UNIVERSITÀ DEGLI STUDI DI TORINO

DIPARTIMENTO DI INFORMATICA

SCUOLA DI SCIENZE DELLA NATURA

Corso di Laurea in Informatica

Tesi di Laurea Triennale

Introduzione di O-grande in Isabelle

Relatore: prof. Stefano BERARDI

Candidato: Lorenzo SCIANDRA

ANNO ACCADEMICO
2019/2020

DICHIARAZIONE DI ORIGINALITÀ

Dichiaro di essere responsabile del contenuto dell'elaborato che presento al fine del conseguimento del titolo, di non avere plagiato in tutto o in parte il lavoro prodotto da altri e di aver citato le fonti originali in modo congruente alle normative vigenti in materia di plagio e di diritto d'autore. Sono inoltre consapevole che nel caso la mia dichiarazione risultasse mendace, potrei incorrere nelle sanzioni previste dalla legge e la mia ammissione alla prova finale potrebbe essere negata.

Indice

Introduzione

Nel 1986 venne rilasciata, sotto la direzione di Larry Paulson dell'Università di Cambridge e Tobias Nipkow della TU di Monaco, la prima versione di Isabelle, un dimostratore di teoremi interattivo che utilizza logiche di ordine superiore.
Vent'anni dopo circa, il primo gennaio del 2004, Jeremy Avigad e Kevin Donnelly della Carnegie Mellon University pubblicano l'articolo *"Formalizing O notation in Isabelle"* con l'intento di proporre una possibile introduzione della notazione asintotica O-grande in Isabelle.
Le motivazioni alla base della decisione di teorizzare una formalizzazione Isabelle delle nozioni matematiche che riguardano O-grande, risiedono nelle vaste applicazioni che una tale notazione asintotica possiede in ambito informatico per lo studio sui tempi di calcolo degli algoritmi.
La difficoltà insita in questo obiettivo è strettamente correlata alle molteplici ripercussioni che un tale sviluppo potrebbe comportare. Anche gli svariati anni di continui perfezionamenti e modifiche che seguirono questa prima versione ne sono da testimoni.
Lo scopo della tesi è analizzare l'idea originaria e mostrare la sua evoluzione fino alla formulazione che ad oggi è contenuta nella libreria Isabelle.
La metodologia scelta per redigere questa trattazione si basa sulla decisione di presentare una prima introduzione teorica dei temi interessati, per poi effettuare una graduale transizione verso il codice Isabelle, che darà forma e sostanza all'idea di Avigad e Donnelly. Partendo quindi proprio dall'articolo del 2004 saranno ampliate le nozioni di base lì presentate e, utilizzando l'applicativo aggiornato all'ultima versione del 2020, saranno esposte e dimostrate le proprietà principali.

Vediamo ora un accenno degli argomenti discussi nei vari capitoli.

Il primo capitolo sarà dedicato allo studio delle nozioni fondamentali, gli strumenti essenziali, che permetteranno al lettore di orientarsi meglio in quello che è

il contesto che gravita attorno alla trattazione. Concetti analitici come l'asintoto e le sue applicazioni nel mondo algoritmico, strutture algebriche come l'anello e le proprietà di cui gode e persino informazioni preliminari relative ad Isabelle potranno risultare fondamentali per una maggiore comprensione per i non addetti ai lavori.

Nel secondo capitolo si evidenzierà invece la formalizzazione vera e propria proposta per O-grande, dando inizio alla trasformazione che porterà la tesi da una prevalenza di argomenti teorici, distintivi del primo capitolo, all'implementazione in codice che caratterizzerà invece il terzo.

Nell'ultimo capitolo, forti dei concetti presentati nelle sezioni precedenti, si analizzeranno le proprietà base che una tale formalizzazione deve possedere e, trasponendole in Isabelle, ne sarà verificata la loro correttezza.

La tesi termina con la conclusione finale, nella quale verranno mostrati sia i risvolti futuri che un tale lavoro implica sia alcuni suggerimenti su come si possa migliorare la comprensione delle teorie Isabelle. Come risulterà chiaro con la lettura della tesi, spesso le dimostrazioni dei lemmi possono risultare difficili da capire senza l'utilizzo dell'applicativo che ne espliciti i passaggi richiesti.
Quella che si proporrà quindi nella conclusione sarà una possibile idea su come si possano riformulare i lemmi Isabelle rendondoli estremamente più facili da comprendere.

Capitolo 1

Concetti Fondamentali

Senza pretesa alcuna di fornire una presentazione esaustiva degli argomenti, in questo primo capitolo si espongono semplicemente quelle nozioni di appoggio che saranno spesso richiamate nei capitoli successivi. Per una trattazione completa degli argomenti si rimanda il lettore rispettivamente a [1] testo di riferimento per la notazione asintotica in ambito algoritmico, [2] per i costrutti algebrici e [3] per l'introduzione a Isabelle.

