Bollettino
Unione Matematica Italiana

Sezione A – La Matematica nella Società e nella Cultura

Marco Maggesi, Carlos Simpson

Verifica automatica del ragionamento matematico

Bollettino dell'Unione Matematica Italiana, Serie 8, Vol. **9-A**—*La Matematica nella Società e nella Cultura* (2006), n.3-1, p. 361–389.

Unione Matematica Italiana

<http://www.bdim.eu/item?id=BUMI_2006_8_9A_3-1_361_0>

Bollettino dell'Unione Matematica Italiana, Unione Matematica Italiana, 2006.

Bollettino U. M. I.
La matematica nella Società e nella Cultura
Serie VIII, Vol. IX-A, Dicembre 2006/1, 361-389

Verifica automatica del ragionamento matematico

MARCO MAGGESI - CARLOS SIMPSON

1. – Introduzione.

Un giorno i computer potrebbero diventare più capaci degli esseri umani nel dimostrare teoremi. Ma, per il momento, quel giorno sembra ancora lontano e non pare che la *Dimostrazione Automatica dei Teoremi* possa intervenire nel lavoro quotidiano della grande maggioranza dei matematici ancora per qualche tempo. Invece la *Matematica Verificata al Calcolatore*, cioè l'uso del computer per certificare la correttezza di un ragionamento logico-matematico, è, già da qualche anno, una realtà che trova interesse e applicazione in alcuni segmenti sia della ricerca pura che del mondo produttivo. In un futuro non lontano, la richiesta di personale addestrato all'uso di tecniche di verifica automatica al computer potrebbe ampliarsi e costituire una nuova area di lavoro per una porzione rilevante dei matematici professionisti.

La differenza di base tra *dimostrazione automatica* e *verifica automatica* non ha bisogno di molte spiegazioni. La prima si occupa di costruire una macchina capace di dimostrare teoremi: idealmente la macchina dovrebbe sintetizzare la dimostrazione senza nessun aiuto da parte dell'uomo. La verifica automatica, invece, si occupa di costruire una macchina capace di leggere un testo matematico (codificato in un opportuno linguaggio simbolico) e segnalare ogni errore formale in esso contenuto, ovvero certificare che il testo non contiene nessun errore. La macchina dovrebbe essere capace di certificare testi matematici scritti in uno stile il più possibile rassomigliante a quello di un normale testo matematico (sintassi a parte).

Dimostrazione automatica e verifica automatica non sono realmente due domini distinti di ricerca, ma, piuttosto, due punti di vista com-

plementari sullo stesso problema. Se, durante la verifica di un testo matematico, alcune parti del ragionamento vengono «trovate» autonomamente dal computer, tanto meglio: in questo senso la verifica automatica non esclude a priori la dimostrazione automatica. Viceversa, un dimostratore automatico può essere istruito in molti modi da un matematico esperto, al fine di aiutarlo a sintetizzare una dimostrazione (ad esempio configurando oculatamente le strategie di ricerca, oppure premettendo alla dimostrazione di un risultato significativo una sequenza ben scelta di lemmi preparatori). In linea di principio, quindi, un dimostratore automatico può essere utilizzato non solo per trovare nuove dimostrazioni, ma anche per controllare una dimostrazione che già si conosce. È possibile che in futuro i sistemi di dimostrazione assistita saranno sempre più somiglianti ad un ibrido tra un programma di verifica automatica ed uno di dimostrazione automatica. In ogni caso un obiettivo fondamentale compete ad entrambe le discipline: impiegare il computer per produrre «certezze matematiche» da poter sfruttare nella ricerca e nel mondo produttivo.

Esiste un'ampia letteratura, anche a livello divulgativo, sulla matematica formalizzata. Come punti di partenza (anche per ricerche in Internet) consigliamo [1, 19, 14].

In questo lavoro, dopo avere esaminato in dettaglio un semplice esempio nella sezione 2, faremo una breve panoramica sulle applicazioni di questa disciplina nella sezione 3. Nella sezione 4 discuteremo l'impatto che le tecniche di matematica assistita al computer potrebbero avere sull'organizzazione della ricerca. Infine, azzarderemo alcune previsioni su quello che ci riserverà il prossimo futuro nella sezione 5.

Desideriamo ringraziare profondamente André Hirschowitz per averci introdotto, con entusiasmo e competenza, in questo affascinante settore di ricerca.

2. – Un semplice esempio.

Come spunto per introdurre una discussione più generale, illustreremo in questa sezione un semplice esempio. Presenteremo un

frammento della teoria dei gruppi in tre modi diversi:

1. Nel linguaggio matematico consueto;
2. Con il codice per un verificatore automatico di teoremi;
3. Con il codice per un dimostratore automatico di teoremi.

Si tratta di un esempio molto limitato che in nessun modo può dare un'idea che rispecchi l'attuale stato dell'arte delle tecniche di automazione disponibili oggigiorno. In particolare, le dimostrazioni che vedremo si limiteranno a ragionamenti di natura equazionale (non vi compaiono, ad esempio, dimostrazioni per induzione).

Abbiamo scelto Isabelle/HOL [21, 31] come software di formalizzazione e Otter [35] come dimostratore automatico di teoremi. Nel panorama attuale che comprende varie decine di software, la nostra scelta non può che risultare almeno in parte arbitraria. Si tratta comunque di due software che hanno un ampio seguito di utenti, che sono stati utilizzati in diversi lavori di ricerca pionieristici ed originali.

2.1. *Primo stile: linguaggio matematico consueto.*

Consideriamo un gruppo G, cioè una quadrupla $(G, \circ, (\,\cdot\,)^{-1}, e)$ formata da un insieme G, un prodotto $\circ\colon G \times G \to G$, una applicazione $(\,\cdot\,)^{-1}\colon G \to G$ e da un elemento $e \in G$ che soddisfano i tre assiomi

ASSIOMA 1 (Associatività). – $\forall x\, y\, z \in G,\ x \circ (y \circ z) = (x \circ y) \circ z$

ASSIOMA 2 (Neutro a sinistra). – $\forall x \in G,\ e \circ x = x$

ASSIOMA 3 (Inverso a sinistra). – $\forall x \in G,\ x^{-1} \circ x = e$

Conveniamo in seguito di considerare il prodotto $\circ$ come associativo a destra. A partire da questi assiomi vogliamo dimostrare i seguenti risultati:

LEMMA 4 (Cancellazione a sinistra). – $\forall x\, y\, z \in G, x \circ y = x \circ z \iff y = z$.

DIMOSTRAZIONE. – Supponiamo di avere $x \circ y = x \circ z$. Allora, per la proprietà associativa, abbiamo che $(x^{-1} \circ x) \circ y = (x^{-1} \circ x) \circ z$, da cui segue $y = z$ per gli assiomi 2 e 3. Viceversa, se $y = z$ segue subito che $x \circ y = x \circ z$. □

LEMMA 5 [Neutro a destra]. – $\forall x \in G, x \circ e = x$.

DIMOSTRAZIONE. – Abbiamo che $x^{-1} \circ (x \circ e) = (x^{-1} \circ x) \circ e = e \circ e = e = x^{-1} \circ x$. La tesi quindi segue dal lemma 4. □

LEMMA 6 [Inverso a destra]. – $\forall x \in G, x \circ x^{-1} = e$.

