeness". *Journal of Computer and System Sciences.* 1982.

[HU79] J. E. Hopcroft, J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Koz83] D. Kozen. "Results on the Propositional mu-calculus". *Theoretical Computer Science.* 1983.

[MC80] C. Mead, L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[Men64] E. Mendelson. *Introduction to Mathematical Logic.* Van Nostrand, New York, 1964.

[MP82] Z. Manna, A. Pnueli. "Verification of Concurrent Programs: the Temporal Framework". *Correctness Problem in Computer Science.* (Eds. R. S. Boyer, J. S. Moore), Academic Press, London, 1982.

[PE88] D. A. Pucknell, K. Eshraghian. *Basic VLSI Design.* Prentice Hall, 1988.

[Rei83] D. Reidel. *Handbook of Philosophical Logic.* (Eds. D. Gabbay, F. Guenthner), Boston, 1983.

[Ric92] E. Riccobene. *Modelli Matematici per Linguaggi Logici.* Tesi di Dottorato, 1992.

[RU71] N. Reshcer, A. Urquhart. *Temporal Logic.*

[Ull84] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

[VW86] M. Vardi, P. Wolper. "An Automata-Theoretic Approach to Automatic Program Verification".". *Proceedings of the annual Symposium on Logic in Computer Science.* IEEE Computer Society Press, Washington, 1986.

[Wol85] P. Wolper. [BM84], "The Tableau Method for Temporal Logic: an Overview". *Logique et Analyse*, 1985.

This article was processed using the LaTeX macro package with LLNCS style