1.1 Notazione Asintotica

Le risposte esatte affascinano, la perfezione che si cela in una incontestabile verità è fonte di estrema soddisfazione, ma esiste un tempo in cui c'è il bisogno di approssimare. Se ci imbattiamo infatti in una sommatoria o in una ricorrenza priva di una forma chiusa, non siamo in grado di trovare la soluzione perfetta, ma vorremmo comunque sapere qualcosa riguardo alla possibile risposta.
Il termine "asintoto" deriva dal Greco *asýmptōtos* e significa "che non s' incontra". Quando gli antichi matematici greci studiarono le sezione coniche introdussero questo termine in corrispondenza delle rette, chiamate appunto asintoti, che definiscono le iperboli. Il grafo della funzione $y = \sqrt{(1+x^2)}$ rappresenta ad esempio un'iperbole e le rette y = x e y = -x sono i suoi asintoti. La curva si avvicina sempre più alle rette, ma non le toccherà mai.
La notazione asintotica trova impiego oggigiorno in numerose branche della matematica e dell'informatica teorica. In quest'ultima in particolare, viene usata per analizzare il tasso di crescita del costo della computazione di un algoritmo, dandone una buona stima della sua efficienza. La funzione che tiene conto del costo

di un algoritmo può essere definita in modo da considerare il tempo, lo spazio, o qualsiasi altra risorsa impiegata. Quando operiamo con algoritmi diversi risulta spesso necessario doverli confrontare per determinare quali tra i due si possa considerare in via teorica il più efficiente. Un algoritmo che possiede una funzione dei costi asintoticamente più efficiente di un altro, sarà generalmente sempre migliore per qualsiasi input, fatta eccezione di quelli di piccole dimensioni.
In questa trattazione sulla notazione asintotica ci concentreremo solo sul significato di O-grande, l'unica di nostro interesse, tralasciando quindi i concetti di o-piccolo, Θ, Ω e ω, che indicano rispettivamente un limite asintotico superiore stretto, un limite asintoticamente stretto, un limite asintotico inferiore e un limite asintotico inferiore stretto.
La notazione O-grande fu introdotta da Paul Bachmann nel 1894 e resa popolare negli anni a seguire grazie al contributo di Edmund Landau e altri matematici. La caratteristica fondamentale di questa notazione è la capacità di sopprimere i dettagli non importanti per permetterci di concentrarci solo sugli aspetti più indicativi. Con la notazione O-grande si fa riferimento ad un limite asintotico superiore di una funzione. Data $g(n)$, indichiamo con $O(g(n))$ l'insieme delle funzioni:

$$O(g(x)) = \{f(x) \mid \exists\, c > 0, x_0 > 0 \text{ costanti t.c. } 0 \leq f(x) \leq cg(x)\ \forall x \geq x_0\}$$

In altre parole è l'insieme di tutte le funzioni $f(x)$ tale per cui $\forall x \geq x_0$ la funzione $f(x)$ coincide o è minore di $g(x)$ moltiplicata per una costante $c > 0$. La definizione qua presentata va a limitare il concetto analitico di O-grande andando a considerare solo $f(x)$ e $g(x)$ funzioni asintoticamente non negative, ovvero non negative per valori grandi di x. Tutto questo in accordo con un punto di vista puramente algoritmico che considera funzioni positive dei costi, generalmente direttamente proporzionali con l'input, in modo che non si possa verificare un costo negativo di tempo o spazio per un qualsiasi valore in input.
Quando si confrontano due funzioni $f(x)$ e $g(x)$ si assume che abbiano lo stesso dominio e codominio e che la nozione di moltiplicazione sia definita nei loro codomini. Questo può essere ottenuto facilmente trattando funzioni definite su $\mathbb{R}$, anello ordinato non degenere, senza limitare la generalità della trattazione dato che le funzioni che rappresentano i costi computazionali rientrano pienamente in tale restrizione.
Poichè la notazione O-grande descrive un limite superiore, può essere correttamente usata per l'analisi dei casi pessimi di esecuzione di un algoritmo, indicando un limite massimo delle risorse impiegate.
Da notare che, nonostante $O(g(n))$ rappresenti un insieme di funzioni, si usi l'espressione $f(n) = O(g(n))$ per indicare che la funzione f sia un membro del-

l'insieme considerato. Coerentemente con una interpretazione insiemistica del simbolo di uguaglianza in un'espressione contenente un simbolo di O-grande, l'asserzione

$$x^2 + O(x) = O(x^3)$$

risulterà corretta, mentre l'asserzione simmetrica

$$O(x^3) = x^2 + O(x)$$

no. In particolare una qualsiasi somma del tipo $f + O(g)$ può essere vista come somma insiemistica

$$f + O(g) = \{f\} + O(g) = \{f + l \mid l \in O(g)\},$$

in cui f viene considerato come l'insieme contenente come unico elemento se stesso. Di conseguenza questo permette di interpretare l'espressione