DIMOSTRAZIONE. – Analoga alla precedente. □

LEMMA 7 [Inverso del prodotto]. – $\forall x\, y \in G, (x \circ y)^{-1} = y^{-1} \circ x^{-1}$

DIMOSTRAZIONE. – Abbiamo che $(x \circ y) \circ (x \circ y)^{-1} = e$ per il lemma 6. Inoltre $e = x \circ (y \circ y^{-1}) \circ x^{-1}$ per i lemmi 5 e 6. Quindi $(x \circ y) \circ (x \circ y)^{-1} = (x \circ y) \circ (y^{-1} \circ x^{-1})$. La tesi segue applicando il lemma 4. □

Adesso dimostriamo che ogni gruppo di ordine 2 è abeliano:

TEOREMA 8. – *Se $x \circ x = e$ per ogni $x \in G$, allora G è abeliano.*

DIMOSTRAZIONE. – In un gruppo di ordine 2 ogni elemento è l'inverso di se stesso. Infatti se $x \circ x = e$, moltiplicando ambo i membri a sinistra per x^{-1} abbiamo $x^{-1} \circ (x \circ x) = x^{-1} \circ e$ da cui segue $x = x^{-1}$. Per il lemma 7 abbiamo $x \circ y = (x \circ y)^{-1} = y^{-1} \circ x^{-1} = y \circ x$. □

Infine, indichiamo con $[x, y] = x \circ y \circ x^{-1} \circ y^{-1}$ il commutatore della coppia (x, y) di elementi di G, e dimostriamo che se ogni commutatore appartiene al centro di G, allora vale l'equazione $[x, y \circ z] = [x, y] \circ [x, z]$.

TEOREMA 9. – *Se $\forall x\, y\, z \in G, [x, y] \circ z = z \circ [x, y]$, allora $\forall x\, y\, z \in G$, $[x, y \circ z] = [x, y] \circ [x, z]$.*

DIMOSTRAZIONE. – Dall'ipotesi si ha

$$z \circ x^{-1} \circ z^{-1} \circ y^{-1} = x^{-1} \circ ([x, z] \circ y^{-1}) = x^{-1} \circ y^{-1} \circ [x, z]$$

da cui segue la tesi, moltiplicando ambo i membri a sinistra per $x \circ y$ ed usando il lemma 7. □

2.2. *Secondo stile: verifica al computer.*

Vediamo come questi risultati appena esposti possono essere formalizzati con un programma per la verifica delle dimostrazioni. ([1])

In figura 1 è riportato il codice per il sistema Isabelle/HOL che introduce gli assiomi per la teoria dei gruppi.

```
1   theory Group_Demo = Main:
2   locale group =
3     fixes prod :: "['a, 'a] => 'a" (infixl "**" 80)
4     and one and inv ("i(_)" [90] 91)
5     assumes assoc: "(x ** y) ** z = x ** (y ** z)"
6     and l_one: "one ** x = x"
7     and l_inv: "i x ** x = one"
```

Fig. 1. – Assiomi per la teoria dei gruppi in Isabelle/HOL.

I comandi delle prime due righe sono di natura, per così dire, *burocratica* e non sono essenziali per la comprensione di quello che segue. Servono a dare un nome alla teoria che stiamo costruendo (in questo caso «`Group_Demo`») e all'insieme di assiomi che stiamo considerando in questa teoria (cioè «`group`») per permetterci di riusare

([1]) Avvertiamo subito il lettore che siamo costretti a dare delle spiegazioni assai parziali del codice che presentiamo in questa sezione e nella successiva. Si consideri che per imparare ad usare le funzionalità di base di un sistema di dimostrazione assistita possono essere necessarie diverse settimane. Tuttavia, al di là degli aspetti tecnici, i punti fondamentali dovrebbero risultare chiari seguendo il parallelo con la sezione precedente.

queste definizioni in altri contesti. Le righe 3 e 4 introducono le costanti «prod», «one», «inv» con le relative abbreviazioni speciali («**» per «prod» e «i» per «inv») ed altre informazioni riguardanti il *tipo* («['a, 'a] ⇒ 'a») di queste costanti. Le righe successive dichiarano gli assiomi per questa teoria: «assoc», «l_one», «r_one».

In ogni formula la quantificazione universale delle variabili è implicita.

Adesso che abbiamo introdotto le costanti e gli assiomi che ci servono, possiamo cominciare a derivare i risultati che ci interessano. Cominciamo con la formalizzazione del lemma 4. Il codice relativo si trova in figura 2.

```
 8  lemma (in group) l_cancel: "(x ** y = x ** z) = (y = z)"
 9  proof
10    assume "x ** y = x ** z"
11    hence  "(i x ** x) ** y = (i x ** x) ** z"
12      by (simp add: assoc)
13    thus "y = z" by (simp add: l_one l_inv)
14  next
15    assume "y = z" thus "x ** y = x ** z" by simp
16  qed
```

Fig. 2. – Formalizzazione del lemma 4.

Alla riga 8 viene introdotto l'enunciato del lemma, al quale viene assegnato il nome «l_cancel». L'annotazione «(in group)» indica che questo lemma va letto nel contesto degli assiomi introdotti precedentemente con il comando «locale group» alla seconda riga del file. Il segno di uguaglianza principale nell'enunciato del lemma (riga 8) va considerato come un «se e solo se». La dimostrazione è divisa in due parti separate dalla parola chiave «next», corrispondenti alle due implicazioni («se» e «solo se») necessarie per dimostrare l'equivalenza logica. Per capire meglio questo script, consigliamo di mettere a confronto queste righe con la dimostrazione data in 4. La tattica «simp» serve per semplificare l'enunciato. Isabelle ha un archivio di regole di semplificazione e altre possono essere aggiunte all'occorrenza, come nel caso dell'i-

struzione «simp add: assoc» che usiamo alla riga 12. Ogni passaggio viene giustificato mediante l'uso di una tattica con il comando «by», oppure, quando è necessario un ragionamento più articolato, usando il costrutto «proof... end».

La formalizzazione del lemma 5, riportata in figura 3, non contiene sostanziali novità rispetto a quella precedente. Alla riga 20 siamo costretti ad orientare l'assioma di associatività con la proprietà simmetrica dell'uguaglianza usando l'espressione «sym[OF assoc]» prima di passarlo come regola di riscrittura alla tattica «simp».

```
17  lemma (in group) r_one: "x ** one = x"
18  proof -
19    have "i x ** (x ** one) = i x ** x"
20      by (simp add: l_one l_inv sym[OF assoc])
21    thus ?thesis by (simp add: l_cancel)
22  qed
```

Fig. 3. – Formalizzazione del lemma 5.

Per completezza, riportiamo anche in figura 4 la dimostrazione del lemma 6, del tutto analoga alla precedente.

```
23  lemma (in group) r_inv: "x ** i x = one"
24  proof -
25    have "i x ** (x ** i x) = i x ** one"
26      by (simp add: r_one l_one l_inv sym[OF assoc])
27    thus ?thesis by (simp add: l_cancel)
28  qed
```

Fig. 4. – Formalizzazione del lemma 6.

Un po' di lavoro in più è necessario per la dimostrazione del lemma 7, perché richiede un numero maggiore di passaggi di riscrittura che vengono resi con il costrutto «have ... also have ... finally». La dimostrazione è contenuta in figura 5.

```
29  lemma (in group) inv_mul: "i (x ** y) = i y ** i x"
30  proof -
31    have "(x ** y) ** i (x ** y) = one" by (simp add: r_inv)
32    also have "one = x ** (y ** i y) ** i x"
33      by (simp add: r_inv r_one)
34    finally have
35      "(x ** y) ** i (x ** y) = (x ** y) ** (i y ** i x)"
36      by (simp add: assoc)
37    thus ?thesis by (simp add: l_cancel)
38  qed
```

Fig. 5. – Formalizzazione del lemma 7.

La dimostrazione del teorema 8 (figura 6) richiede un risultato intermedio, al quale viene assegnato il nome «`inv_id`» (riga 43), il quale poi viene usato come regola di semplificazione alla riga successiva. Anche all'ipotesi del teorema viene dato il nome esplicito «`H`» per poterla richiamare alla riga 42. Nell'ipotesi «`H`» e alle righe 42 e 43 è necessario usare esplicitamente il quantificatore universale $\forall$ rappresentato dal punto esclamativo «`!`». La notazione «`...`» che compare alla riga 45 è una abbreviazione che serve ad indicare il secondo membro dell'ultima uguaglianza dimostrata (in questo caso «`i (x ** y)`»).

```
39  lemma (in group) assumes H: "!x. x ** x = one"
40    shows "x ** y = y ** x"
41  proof -
42    have "!x. x ** i x = x ** x" by (simp add: r_inv H)
43    hence inv_id: "!x. i x = x" by (simp add: l_cancel)
44    have "x ** y = i (x ** y)" by (simp add: inv_id)
45    also have "... = i y ** i x" by (simp add: inv_mul)
46    finally show ?thesis by (simp add: inv_id)
47  qed
```

Fig. 6. – Formalizzazione del teorema 8.

Infine, la figura successiva mostra la formalizzazione del teorema 9. L'enunciato contiene anche la definizione di commutatore di un gruppo indicato con la costante «`h( , )`» insieme alla proprietà «`h_def`» che lo

caratterizza. Per il resto, la dimostrazione è del tutto somigliante alle precedenti.

```
48  lemma (in group)
49    fixes h :: "['a, 'a] => 'a"
50    assumes h_def: "!x y. h x y = x ** y ** i x ** i y"
51    assumes H: "!x y z. h x y ** z = z ** h x y"
52    shows "h x (y ** z) = h x y ** h x z"
53  proof -
54    have "z ** i x ** i z ** i y = i x ** (h x z ** i y)"
55      by (simp add: H h_def sym[OF assoc] l_one l_inv)
56    also have "... = i x ** i y ** h x z"
57      by (simp add: H sym[OF assoc])
58    finally have
59      "z ** i x ** i z ** i y = i x ** i y ** h x z" .
60    thus ?thesis by (simp add: h_def assoc inv_mul)
61  qed
```

Fig. 7. – Formalizzazione del teorema 9.

Questo conclude la nostra formalizzazione della sezione 2.1. Ovviamente, gli stessi teoremi avrebbero potuto essere formalizzati in diversi altri modi. ([2])

A questo punto il lettore può avere l'impressione che l'interazione con Isabelle/HOL sia basata su un numeroso e complesso insieme di comandi. In realtà, i costrutti necessari non sono poi così tanti: i comandi contenuti nel codice che abbiamo appena illustrato, insieme a pochi altri, sono già sufficienti per lo sviluppo di una qualsiasi teoria matematica. Tuttavia per avere una visione unitaria sul funzionamento e l'uso di Isabelle/HOL è necessario conoscere qualche ulteriore dettaglio sulla sua struttura interna. In particolare, vale la pena menzionare che Isabelle/HOL ha un rigoroso ed elegante fondamento costruito a partire da due elementi importanti: (i) attraverso l'adozione di un sistema logico ben noto e molto studiato (un'evoluzione della teoria dei tipi semplice di Alonzo Church [9]); (ii) con una architettura

([2]) Ad esempio, avremmo potuto usare tattiche più sofisticate di «`simp`» per far trovare ad Isabelle alcune dimostrazioni o parti di una dimostrazione in modo automatico.

software avanzata tale da garantire la correttezza dell'intero sistema, che si basa sull'idea di ricondurre ogni parte del programma all'uso di un piccolo numero di primitive implementate in un nucleo minimale. Il «nucleo» di Isabelle è il responsabile finale di qualsiasi manipolazione logica che venga effettuata. In questo modo, il sistema può essere esteso con nuove funzionalità sempre più evolute, senza che ne venga mai compromessa la coerenza nel suo insieme.

Sottolineiamo che le dimostrazioni vengono inserite *interattivamente*. Per avere un'idea si veda la schermata in figura 8. Come si vede, la finestra superiore contiene il codice della dimostrazione e quella inferiore lo stato della dimostrazione. La parte di codice già verificata viene evidenziata in blu. È possibile *navigare* nella dimostrazione avanzando nella verifica o tornando indietro (ad esempio quando vogliamo cambiare il codice perché abbiamo imboccato un vicolo cieco).

Fig. 8. – Una schermata della dimostrazione interattiva con Isabelle.

Sono allo studio anche interfacce grafiche avanzate, nelle quali viene impiegato il mouse per le azioni più comuni necessarie durante una dimostrazione [5].

2.3. *Terzo stile: dimostrazione automatica.*

Adesso mostriamo in che modo i risultati precedenti possono essere prodotti con il sistema di dimostrazione automatica Otter [35]. Consideriamo, ad esempio, lo script di figura 9 che, pur essendo molto elementare, permette di ricavare alcuni risultati basilari di teoria dei gruppi.

Per grandi linee, questo codice dovrebbe essere abbastanza chiaro. Il prodotto, l'elemento neutro e l'inverso sono indicati rispettivamente mediante «`f( , )`», «`e`» e «`g( )`». Gli assiomi della teoria dei gruppi si trovano alle righe 4, 5 e 6. Otter considera le ultime lettere dell'alfabeto (`u`, `v`, `x`, ...) come variabili (cioè vengono implicitamente quantificate universalmente in ogni formula) e tutte le altre come costanti, coerentemente con l'abituale pratica matematica. Il lettore sarà probabilmente sorpreso di vedere alla riga 3 la proprietà riflessiva dell'uguaglianza, mentre non compaiono le proprietà simmetrica e transitiva. La ragione sta nel fatto che Otter utilizza una regola d'inferenza molto potente chiamata *paramodulazione*, che rende questi ultimi assiomi superflui.

```
1   set(auto).
2   list(usable).
3   x = x.
4   f(e,x) = x.
5   f(g(x),x) = e.
6   f(f(x,y),z) = f(x,f(y,z)).
7   end_of_list.
```

Fig. 9. – Script per la ricerca di un set completo di regole di riscrittura per i gruppi liberi.

Lo script di figura 9 produce come risultato le formule riportate in figura 10 dove si vede come Otter, a partire dagli assiomi della teoria dei gruppi, sia capace di ricavare, tra gli altri, i lemmi 5 (elemento neutro a destra, riga 5), 6 (inverso a destra, riga 8), 7 (inverso di un prodotto, riga 10). Vale la pena di sottolineare che, in questo caso,

Otter non si limita trovare le dimostrazioni, ma addirittura sintetizza gli enunciati «interessanti» ([3]).

```
1   f(e,x)=x.
2   f(g(x),x)=e.
3   f(f(x,y),z)=f(x,f(y,z)).
4   f(g(x),f(x,y))=y.
5   f(x,e)=x.
6   g(e)=e.
7   g(g(x))=x.
8   f(x,g(x))=e.
9   f(x,f(g(x),y))=y.
10  g(f(x,y))=f(g(y),g(x)).
```

Fig. 10. – Il set di regole di riscrittura per i gruppi liberi trovato da Otter.

Consideriamo adesso il lemma 8: ogni gruppo di ordine 2 è abeliano. La dimostrazione di questo lemma può essere sintetizzata automaticamente con Otter usando il codice in figura 11.

```
1   set(auto).
2   list(usable).
3   x = x.
4   f(e,x) = x.
5   f(g(x),x) = e.
6   f(f(x,y),z) = f(x,f(y,z)).
7   f(x,x)=e.
8   f(a,b) != f(b,a).
9   end_of_list.
```

Fig. 11. – Dimostrazione del teorema 8 in Otter.

L'istruzione «`set(auto).`» alla prima riga istruisce Otter perché usi una strategia automatica di dimostrazione: ciò significa che dele-

([3]) Alcuni lettori avranno riconosciuto che, in questo caso, la strategia di Otter altro non è che l'applicazione dell'algoritmo di Knuth-Bendix per la ricerca di un insieme completo di regole di riscrittura [25].

ghiamo al sistema la responsabilità di scegliere la strategia migliore per trovare la dimostrazione. Il programma metterà in atto un'euristica affinata dagli sviluppatori in vari anni di esperimenti. La vera e propria lista di assiomi e ipotesi si trova racchiusa tra le istruzioni `list(usable)` e `end_of_list`. La differenza importante, rispetto allo script precedente, risiede nella penultima riga che fornisce ad Otter la negazione della tesi: «`f(a,b)!=f(b,a).`». In altre parole, chiediamo ad Otter di trovare una dimostrazione per assurdo.