$$f + O(g) = O(h)$$

come una relazione di inclusione $f + O(g) \subseteq O(h)$.
In letteratura esistono ulteriori costrutti per separare quei casi in cui la funzione $f(x)$ è sempre $\leq$ a $g(x)$, da quelli in cui per valori molto piccoli di x, può capitare che $f(x)$ sia $>$ di $g(x)$. In particolare indicato con A l'insieme delle funzioni definite da δ a τ e S l'insieme di elementi di tipo δ, si definisce "$A\,on\,S$" l'insieme

$$\{f \mid \exists g \in A\, \forall x \in S\, (f(x) = g(x))\}$$

di tutte le funzioni che concordano con alcune funzioni in A sugli elementi di S. Mentre si denota con "$A\,eventually$" l'insieme

$$\{f \mid \exists k\, \exists g \in A\, \forall x \geq k\, (f(x) = g(x))\}$$

di tutte le funzioni che alla fine concordano con qualche funzione di A.
Ad esempio prese le funzioni $f(x) = x + 1$ e $g(x) = x^2$ dovrei dire che "$f = O(g)$ *eventually*" se considero $S = \mathbb{N}$, dato che per $x = 0$ $f(x) > g(x)$, mentre potrei dire "$f = O(g)\,on\,\mathbb{N}^+$" se prendo $S = \mathbb{N}^+$, escludendo quindi l'unico caso particolare dell'esempio.

1.2 L'anello e altre strutture algebriche

Dato un insieme A, si definisce l'anello $(A,+,*)$ come l'insieme A dotato di due operazioni binarie $+$ e $*$, chiamate generalmente addizione e moltiplicazione. A risulta inoltre chiuso rispetto alle sue due operazioni ovvero $\forall a,b \in A \;\; a*b \in A$ e $a+b \in A$. Le proprietà che contraddistinguono l'anello sono:

1. $+$ è associativa, ossia $(a+b)+c = a+(b+c) \quad \forall a,b,c \in A$;
2. $\exists\, e_1$ neutro rispetto alla $+$ t.c $\;\; a+e_1 = e_1+a = a \quad \forall a \in A$;
3. $\forall a \in A \;\; \exists a^{-1}$ t.c $\;\; a+a^{-1} = e_1$ (esistenza dell'inverso rispetto alla +);
4. $+$ è commutativa, ossia $a+b = b+a \quad \forall a,b \in A$;
5. $*$ è associativa, cioè $(a*b)*c = a*(b*c) \quad \forall a,b,c \in A$;
6. valgono le seguenti leggi distributive: $a*(b+c) = a*b+a*c$, $(a+b)*c = a*c+b*c \quad \forall a,b,c \in A$.

Se consideriamo invece un insieme B dotato di una singola operazione +, facendo riferimento alle proprietà appena introdotte, avremo un:

- Semigruppo se $(B,+)$ gode della proprietà 1. ;
- Monoide se $(B,+)$ gode delle proprietà 1. e 2. ;
- Gruppo se $(B,+)$ gode delle proprietà 1. , 2. e 3. .

Tutte queste strutture algebriche appena definite vengono ulteriormente definite come abeliane quando la loro operazione è commutativa e quindi dotata dalla proprietà (4). Con le informazioni appena introdotte possiamo affermare che se l'insieme A, prima considerato, formasse un monoide con la prima operazione (A,+) ed un semigruppo con la seconda (A,*), potremmo dire di star considerando un semianello nel caso in cui valesse la distributività della moltiplicazione rispetto all'addizione in aggiunta alla proprietà che $\forall a \in A \;\; a*0 = 0*a = 0$.
Faremo spesso riferimento nella trattazione al concetto di anello non degenere. Con anello non degenere indichiamo un anello non composto univocamente dall'elemento neutro o alternativamente, nel caso in cui fossero definiti entrambi i neutri, anelli in cui $e_1 \neq e_2$.
Un dominio di integrità è un anello commutativo privo di divisori dello zero.
Più precisamente $(A,+,*)$ è un dominio di integrità se valgono le seguenti proprietà:

- Commutatività del prodotto: $a * b = b * a \quad \forall a, b \in A$;
- Annullamento del prodotto: $a * b = 0 \implies a = 0 \vee b = 0$.

Denotiamo invece con anello unitario o con unità, un anello dotato di e_2, elemento neutro per la seconda operazione.
La presenza di quest'ultima proprietà insieme con la commutatività della moltiplicazione e l'esistenza dell'inverso moltiplicativo, per ogni elemento non nullo, definisce il cosiddetto campo.
Con l'espressione campo ordinato si fa riferimento ad un campo in cui valga una relazione d'ordine binaria sull'insieme A. In particolare una relazione ρ da un insieme A ad un insieme B, è un'associazione che lega elementi di A con elementi di B secondo un determinato criterio. Risulta quindi un sottoinsieme del prodotto cartesiano A x B che, nel caso in cui $A = B$ si denota semplicemente come relazione definita su A. Coerentemente con la definizione la sintassi corretta per esprimere che due elementi sono in relazione dovrebbe quindi essere $(a, b) \in \rho$, ma in realtà si usa scrivere semplicemente $a \rho b$. Una relazione viene detta d'ordine se verifica le seguenti proprietà:

- Riflessiva: $a \rho a \quad \forall a \in A$;
- Antisimmetrica: $a \rho b,\ b \rho a \implies a = b \quad \forall a, b \in A$;
- Transitiva: $a \rho b,\ b \rho c \implies a \rho c \quad \forall a, b, c \in A$.

Se su un qualsiasi insieme vale tale relazione allora è possibile ordinare l'insieme stesso che verrà di conseguenza denominato come ordinato. Un campo ordinato viene detto archimedeo se vale l'assioma di Archimede, ossia date due grandezze omogenee, esiste sempre un multiplo dell'una che supera l'altra :

$$\forall a, b \in A \ \ t.c \ \ 0 < a < b, \exists n \in \mathbb{N} \ b \leq (n * a)$$

Rispetto alla relazione appena presentata distinguiamo ρ_1 come relazione d'equivalenza in A_1 quando, invece della proprietà antisimmetrica, gode della proprietà simmetrica $(a \rho_1 b) \implies (b \rho_1 a) \ \forall a, b \in A_1$.
Grazie a queste prime definizioni saranno ora presentate una serie di dimostrazioni e risultati notevoli tratti sempre dal testo [2], che permetteranno una maggiore comprensione della costruzione dei reali presentata nel sottocapitolo 2.2 e utilizzata come possibile codominio delle funzioni trattate. Introduciamo adesso il

concetto di classe di equivalenza modulo ρ_1 di un elemento $a \in A_1$ come l'insieme di tutti gli elementi di A_1 che sono equivalenti ad a:

$$[a] = \{\, b \in A_1 \mid b \ \rho_1 \ a \,\}$$

Ad esempio presa la classica relazione '=' di uguaglianza definita su un qualsiasi insieme, le classi di equivalenza di ogni elemento saranno costituite da singletons, cioè risulteranno sottoinsiemi ridotti all'elemento stesso.
L'insieme quoziente di A_1 rispetto a ρ_1, denota invece l'insieme di tutte le classi di equivalenza modulo ρ_1:

$$A_1/\rho_1 = \{\, [a] \mid a \in A_1 \,\}$$

Utilizzando questo costrutto gli elementi tra loro equivalenti in A_1 si ritrovano all'interno di un unico elemento $[a]$ di A_1/ρ_1 e l'insieme di tutte le classi d'equivalenza formerà una partizione di A_1. Considerando un anello $(R,+,*)$, allora la relazione ρ_1 definita su R si dirà compatibile con le operazioni di R se $\forall a_1, a_2, b_1, b_2 \in R$:

$$a_1 \ \rho_1 \ a_2, \quad b_1 \ \rho_1 \ b_2 \quad \Longrightarrow \quad \begin{cases} (a_1 + b_1) \ \rho_1 \ (a_2 + b_2) \\ (a_1 * b_1) \ \rho_1 \ (a_2 * b_2) \end{cases}$$

Possiamo ora far vedere che assegnare in un anello R una relazione di equivalenza compatibile con entrambe le operazioni equivale ad assegnare un ideale bilatero. Il termine ideale di un anello R indica un sottogruppo additivo I tale che $\forall i \in I$ e $\forall r \in R$, $r * i \in I$. Da questo segue che ogni anello ha come ideali banali $\{0\}$ e se stesso. Si fa un'ulteriore distinzione tra ideale destro o sinistro a seconda dell'associazione dell'operazione tra r e i. Un ideale bilatero è invece un ideale destro e sinistro contemporaneamente e si denota come $I \trianglelefteq A$. Ritornando all'anello R e alla relazione di equivalenza ρ_1 prima considerati allora :

$$I = \{\, x \in R \mid x \ \rho_1 \ 0 \,\}$$

è un ideale bilatero di R.
Dimostrazione. Proveremo che $\forall x, y \in I$ e $\forall r \in R$, si ha $(x - y) \in I$ e $(x * r) \in I$, $(r * x) \in I$:

$$x \in I, \ y \in I \implies x \ \rho_1 \ 0, \ y \ \rho_1 \ 0 \implies \begin{cases} x \ \rho_1 \ y \\ -y \ \rho_1 - y \end{cases}$$

Da qui, mediante la compatibilità di ρ_1 rispetto all'addizione possiamo procedere affermando che:

$$x\ \rho_1\ y,\ -y\ \rho_1 - y \implies (x-y)\ \rho_1\ (y-y) = (x-y)\ \rho_1\ 0$$

e quindi $(x-y) \in I$. Allo stesso modo presi $x \in I$ e $r \in R$, possiamo dedurre per compatibilità rispetto alla moltiplicazione:

$$x\ \rho_1\ 0,\ r\ \rho_1\ r \implies (x*r)\ \rho_1\ (0*r) = (x*r)\ \rho_1\ 0$$

ossia $x*r \in I$ e analogamente si prova $r*x \in I$. □
Una relazione così definita prende anche il nome di congruenza modulo I. In accordo con la formalizzazione del campo dei reali, che sarà presentata nel sottocapitolo 2.1 introduciamo le ultime definizioni necessarie. Un ideale I di un anello R, con $I \neq R$, si dice massimale se