Quando lo script precedente viene inserito, Otter risponde in maniera pressoché istantanea (nell'ordine di qualche millesimo di secondo su di un normale PC) fornendo una dimostrazione del teorema e diverse altre informazioni che includono delle statistiche e una traccia dell'albero di ricerca della dimostrazione. Evitiamo di riportare gran parte di questi dati, limitandoci a copiare soltanto la dimostrazione trovata da Otter, la quale si presenta come in figura 12.

```
1   1 [] f(a,b)!=f(b,a).
2   2 [copy,1,flip.1] f(b,a)!=f(a,b).
3   5,4 [] f(e,x)=x.
4   8 [] f(f(x,y),z)=f(x,f(y,z)).
5   10 [] f(x,x)=e.
6   12 [para_into,8.1.1.1,10.1.1,demod,5,flip.1] f(x,f(x,y))=y
7   16 [para_into,8.1.1,10.1.1,flip.1] f(x,f(y,f(x,y)))=e.
8   19,18 [para_into,12.1.1.2,10.1.1] f(x,e)=x.
9   26 [para_from,16.1.1,12.1.1.2,demod,19,flip.1] f(x,f(y,x))
10  30 [para_from,26.1.1,12.1.1.2] f(x,y)=f(y,x).
11  31 [binary,30.1,2.1] $F.
```

Fig. 12. – Output dello script di figura 11.

Ci limiteremo a spiegare solo superficialmente questo listato. Ad ogni riga corrisponde una nuova formula che viene dimostrata e aggiunta alla base di dati del motore di ricerca e, quindi, utilizzata nei passi successivi della dimostrazione. La formula stessa si può leggere alla fine della riga. Le espressioni tra parentesi quadre indicano le regole di inferenza (`para_into`, `demod`, eccetera) che il programma ha utilizzato per derivare la formula. La dimostrazione termina con la

derivazione della formula «Falso» (indicato dal simbolo `$F`) che conclude la dimostrazione per assurdo cominciata con la negazione della formula «`f(x,y)=f(y,x)`», come suggerito implicitamente dal modo in cui abbiamo fornito il problema al programma.

```
1   set(knuth_bendix).
2   assign(pick_given_ratio,8).
3   assign(stats_level,1).
4   lex([a,b,c,e,g(_),h(_,_),f(_,_)]).
5   list(usable).
6     f(f(x,y),z)=f(x,f(y,z)).
7     f(e,x)=x.
8     f(g(x),x)=e.
9     h(a,f(b,c))!=f(h(a,b),h(a,c)).
10  end_of_list.
11  list(sos).
12    h(x,y)=f(x,f(y,f(g(x),g(y)))).
13    f(h(x,y),z) = f(z,h(x,y)).
14  end_of_list.
```

Fig. 13. – Script per la dimostrazione del teorema 9.

Presentiamo infine uno script per la ricerca della dimostrazione del teorema 9. Anche in questo caso Otter sarebbe in grado di trovare la dimostrazione in maniera completamente automatica. Tuttavia, il tempo necessario per la ricerca questa volta è di gran lunga superiore a quello dello script di figura 11 (circa 40 secondi sul nostro computer, cioè nell'ordine di 10 000 volte più lento). Inoltre, anche la dimostrazione generata è assai più complicata: il listato prodotto è di 101 righe. Questa differenza colpisce soprattutto se mettiamo a confronto i due teoremi così come appaiono nelle sezioni 2.1 e 2.2 dove, a prima vista, sia i loro enunciati che le relative dimostrazioni sembrano essere di complessità comparabile. È facile intuire che, per problemi di poco più complessi, diventa assai difficile prevedere in quali casi è possibile produrre delle dimostrazioni automaticamente e in quali, invece, è necessario un tempo di esecuzione proibitivo.

In questo caso, per aiutare Otter a sintetizzare una dimostrazione

migliore e in modo più efficiente, è necessario fornire qualche indicazione in più sulla strategia di ricerca, come nello script di figura 13. Con questo script, Otter risponde in circa un decimo di secondo e produce una dimostrazione di 58 righe (il codice riportato è una leggera modifica di quello di John Kalman disponibile dalla pagina internet di Otter ([4])).

Nello script viene specificata esplicitamente la strategia di dimostrazione da usare (riga 1) e vengono forniti dei parametri addizionali, come, ad esempio, il modo con cui misurare la complessità di un termine (il comando «`lex`» alla riga 4). Durante la ricerca della dimostrazione, Otter mantiene una base di dati delle formule che possono essere usate nella dimostrazione, catalogate, a seconda di come verranno impiegate, in varie liste tra cui quelle denominate «`usable`» e «`sos`» che compaiono nel nostro codice. La prima lista contiene le formule che possono essere immediatamente utilizzate per creare nuove inferenze, mentre la seconda va considerata come una specie di lista d'attesa. Il contenuto delle liste «`usable`» e «`sos`», cambia dinamicamente nel corso della ricerca della dimostrazione. Se le ipotesi e gli assiomi non vengono ripartiti inizialmente nelle due liste in maniera opportuna, Otter non è capace di trovare la dimostrazione ([5]).

La morale che possiamo trarne è che, se pure è possibile trovare una dimostrazione in modo automatico, scrivere una *buona* dimostrazione rimane un processo essenzialmente creativo.

3. – Esempi e applicazioni.

3.1. *Verifica del Software e dell'Hardware.*

È ben noto che la fase di verifica (la ricerca dei *«bug»*) risulta essere spesso una delle più onerose tra quelle necessarie nella realizzazione del software e dell'hardware. Assicurarsi che un dato dispositivo

([4]) http://www-unix.mcs.anl.gov/AR/otter/examples33/kalman/ex_4.in

([5]) Ad esempio, nel presente caso, se tutte le ipotesi vengono inserite nella lista «`usable`», il programma termina immediatamente senza trovare la dimostrazione.

(programma o circuito) si comporti nel modo atteso in ogni varietà di circostanze significa controllare che questo aderisca ad un insieme preciso e coerente di specifiche, cioè, in definitiva, a dimostrare un teorema. L'automazione di questo processo di verifica e la realizzazione di programmi adatti a questo specifico obiettivo costituisce oggi uno dei maggiori filoni di ricerca e una delle maggiori motivazioni allo sviluppo di sistemi di dimostrazione assistita. Con l'aiuto di un programma al calcolatore è possibile, almeno in linea di principio, realizzare dispositivi *perfetti* fin dalla loro concezione. Specialmente quando si tratta di realizzare sistemi critici (un dispositivo per il controllo di un aereo o un programma di crittografia), l'uso di tecniche di questo tipo può essere non solo utile, ma anche auspicabile.

L'argomento, a prima vista, ha solo un interesse marginale per chi si occupa della dimostrazione al computer di teoremi di matematica pura. Ma vale la pena di sottolineare che la verifica del software e dell'hardware è certamente uno dei problemi guida nella ricerca in questo settore e la maggior parte dei sistemi di dimostrazione assistita oggi disponibili (tra questi anche quelli tra i più utilizzati per la dimostrazione di teoremi di matematica pura) vengono disegnati e sviluppati con l'esplicita intenzione di fornire un ambiente specializzato per questo scopo.

Questo si rivela, ad esempio, nella scelta della *logica* implementata nel sistema (cioè il sistema formale nel quale si enunciano e dimostrano i teoremi): classicamente, la matematica viene fondata a partire da una teoria degli insiemi del primo ordine (ad esempio quella di Zermelo-Fränkel), mentre la maggior parte dei software di dimostrazione al computer usa invece teorie diverse, spesso costruttive (intuizioniste) e basate su qualche variante del lambda-calcolo tipato.