$$\forall U \trianglelefteq R \text{ tale che } I \subseteq U \subseteq R \implies U = I \text{ o } U = R$$

cioè non esistono ideali intermedi U tra I e R.
Con questo concetto possiamo concludere presentando le ultime due definizioni rilevanti con le annesse dimostrazioni. La prima afferma che un anello commutativo con unità è un campo se e solo se è privo di ideali non banali.
Dimostrazione. Sia R un campo e sia I un ideale non nullo di R, tale che $\exists i \neq 0,\ i \in I$. Dato che R è un campo esisterà l'inverso moltiplicativo di i, denotato come i^{-1}, ma allora $i*i^{-1} = 1 \in I$, il che implica che I conterrà ogni $r \in R$, dato che $1*r$ deve appartenere ad I. Da qui deriva $I = R$, ideale banale.
Viceversa, sia R un anello commutativo con unità privo di ideali non banali e $a \neq 0$ un suo elemento. L'insieme $I = \{a*r \mid r \in R\}$ sarà un ideale che conterrà sicuramente a, dato che R possiede l'unità. Per ipotesi I è un ideale banale ed essendo diverso da zero, sarà uguale a R. Possiamo quindi asserire che possiede anche lui il neutro moltiplicativo che per costruzione di I, può essere scritto come $1 = a*r$, per qualche $r \in R$. Questo prova che a è invertibile e ciò può essere fatto per un qualsiasi a in input. L'esistenza dell'inverso moltiplicativo per ogni elemento non nullo conclude la prova che R sia un campo. □
L'ultima definizione di questa sezione dimostra che se R è un anello commutativo con unità, un ideale I sarà massimale se e solo se R/I è un campo.
Dimostrazione. Se R è commutativo con unità, anche R/I è commutativo con unità, ma allora per la dimostrazione appena fatta

$$R/I \text{ campo} \iff R/I \text{ è privo di ideali non banali}$$

Per la corrispondenza biunivoca che esiste tra gli ideali di R contenti I e gli ideali di R/I, questo significa che

$$R/I \text{ campo} \iff I \text{ è massimale.} \qquad \square$$

1.3 Introduzione a Isabelle

Isabelle è un "proof assistant" sviluppato sotto la direzione di Larry Paulson presso l'Università di Cambridge e Tobias Nipkow presso la TU di Monaco.
Isabelle è un sistema che permette di implementare formalismi logici e si distingue da Isabelle/HOL, la sua specializzazione che tratta logiche di ordine superiore (High Order Logic). Si fa spesso riferimento a quest'ultima con l'espressione:

$$\text{HOL} = \textit{programmazione funzionale} + \textit{Logica}$$

In particolare Isabelle introduce HOL come un linguaggio di programmazione funzionale, in cui si forniscono prove induttive di proprietà di funzioni ricorsive. La logica HOL così fatta si rifà alla teoria dei tipi di Church, originariamente introdotta da Bertrand Russel nei primi del novecento come fondamento affidabile della matematica di fronte alle contraddittorietà da lui notate in alcune formulazioni insiemistiche, come la nota antinomia del barbiere. Vi sono molteplici interpretazioni della teoria dei tipi, una tra queste considera essenzialmente i tipi all'incirca come insiemi, ma capaci di fornire informazioni sintattiche (es1: $3 + (7 * 8)^5 : \textit{nat}$), dove invece gli insiemi forniscono informazioni semantiche (es2: $3 \in \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}$).
La differenza tra questi due diversi livelli d'informazione non è sempre così facile da distinguere, ma negli esempi presentati si può notare chiaramente che $3 + (7 * 8)^5$ sia di tipo nat come semplice conseguenza del fatto che 3, 7 e 8 siano numeri naturali e $+, *$ siano le due operazioni definite nell'anello dei naturali. Mentre la definizione di appartenenza ai naturali "$3 \in \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}$" secondo l'istanza dell'ultimo teorema di Fermat, necessita di una prova per essere verificata. Nel primo esempio non abbiamo bisogno di alcuna prova, ma di una unica computazione per affermarlo. Si potrebbe quindi dire che un semplice criterio per capire quando si stiano trattando informazioni da un punto di vista sintattico rispetto al semantico sia la possibile esistenza di un algoritmo in grado di verificarle. Un'ulteriore raffinatezza pone il fuoco della differenza nella decidibilità computazionale dell'informazione che si sta verificando.

Negli anni '70 Martin-Lof introdusse una teoria dei tipi chiamata Intuitionistic Type Theory, spesso abbreviata come MLTT che ha da subito attratto l'interesse sia di matematici che di informatici. Martin-Lof stesso descrisse la sua teoria dei tipi sia come paradigma di programmazione sia come una teoria degli insiemi adatta a formalizzare la matematica costruttiva, con la possibilità di estrarre il contenuto computazionale delle dimostrazioni tramite programmi. Questa doppia entità di MLTT ha ispirato l'introduzione di programmi detti "proof-assistant" in grado di aiutare un utente a formalizzare una dimostrazione matematica al calcolatore.
I tipi presenti in Isabelle sono:

- tipi base: bool (booleani), nat ($\mathbb{N}$), int ($\mathbb{Z}$)
- costruttori di tipo: list (liste), set (insiemi)
- tipo funzione: denotati da '$\Rightarrow$'
- variabili di tipo: denotate da $'a, 'b$ ecc..

Tutti i tipi possono essere facilmente costruiti usando la key-word **datatype** seguita da una definizione formata dal caso base e dal caso generico induttivo. Il tipo base "nat" ad esempio, può essere definito a partire dal principio di induzione in accordo con gli assiomi di Peano. La definizione di tutti i concetti primitivi dei naturali come la relazione d'ordine e le operazioni di somma e prodotto vengono quindi eseguite per ricorrenza, e le loro proprietà sono dimostrate per induzione. Ci basterà un solo simbolo di costante $0 \in \mathbb{N}$ ed una funzione iniettiva $Suc(n) : \mathbb{N} \to \mathbb{N}$ detta successore, che non abbia 0 nella sua immagine, per definire $\mathbb{N}$ come:

$$\textbf{datatype}\ nat = 0 \mid Suc(nat)$$

A questo punto, per definire una qualsiasi proprietà o funzione su $\mathbb{N}$ basterà farlo per il caso base 0 e il caso induttivo Suc(n) e verrà conseguentemente estesa ricorsivamente per ogni $n \in \mathbb{N}$.

$$\begin{aligned}&\textbf{fun}\ add :: \text{"}nat \Rightarrow nat \Rightarrow nat\text{"}\ \textbf{where}\\ &\text{"}\textbf{add}\ \ 0\ n = n\text{"}|\\ &\text{"}\textbf{add}\ \ (Suc\, m)\ n = Suc\,(add\, m\, n)\text{"}\end{aligned}$$

Da quanto appena scritto si può notare come la funzione binaria "add" venga definita in Isabelle come "$nat \Rightarrow nat \Rightarrow nat$", questo perchè la notazione (A x B) $\Rightarrow C$ risulta isomorfa a $A \Rightarrow (B \Rightarrow C)$ e di conseguenza anche a $A \Rightarrow B \Rightarrow C$.

Quello di introdurre funzioni attraverso elementi di tipo funzione a cui viene associata una definizione precisa per casi, non è ovviamente l'unico modo a nostra disposizione per trattare le funzioni. Si può infatti utilizzare in Isabelle il λ-calcolo, introdotto da Alonzo Church nel 1930, che risulta un modo semplice di scrivere funzioni anonime. Invece di scrivere $f(x) = x + 5$, che sottintende in Isabelle la dichiarazione di 'f' di tipo funzione, possiamo definire anonimamente $\lambda x.\, x + 5$, che risulta semplicemente un termine che aggiunge 5 ad un parametro passato. Basterà infatti scrivere un'espressione del tipo $(\lambda x.\, t)\, a$, per sostituire la variabile x, con quanto specificato in a, all'interno di t. La sintassi delle λ-espressioni segue la forma:

$$M, N ::= \quad x \mid (\lambda x.\, M) \mid (M\, N)$$

dove x rappresenta una variabile qualsiasi, $(\lambda x.\, M)$ una funzione con argomento x e corpo M ed infine $(M\, N)$ rappresenta l'applicazione della funzione M all'argomento N. Come specificato poco fa una λ-espressione in Isabelle rappresenta un termine che, come in ogni linguaggio logico di ordine almeno uguale al primo, risultano elementi fondamentali. I termini vengono identificati da variabili, costanti o dall'applicazione di funzioni. Se f è definita come $\tau_1 \to \tau_2$, e t è di tipo τ_1, allora l'applicazione di $f(t)$ produrrà un termine di tipo τ_2. Si denoterà con $(t :: \tau_1)$ il fatto che il termine t abbia tipo τ_1. Di per se Isabelle computa automaticamente il tipo di tutte le variabili in un termine e questo processo è noto come inferenza di tipo.
Il codice Isabelle generato viene salvato in un file con struttura specifica e fissa

theory T
imports $T_1, ..., T_n$
begin
definitions, theoremas and proofs
end

chiamato theory. Uno degli aspetti fondamentali delle formalizzazioni introdotte in Isabelle è la possibilità di includere precedenti teorie e fare quindi una costruzione incrementale dando per corretto tutto ciò che vi si pone al di sotto.

Capitolo 2

Formalizzazione

La formalizzazione della notazione O-grande in Isabelle presentata da Jeremy Avigad e Kevin Donnelly della Carnegie Mellon University [6] utilizza parte della vecchia libreria HOL-Complex realizzata da Jacques D. Fleuriot e Lawrence C. Paulson nel 2003, ad oggi chiamata semplicemente HOL. Questa include una teoria dei numeri reali e dei rudimentali dell'analisi sui reali. La trattazione prevede anche lo sviluppo assiomatico degli anelli realizzato da Markus Wenzel e Gertrud Bauer, che saranno importanti nella formalizzazione.