Bisogna comunque considerare che la dimostrazione della correttezza di un algoritmo può essere legata a risultati di matematica pura anche profondi e viceversa. Quindi, ogni sistema di dimostrazione assistita deve necessariamente avere una certa versatilità da questo punto di vista. Inoltre, il fatto che sia il linguaggio degli insiemi a costituire il fondamento della matematica odierna è forse una questione più teorica che pratica: probabilmente la maggior parte dei matematici non è capace di recitare a memoria neanche uno dei vari sistemi di

assiomi formulati per la teoria degli insiemi. Ad esempio, nella sezione 2.2 il lettore certamente non avrà avuto difficoltà ad accettare che stessimo parlando della teoria dei gruppi, se pur utilizzando la teoria dei tipi anziché quella degli insiemi. Si tratta di un procedimento di abduzione che si rivela essere straordinariamente semplice per il cervello umano.

3.2. *Matematica formalizzata disponibile allo stato attuale.*

Esistono già parecchie librerie di teorie matematiche formalizzate al computer. Per dare un'idea indicativa, possiamo forse dire che, attualmente, è stato già superato il livello corrispondente a quello della preparazione di uno studente ai primi anni di università. In alcuni settori specifici come la logica combinatorica e altre aree particolarmente vicine all'informatica teorica, le teorie formalizzate disponibili sono già assai avanzate e il computer si sta proponendo come un mezzo importante di sostegno alla ricerca.

Segnaliamo che una porzione significativa delle librerie oggi disponibili sono state sviluppate con una predilezione verso l'intuizionismo e il costruttivismo e possono, quindi, essere utilizzate sia in matematica classica che costruttiva.

Quando, però, ci si concentra ad esaminare la situazione all'interno di ogni singolo software di dimostrazione assistita, il discorso cambia decisamente perché, di fatto, ciascuna libreria matematica è vincolata ad essere usata con il programma nel quale è stata sviluppata. Trasportare una libreria da un sistema ad un altro è teoricamente possibile, ma praticamente molto difficile, almeno allo stato attuale, per il fatto che ogni sistema implementa una logica differente. Come risultato, ogni sistema può disporre di librerie piuttosto avanzate in un certo ambito matematico, ma essere assai carente in molti altri.

Elencare tutti i contributi di Matematica formalizzata oggi disponibili sarebbe impossibile. Ci limiteremo a citare un singolo esempio notevole (non necessariamente per la difficoltà matematica intrinseca del teorema formalizzato) all'interno delle varie discipline, evitando di menzionare le formalizzazioni strettamente pertinenti all'informatica teorica (che sono

ancora più numerose). Per chi è interessato a farsi un'idea più precisa, consigliamo di esplorare i siti Internet dei maggiori sistemi di dimostrazione assistita. Ad esempio, per il sistema Isabelle, rimandiamo alla pagina web di «The Archive of Formal Proofs» [2]. Riguardo ai software oggi disponibili consigliamo il lavoro di Wiedijk [45] che mette a confronto 15 sistemi di dimostrazione assistita sotto diversi punti di vista e, in particolare, in rapporto alla formalizzazione di un semplice risultato matematico: l'irrazionalità di $\sqrt{2}$.

- La consistenza relativa dell'assioma della scelta in teoria degli insiemi [38].
- La correttezza dell'algoritmo di Buchberger [44]. Da questa formalizzazione è stato derivato automaticamente (in gergo si dice *«estratto»*) un programma per il calcolo delle basi di Gröbner. Un software creato in questo modo si dice *«certificato»*, dato che nasce corredato di una dimostrazione formale della sua correttezza controllata da un computer. In questo caso, il programma ottenuto in questo modo risulta essere relativamente efficiente.
- Una dimostrazione intuizionista del teorema fondamentale dell'algebra [16, 17]. In linea di principio, anche da questa formalizzazione, così come nel caso della precedente, può essere estratto in un programma certificato per il calcolo delle radici di un polinomio a coefficienti complessi. Tuttavia, per le sue dimensioni proibitive, non è stato ancora possibile eseguire su di un esempio concreto il programma ottenuto: uno studio ad hoc sull'ottimizzazione del codice generato automaticamente da una dimostrazione formale è tuttora in corso.
- Esistono diverse librerie di analisi di base reale e complessa. Alcune di queste contengono anche risultati avanzati di analisi funzionale. Citiamo tra queste una formalizzazione dell'intero libro di Landau [26] che risale a quasi 30 anni fa [23].
- Una libreria di analisi non standard che è stata utilizzata anche per eseguire un interessante studio sul rigore matematico dei Principia Matematica di Newton [12].
- È tuttora in corso un progetto di ricerca per sviluppare un intero sistema di dimostrazione assistita specializzato per la topologia algebrica e l'algebra omologica [3].

- Il teorema di Sylow sui gruppi finiti [24].
- Il primo teorema di Gödel [32].
- La teoria delle categorie [20, 34], [34].
- Il teorema di Church-Rosser di confluenza del lambda-calcolo [30].

Una menzione a parte è necessaria a proposito della libreria di Mizar [29] che è di gran lunga la più sviluppata fra quelle disponibili. Mizar è un sistema di dimostrazione assistita in uso dalla metà degli anni settanta e tuttora attivamente sviluppato. Dal 1989 i risultati più significativi verificati con questo software vengono regolarmente pubblicati sul «Journal of Formalized Mathematics» [22], una rivista disponibile sia in formato elettronico che cartaceo. Invitiamo il lettore a visitare il sito web di Mizar per avere un'idea dell'incredibile mole di lavoro svolta intorno a questo sistema.

Non mancano neppure esempi interessanti nei quali la scoperta di nuovi teoremi è stata possibile grazie alla dimostrazione automatica. Limitandosi ad indicare alcuni esempi di risultati ottenuti con il sistema Otter possiamo citare:

- La dimostrazione che XCB costituisce un singolo assioma per la logica equivalenziale [46].
- Alcuni risultati originali sulle curve cubiche, come l'unicità della legge di Steiner quinquennaria su di una curva cubica [28].
- La dimostrazione che le algebre di Robbins sono booleane [27].

Come caso di sintesi tra dimostrazione automatica e verifica automatica citiamo il progetto di formalizzazione del Teorema dei Quattro Colori nel sistema Coq da parte di Gonthier ([6]), che si è recentemente concluso in maniera positiva dopo cinque anni di lavoro.

Un altro progetto di lungo periodo riguarda la formalizzazione della dimostrazione della congettura di Keplero (nota anche come diciottesimo problema di Hilbert) a proposito della maggiore densità possibile con la quale è possibile impacchettare delle sfere nello spazio. Una dimostrazione di questa congettura è stata fornita da Hales nel '98 [33]. L'articolo contenente la dimostrazione avrebbe dovuto essere pubblicato sulla prestigiosa rivista Annals of Mathematics, ma, dopo

([6]) http://research.microsoft.com//~gonthier/

quattro anni di lavoro, i referee della rivista hanno dichiarato di non essere capaci di assicurare completamente la correttezza del lavoro a causa dell'enorme quantità di dettagli contenuti nella dimostrazione. Per fugare ogni dubbio sulla correttezza del suo lavoro, Hales ha deciso di intraprendere la via della matematica formalizzata, stimando che saranno necessari 20 anni per completare la dimostrazione formale al computer. Maggiori dettagli sulla storia, piuttosto interessante, di questo progetto possono essere letti su un articolo del New York Times [8] e sul web del progetto «Flyspeck» [18]. Un altro articolo che cita sia questo progetto che quello di Gonthier sul teorema dei quattro colori è apparso recentemente su *The Economist* [39].