2.1 HOL library

Tra le varie teorie definite in questa libreria sicuramente una delle più importanti è quella sui reali che, poggiando direttamente sulla teoria dei razionali, fornisce una formalizzazione dei numeri reali come classi di equivalenza delle sequenze di razionali di Cauchy, secondo la costruzione di Méray-Cantor.
Sia $\mathbb{Q}^{\mathbb{N}}$ l'insieme delle successioni $s = (s_0, s_1, s_2, ...) = (s_i)_{i\in\mathbb{N}}$ di numeri razionali, riusciamo a definire in $\mathbb{Q}^{\mathbb{N}}$ l'addizione e la moltiplicazione per componenti. In particolare presi $s = (s_0, s_1, s_2, ...) = (s_i)_{i\in\mathbb{N}}$ e $p = (p_0, p_1, p_2, ...) = (p_i)_{i\in\mathbb{N}}$ appartenenti a $\mathbb{Q}^{\mathbb{N}}$ poniamo:

$$p + q = (p_0 + q_0, p_1 + q_1, ...) = (p_i + q_i)_{i\in\mathbb{N}}$$
$$p * q = (p_0 * q_0, p_1 * q_1, ...) = (p_i * q_i)_{i\in\mathbb{N}}$$

Possiamo quindi esprimere ogni $r \in \mathbb{Q}$ come una successione costante del termine stesso. La struttura algebrica appena formata $(\mathbb{Q}^{\mathbb{N}}, +, *)$ risulta essere un anello commutativo unitario con e_1, neutro rispetto alla $+$, espresso come successione

costante di tutti 0 ed e_2, neutro rispetto alla $*$, come successione costante di tutti 1. Una successione $s \in \mathbb{Q}^{\mathbb{N}}$ si dice di Cauchy se:

$$\forall \varepsilon \in \mathbb{Q},\, \varepsilon > 0 \;\; \exists v \in \mathbb{N} \;\text{ tale che }\; |\, s_m - s_n \,| \leq \varepsilon \quad \forall m,n > v$$

Una successione di Cauchy $p \in \mathbb{Q}^{\mathbb{N}}$ si dice convergente con limite $r \in \mathbb{Q}$, se la successione $(p-r)$ converge a zero. L'insieme delle successioni convergenti ad uno stesso limite $r \in \mathbb{Q}$ è il numero r pensato come numero reale.
Un sottoinsieme non vuoto dell'anello appena introdotto si dice sottoanello se esso stesso risulta un anello rispetto alle operazioni definite. Denotiamo quindi con C il sottoanello di $\mathbb{Q}^{\mathbb{N}}$ costituito da tutte le successioni di Cauchy, e con C_0 il sottoinsieme di $\mathbb{Q}^{\mathbb{N}}$ delle successioni convergenti a zero, ideale massimale di C. Andremo a definire $\mathbb{R}$ come il quoziente C/C_0, che risulterà il campo dei numeri reali, in accordo con quanto descritto in 1.2. In questa costruzione dei reali abbiamo dato per scontato l'esistenza dei razionali, i quali saranno definiti in Isabelle solamente attraverso la nozione di campo archimedeo.
Tra i rudimenti di analisi accennati nell'introduzione al capitolo, vi è sicuramente la teoria "MacLaurin.thy", direttamente importata nella teoria "BigO.thy" che formalizza il concetto di O-grande in Isabelle e che verrà analizzata nel prossimo sottocapitolo.
La serie di Taylor riguarda un aspetto analitico molto importante, ossia il problema dell'approssimazione di una funzione mediante polinomi. Vi sono molti modi per realizzare una tale approssimazione e quello di Taylor presenta un carattere locale, nel senso che studia un intorno di un punto. Facendo riferimento alle definizioni in [4], presa una funzione $f : (a,b) \to \mathbb{R}$, derivabile in $x_0 \in (a,b)$, allora se $x \in (a,b)$ possiamo scrivere:

$$f(x) = f(x_0) + f'(x_0)(x-x_0) + o(x-x_0) \qquad \text{per } x \to x_0.$$

Possiamo quindi interpretare il polinomio di primo grado appena scritto come un'approssimazione lineare di f, che è tangente alla funzione stessa nel punto di ascissa x_0. Il termine "$o(x-x_0)$" rappresenta invece l'errore $E(x)$, espresso nella forma di Peano, che si commette nell'approssimazione. Quest'ultima risulta tanto migliore quanto più si considera piccoli intorni di x_0, mentre peggiora drasticamente man mano che ci si allontana da x_0. Un risultato più preciso nell'approssimazione della funzione f è ottenibile aumentando il grado del polinomio e di conseguenza le derivate calcolabili di f. In particolare sia $f : (a,b) \to \mathbb{R}$ derivabile n volte in $x_0 \in (a,b)$, allora esiste un unico polinomio $T_n(x)$ calcolabile di grado