3.3. *Insegnamento.*

Fin dalle loro origini i calcolatori hanno avuto un impiego crescente nella didattica della matematica. Con la verifica automatica delle dimostrazioni, si prospetta un nuovo modo di utilizzare il calcolatore per l'insegnamento della matematica, perlomeno per le scuole medie superiori e l'università, che potrà affiancarsi ad altri strumenti già oggi in uso come i sistemi di computer algebra o lo studio di algoritmi notevoli e dei linguaggi di programmazione.

Scrivere una dimostrazione al computer costituisce, di per sé, un esercizio assai formativo che può certamente rivelarsi in qualche modo «illuminante» anche per un matematico professionista. Il programma di dimostrazione assistita diventa, in modo naturale, una sorta di «esercizio interattivo», durante il quale viene controllato che ogni passaggio logico formulato dall'utente sia corretto e che il ragionamento conduca effettivamente alla soluzione del problema proposto. Per uno studente che deve assimilare le regole fondamentali del rigore matematico, il computer può giocare il ruolo di un arbitro affidabile, sempre disponibile, per mettere alla prova le proprie idee e la capacità di formularle correttamente all'interno di un certo sistema formale. Ogni passaggio non adeguatamente motivato si concretizza automaticamente per lo studente sotto forma di un preciso errore segnalato dal computer.

Anche nell'ambito della valutazione normativa, un ambiente di dimostrazione assistita può essere un nuovo strumento per realizzare questionari interattivi con domande a risposta aperta nelle quali viene chiesto di fornire la soluzione di un problema con la relativa giustificazione (dimostrazione).

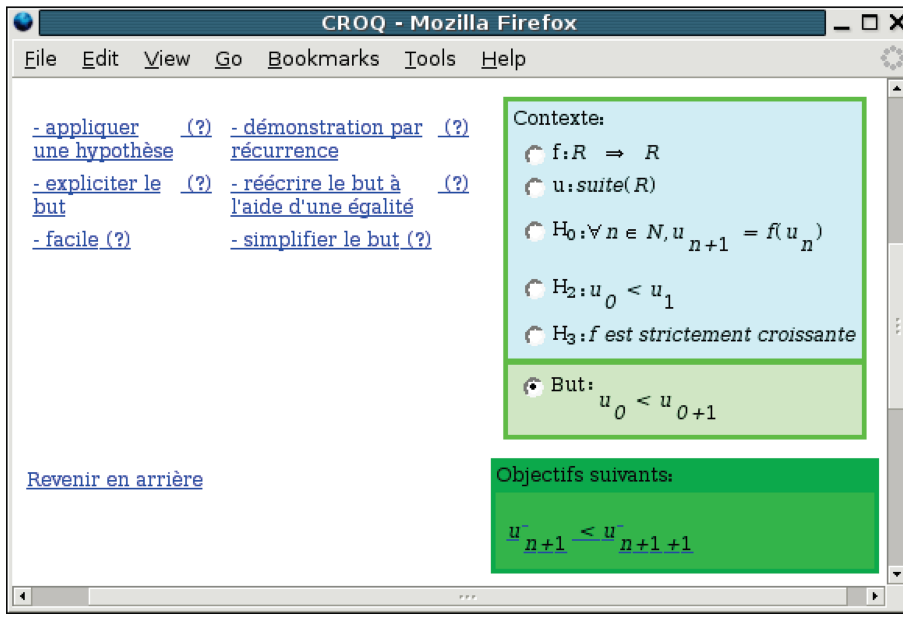

Fig. 14. – Una tappa dello svolgimento di un esercizio Croq.

A questo proposito segnaliamo Croq, un esempio di verifica automatica applicata alla creazione di esercizi interattivi di dimostrazione al computer sviluppato all'Università di Nizza. Si tratta di una estensione sperimentale di WIMS [15] basata sul sistema di dimostrazione assistita Coq [10]. Nel suo uso basilare, il sistema fornisce una interfaccia web per manipolare il contesto di una dimostrazione. La figura 14 contiene una porzione di una schermata Croq durante lo svolgimento della dimostrazione del seguente enunciato:

Se u è una successione tale che $u_0 < u_1$ e $u_{n+1} = f(u_n)$ per ogni naturale n, dove $f\colon \mathbb{R} \to \mathbb{R}$ è una funzione strettamente crescente, allora u è strettamente crescente.

Croq permette di applicare le regole di inferenza più comuni direttamente con il mouse, scegliendo una delle tattiche sulla pulsantiera che si vede a sinistra in figura oppure cliccando su parti «sensibili» di una formula. A destra viene visualizzato, ad ogni passaggio, il «contesto» della dimostrazione con la lista delle ipotesi disponibili e la tesi da dimostrare ed eventuali altre parti della dimostrazione rimaste in sospeso.

Un'altra esperienza di questo tipo è quella portata avanti da Raffalli [41] all'Università della Savoia. Si tratta di un corso di logica per studenti di matematica del terzo e quarto anno, durante il quale viene proposto di scrivere la formalizzazione alcuni teoremi con il sistema PhoX, un software di dimostrazione assistita particolarmente curato per essere impiegato nell'insegnamento, realizzato dallo stesso Raffalli [40].

Terminiamo questa sezione richiamando l'attenzione sul fatto che, a differenza della *verifica automatica* delle dimostrazioni, la *dimostrazione automatica* non costituisce un tema completamente nuovo nell'insegnamento della matematica, neanche a livello preuniversitario. Ad esempio, citiamo l'uso della teoria dell'eliminazione o quella delle basi di Groebner come strumento per derivare, in maniera simbolica, certi teoremi di geometria euclidea. Oppure lo studio di alcuni linguaggi di programmazione logici o dichiarativi, dei quali il Prolog è probabilmente il rappresentante più noto, nei quali l'idea della dimostrazione automatica interviene a vari livelli ed è addirittura essenziale nella comprensione del principio di funzionamento del linguaggio di programmazione stesso.

4. – Organizzazione della ricerca e del sapere matematico.

La possibilità di formalizzare delle dimostrazioni al computer promette di avere un impatto sul modo in cui la comunità dei matematici organizzerà la propria attività di ricerca.

Nella misura in cui diventerà ammissibile domandare agli autori di un lavoro di scrivere le loro dimostrazioni in maniera che i dettagli possano essere verificati al computer, sarà inevitabile un cambiamento del ruolo del referee nel processo di pubblicazione di un nuovo articolo

su di una rivista di matematica. È possibile che a quel punto le riviste cominceranno ad apprezzare sempre di più la possibilità di pubblicare articoli per i quali la correttezza non è più in alcun modo messa in discussione (un'ipotesi che in informatica teorica viene proposta con sempre più insistenza [4]).

Ovviamente, anche quando ogni lavoro matematico venisse *certificato*, rimarrebbero aperte altre questioni, come l'interesse matematico, l'originalità e la difficoltà del contenuto di un certo lavoro. La funzione del referee non potrà certo scomparire, ma potrebbe cambiare considerevolmente.

Possiamo anche immaginare riviste elettroniche nelle quali sia possibile contribuire via Internet in maniera completamente automatizzata delle teorie formalizzata al computer. Non sarebbe un meccanismo poi così diverso dai vari servizi di prepubblicazione online già oggi disponibili, ma con un importante vantaggio: tutto ciò che verrebbe diffuso con questo mezzo sarebbe certamente affidabile dal punto di vista della correttezza matematica.

L'apprendimento della matematica è un altro processo nel quale potranno verificarsi dei sostanziali cambiamenti. Uno dei modi più naturali per apprendere una teoria matematica è quello di scrivere delle note, sforzandosi di esplicitare con cura i dettagli che sono spiegati solo in maniera generale nella letteratura. Questo potrebbe cambiare in due modi. Nel caso in cui nessun contributo matematico formalizzato al computer sia disponibile per una data teoria, uno studente potrebbe essere condotto a scriverne uno lui stesso. D'altro canto, per argomenti già formalizzati al computer in uno o più modi, l'idea di scrivere delle note dettagliate diventerebbe certamente meno attraente e uno studente potrebbe trovare altre vie per integrare il materiale, in modi che forse non possiamo adesso prevedere.

Un possibile effetto collaterale negativo legato al poter controllare delle dimostrazioni formali potrebbe essere quello di ridurre il numero di errori che vengono resi pubblici: la matematica «sbagliata» può essere talvolta assai interessante. Per fare un esempio estremo, possiamo citare il caso della dimostrazione dell'Ultimo Teorema di Fermat Andrew Wiles e Richard Taylor. Com'è noto, la dimostrazione originaria di Wiles contiene un errore che è stato trovato successivamente

alla sua diffusione, che, in seguito, il contributo di Richard Taylor ha permesso di correggere. Se Wiles avesse usato un sistema di dimostrazione assistita, avrebbe potuto accorgersi dell'errore lui stesso e non avrebbe molto probabilmente annunciato la dimostrazione incompleta. Sarebbe riuscito ad aggiustare il ragionamento da solo? Non possiamo saperlo, ma esiste la possibilità che, se la dimostrazione formale al computer fosse già una pratica diffusa, non avremmo ancora avuto una dimostrazione del teorema. Resta il fatto che gli errori matematici possono avere talvolta un effetto positivo se comunicati agli altri. Con l'avvento della verifica al computer, l'idea di comunicare dimostrazioni incomplete o sbagliate potrebbe diventare inaccettabile. Ma siamo ancora certamente lontani dall'epoca in cui questo potrebbe davvero verificarsi.

5. – Tra 10 anni.

Ovviamente non possiamo fare nessuna previsione attendibile su quello che potrà essere lo sviluppo di questa disciplina che solo pochi decenni fa non esisteva ancora. Possiamo tuttavia cercare di fare una semplice «proiezione» nel futuro delle tendenze più palesi che si possono cogliere oggi, per immaginare, se pure in via del tutto ipotetica, quale potrebbe essere la situazione, diciamo, tra 10 anni.

5.1. *Potenza di calcolo.*

La prima direzione ovvia di estrapolazione è quella della velocità computazionale. Se diamo per valida la «legge di Moore» secondo la quale la velocità computazionale raddoppia ogni 18 mesi, possiamo aspettarci che in 10 anni le risorse computazionali disponibili per questo tipo di applicazioni saranno aumentate di un fattore 100. Per un dimostratore di teoremi puro, questo difficilmente apporterà un così grande miglioramento, dato che la maggior parte degli algoritmi sono tendenzialmente esponenziali: potrebbe soltanto permetterci di trovare con il calcolo bruto delle dimostrazioni di appena qualche passo più lunghe rispetto a quelle che possiamo fare oggi. D'altra parte,

potrebbe aumentare significativamente la grandezza delle basi di dati utilizzati nel corso della ricerca o della verifica di una dimostrazione, dato che una base di dati correttamente progettata può effettuare le operazioni di accesso con complessità logaritmica nella grandezza dell'archivio.

Una buona regola approssimativa per un ambiente interattivo di dimostrazione assistita sembra essere la seguente: il tempo impiegato dal computer per accettare un passo di una dimostrazione dovrebbe essere dell'ordine o inferiore a quello necessario ad un utente per concepirlo ed inserirlo. Le procedure che non soddisfano questa regola tendono ad essere inutilizzate e rimpiazzate con tattiche più semplici e veloci, anche se meno potenti. Dato che l'efficacia da parte dell'utente rimarrà probabilmente invariata, un aumento di 100 volte della velocità da parte del computer implica che la grandezza matematica di ogni passo di dimostrazione effettuato dalle tattiche automatiche potrebbe aumentare in maniera piuttosto significativa.

5.2. *Librerie.*

Con il progressivo sviluppo di questa disciplina, i sistemi di dimostrazione assistita stanno diventando sempre più attraenti. Sviluppare teoremi al computer diventerà sempre più semplice e crescerà il rapporto tra il beneficio della garanzia di correttezza e il costo della formalizzazione.

Non è impensabile che nei prossimi anni una parte non trascurabile di matematici contribuirà a qualcuno dei molti progetti già iniziati o ne comincierà uno proprio. In 10 anni, è possibile che questo conduca ad un aumento significativo della quantità complessiva di teoremi formalizzati al computer e, con loro, l'ampiezza della librerie di base.

Tuttavia, che l'aumento della quantità di librerie disponibili si debba tradurre direttamente in maggiore facilità d'uso della dimostrazione assistita, è una questione differente.

Certamente un problema che si porrà con gravità sarà la questione della compatibilità tra i contributi disponibili in diversi ambienti di dimostrazione assistita. Abbiamo l'impressione che sia improbabile

che questo problema possa trovare una soluzione efficace nei prossimi 10 anni. Dovremo probabilmente aspettare più a lungo.

È possibile che prima o poi venga realizzato un nuovo programma che riuscirà ad imporsi come standard, un po' come è avvenuto con il TEX per la formattazione di testi matematici. Per il momento non ci sono indicazioni che ciò stia per avvenire, ma questo non ne esclude tuttavia la possibilità.

5.3. *Ragionare per diagrammi.*

Se pure abbiamo assitito a progressi enormi nelle tecniche per la formalizzazione e la soluzione automatica di problemi di tipo simbolico e algebrico, la ricerca sul modo di rappresentare ragionamenti matematici di tipo visivo o geometrico in una macchina sembra molto meno avanzata. Certe aree della matematica fanno un ampio uso di disegni schematici e diagrammi, come ad esempio in teoria delle categorie. Sarà difficile fare matematica al computer nel modo nel quale siamo adesso abituati fino a quando non ci sarà la flessibilità di codificare e manipolare certe informazioni simboliche in modo visivo come nella pratica attuale con carta e penna. Progetti in questa direzione potrebbero certamente avere inizio nei prossimi anni, ma ci sembra improbabile che essi possano raggiungere in breve tempo dei risultati maturi in 10 anni.

5.4. *Sviluppo di nuove teorie matematiche al computer.*

Fino ad oggi, la matematica pura che è stata formalizzata al computer ha riguardato quasi esclusivamente risultati già noti da tempo e non sembrano ancora esserci esempi notevoli di teorie matematiche nate e sviluppate con l'aiuto di un sistema di dimostrazione assistita. Sembra probabile che entro 10 anni, qualche matematico possa produrre nuove teorie significative con l'aiuto di un sistema di dimostrazione assistita, se pure non necessariamente nell'ambito stretto della matematica classica. Abbiamo l'impressione che una volta che questa strada sarà aperta da un esempio guida, altri ricercatori sa-

ranno spinti a seguire la stessa direzione e aumenterà la possibilità che dei matematici decidano di utilizzare un sistema di verifica automatica come supporto nel corso di un progetto di ricerca, per avere la garanzia della validità delle nuove teorie.

RIFERIMENTI BIBLIOGRAFICI

[1] AA.VV. The QED manifesto. In *Proceedings of the 12th International Conference on Automated Deduction* (Springer-Verlag, 1994), 238-251.

[2] The Archive of Formal Proofs. Url: http://afp.sourceforge.net/.

[3] Jesús Aransay - Clemens Ballarin - Julio Rubio, *Towards a higher reasoning level in formalized Homological Algebra*. In *Thérèse Hardin and Renaud Rioboo, editors,11th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning (Calculemus)* (Rome, Italy, 2003), 84-88. Aracne Editrice.

[4] Brian E. Aydemir - Aaron Bohannon - Matthew Fairbairn - J. Nathan Foster - Benjamin C. Pierce - Peter Sewell - Dimitrios Vytiniotis - Geoffrey Washburn - Stephanie Weirich - Steve Zdancewic, *Mechanized metatheory for the masses: The PoplMark challenge*. In *International Conference on Theorem Proving in Higher Order Logics (TPHOLs),* Lecture Notes in Computer Science. Springer Verlag, August 2005. Url: http://fling-l.seas.upenn.edu/~plclub/cgi-bin/poplmark/.

[5] Yves Bertot - Gilles Kahn - Laurent Théry, *Proof by pointing*. In *Theoretical aspects of computer software (Sendai, 1994),* volume 789 of *Lecture Notes in Comput. Sci.* (Springer, Berlin, 1994), 141--160.

[6] Alan Bundy, *The Computer Modelling of Mathematical Reasoning.* Academic Press, 1983. Trad. it. [7].

[7] Alan Bundy, *L'automazione del ragionamento matematico: dalla dimostrazione dei teoremi alla formazione dei concetti.* Muzzio, Padova, 1986. Edizione italiana di [6]. Traduzione a cura di Mauro Boscarol.

[8] Kenneth Chang, In *Math, Computers Don't Lie. Or Do They?* New York Times, April 6, 2004.

[9] Alonzo Church, *A formulation of the simple theory of types*. Journal of Symbolic Logic, **5** (1940), 56-68.

[10] *The Coq proof assistant*. Url: http://coq.inria.fr/.

[11] J. D. Fleuriot, *A Combination of Geometry Theorem Proving and Nonstandard Analysis, with Application to Newton's Principia.* (Springer-Verlag, 2001).

[12] Jacques D. Fleuriot - Lawrence C. Paulson, *Proving Newton's Propositio Kepleriana using geometry and nonstandard analysis in Isabelle*. In *Automated deduction in geometry (Beijing, 1998),* volume 1669 of *Lecture Notes in Comput. Sci.* (Springer, Berlin, 1999), 47-66.

[13] Jacques D. Fleuriot - Lawrence C. Paulson, *Mechanizing nonstandard real analysis*. LMS J. Comput. Math., **3** (electronic) (2000), 140-190.

[14] *Formal methods web page at Oxford*. Url: http://www.afm.sbu.ac.uk/.

[15] Xiao Gang, *WWW Interactive Multipurpose Server*. Url: http://wims.unice.fr/.

[16] Freek Geuvers - Jan Herman - Randy Wiedijk - Henk Zwanenburg - Barendregt Pollack, *FTA: Fundamental Theorem of Algebra Project*. Url: http://www.cs.kun.nl/~freek/fta/.

[17] HERMAN GEUVERS - FREEK WIEDIJK - JAN ZWANENBURG, *A constructive proof of the Fundamental Theorem of Algebra without using the rationals*. In *TYPES*, 2000, 96-111.
[18] THOMAS HALES, *The flyspeck project*. Url: http://www.math.pitt.edu/~thales/flyspeck/.
[19] JOHN HARRISON, *Formalized mathematics*. Technical Report 36, Turku Centre for Computer Science (TUCS), Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland, 1996. Url: http://www.cl.cam.ac.uk/users/jrh/papers/form-math3.html.
[20] GÉRARD HUET - AMOKRANE SAÏBI, *Constructive category theory*. In *Proof, language, and interaction*, Found. Comput. Ser., MIT Press, Cambridge, MA, 2000, 239-275.
[21] ISABELLE, Url: http://www.cl.cam.ac.uk/Research/HVG/Isabelle/.
[22] *Journal of Formalized Mathematics*. Url: http://mizar.org/JFM/.
[23] L.S. VAN BENTHEM JUTTING, *Checking Landau's «Grundlagen» in the AUTOMATH System*. PhD thesis, Eindhoven University of Technology, 1977.
[24] FLORIAN KAMMÜLLER - LAWRENCE C. PAULSON, *A formal proof of Sylow's theorem*. An experiment in abstract algebra with Isabelle HOL. *J. Automat. Reason.*, 23(3-4) (1999), 235-264.
[25] D. E. KNUTH - P. B. BENDIX, *Simple word problems in universal algebras*. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970* (Springer, Berlin, Heidelberg, 1983), 342-376.
[26] EDMUND LANDAU, *Grundlagen der Analysis*. Akademische Verlagsgesellschaft, Leipzig, Germany, 1930. English translation *Foundations of Analysis*, Chelsea Publishing Company, 1951.
[27] W. MCCUNE, *Solution of the Robbins problem*. Journal of Automated Reasoning (1997), 263-276. Url: http://www-unix.mcs.anl.gov/~mccune/papers/robbins/.
[28] W. MCCUNE - R. PADMANABHAN, *Automated deduction in equational logic and cubic curves*, volume 1095 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1996. Lecture Notes in Artificial Intelligence.
[29] *Mizar system*. Url: http://mizar.uw.bialystok.pl/.
[30] TOBIAS NIPKOW, *More Church-Rosser proofs* (in Isabelle/HOL). In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction (CADE-13)*, pp. 733--747, New Brunswick, New Jersey, July 1996. Springer-Verlag LNAI 1104.
[31] TOBIAS NIPKOW - LAWRENCE C. PAULSON - MARKUS WENZEL, *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
[32] RUSSELL O'CONNOR, *Proof of Gödel's first incompleteness theorem in Coq*. Url: http://math.berkeley.edu/~roconnor/godel.html.
[33] JOSEPH OESTERLÉ, *Densité maximale des empilements de sphéres en dimension 3 (d'après Thomas C. Hales et Samuel P. Ferguson)*. Astérisque, (266):Exp. No. 863, 5, 405-413, 2000. Séminaire Bourbaki, Vol. 1998/99.
[34] GREG O'KEEFE, Towards a readable formalisation of category theory. *Electronic Notes in Theoretical Computer Science*, 91:212--228, February 2004.
[35] OTTER, *An automated deduction system*. Url: http://www-unix.mcs.anl.gov/AR/otter/.
[36] R. PADMANABHAN - W. MCCUNE, *An Equational Characterization of the Conic Construction on Cubic Curves*. Number 1095 in Lecture Notes in Computer Science (AI subseries). Springer-Verlag, 1996.
[37] WILLIAM PADMANABHAN - R. MCCUNE, *Automated Deduction in Equational Logic and Cubic Curves* (Springer Verlag, 1996).
[38] LAWRENCE C. PAULSON, *The relative consistency of the axiom of choice mechanized using Isabelle/ZF*. LMS J. Comput. Math., 6 (electronic) (2003), 198-248. Appendix A available electronically at http://www.lms.ac.uk/jcm/6/lms2003-001/appendix-a/.

[39] Proof and beauty. *The Economist*, marzo 2005.
[40] C. RAFFALLI, *The PhoX proof assistant*. Url: http://www.lama.univ-savoie.fr/~raffalli/phox.html.
[41] CHRISTOPHE RAFFALLI - RENÉ DAVID, *Computer assisted teaching in mathematics*. In *Workshop on 35 years of Automath (Avril 2002, Edingurgh)*, 2002.
[42] CARLOS T. SIMPSON, *Computer theorem proving in math*, 2003. arXiv:math.HO/0311260.
[43] CARLOS T. SIMPSON, *Set-theoretical mathematics in Coq*, 2004. arXiv:math.LO/0402336.
[44] LAURENT THÉRY, *A machine-checked implementation of Buchberger's algorithm*. J. Automat. Reason., 26(2) (2001), 107-137.
[45] FREEK WIEDIJK, *Comparing mathematical provers*. In Andrea Asperti, Bruno Buchberger, and James Davenport, editors, *Mathematical Knowledge Management*, number 2594 in Lecture Notes on Computer Science, pp 188-202. Springer, 2003. Proceedings of MKM 2003.
[46] LARRY WOS - DOLPH ULRICH - BRANDEN FITELSON, *Vanquishing the XCB Question: The Methodological Discovery of the Last Shortest Single Axiom for the Equivalential Calculus*. Journal of Automated Reasoning, **29**(2) (2002), 107-124.

Marco Maggesi: Dipartimento di Matematica «U. Dini»
Università degli studi di Firenze

Carlos Simpson: Laboratoire J. A. Dieudonné, CNRS UMR 6621
Université de Nice-Sophia Antipolis