$\leq n$, ottenuto dalla formula di Taylor, nella forma:

$$T_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k$$

$T_n(x)$ così costruito prende il nome di polinomio di Taylor di grado n generato da f con centro in x_0. Se $x_0 = 0$, T_n assume la forma

$$T_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(0)}{k!} x^k$$

e viene detto anche polinomio di Mac Laurin di grado n. Questa teoria analitica in Isabelle agevola la trattazione di O-grande per funzioni, dato che ci consente di confrontare due funzioni più facilmente come polinomi.
La terza importante teoria presente nella libreria, sempre accennata nell'introduzione al capitolo, è quella che tratta lo sviluppo assiomatico degli anelli. I concetti di cui necessita questa costruzione riguardano i gruppi, gli insiemi e le funzioni, che vengono importati per realizzare la nozione di anello attraverso usa serie di passi caratterizzati dall'utilizzo della key-word **class**. In Isabelle questa notazione permette di definire delle "teorie locali" separando la teoria, ovvero la specifica assiomatica costituita da parametri ed ipotesi, dal suo corpo costituito da estensioni arbitrarie di definizioni. L'aspetto chiave di questo costrutto è la possibilità di generare delle sottoclassi estendendo classi già esistenti, mediante l'introduzione di ulteriori operazioni e/o proprietà. Tornando al concetto di anello, questo verrà formato per estensione incrementale di varie classi, la prima delle quali risulta quella di semianello.

class semiring = ab_semigroup_add + semigroup_mult +
assumes distrib_right [algebra_simps, algebra_split_simps]:
"$(a+b)*c = a*c+b*c$"
assumes distrib_left [algebra_simps, algebra_split_simps]:
"$a*(b+c) = a*b+a*c$"

La classe di semianello, come si può leggere dalla prima riga, viene a sua volta definita estendendo le classi di semigruppo abeliano additivo e semigruppo moltiplicativo introducendo la distributività a destra e a sinistra della moltiplicazione rispetto alla somma. Per un ulteriore approfondimento si rimanda a [5], contenente il link alla libreria che organizza le teorie accennate.

2.2 O-grande in Isabelle

La formalizzazione trattata in questo sottocapitolo sarà quella ad oggi implementata nella libreria Isabelle, che poggia le basi sulla proposta originale di Avigad e Donnelly del 2004. Volendo generalizzare lo studio della complessità, la formalizzazione non è equivalente a quella introdotta in 1.1 se non per funzioni continue. Ci avvicineremo più al significato analitico del termine O-grande rispetto alla sua semplificazione algoritmica introdotta nel primo capitolo, che si limita a considerare solo funzioni asintoticamente non negative. Per far questo introduciamo il concetto di valore assoluto come

$$|x| = \begin{cases} x & \text{se } 0 \leq x \\ -x & \text{altrimenti} \end{cases}$$

che deve valere nel codominio delle funzioni che prenderemo in considerazione. Il valore assoluto è definibile in $\mathbb{Z}$, $\mathbb{Q}$ o $\mathbb{R}$.
Presentiamo ora l'origine della nuova notazione contenente il valore assoluto, per poi confrontarla con la prima formulazione del sottocapitolo 1.1. Facendo riferimento a quanto riportato in [7] la notazione O-grande, introdotta da Bachmann, prese spunto dalla precedente nozione di L-grande proposta da N. G. de Bruijn. Mentre però $L(n)$ era usata per indicare un numero con valore assoluto non maggiore di n, Bachmann decise di generalizzare e considerare con $O(\alpha)$ e $\alpha \neq 0$ un qualsiasi numero il cui valore assoluto fosse al più $|\alpha|$ moltiplicato per una qualche costante. L'applicazione di una tale notazione per descrivere le funzioni deve prevedere vincoli sulle variabili coinvolte. Per esempio l'espressione

$$f(n) = O(g(n)) \quad \text{quando } n \to \infty$$

afferma che la condizione di O-grande è garantita quando n è "vicino" ad ∞ e che quindi non ci interessiamo del comportamento della funzione se non per valori molto grandi di n. Il vincolo sulla variabile risulta quindi quello di non considerare la funzione su tutto il dominio, bensì focalizzare l'attenzione su intervalli in cui il valore di n è molto grande. Allo stesso modo l'espressione

$$f(x) = O(g(x)) \quad \text{quando } x \to 0$$

pone, in maniera opposta rispetto a prima, il nostro interesse su valori di x molto piccoli. Introduciamo ora la nuova definizione di O-grande proposta da Avigad e Donnelly per la formalizzazione in Isabelle

$$O(g(x)) = \{h(x) \mid \exists c\ \forall x\ (\ |h(x)| \leq c * |g(x)|\ )\}$$

La principale differenza rispetto alla precedente risiede nell'assenza di limitazioni sul valore di x dopo il quale il concetto possa essere considerato vero, dato che si quantifica universalmente su x. Questa nuova definizione viene infatti usata per analizzare le funzioni in particolari intervalli d'interesse, in cui il dominio di queste non deve più essere limitato da alcun x_0. La presenza del valore assoluto inoltre non ci impone più di valutare solamente funzioni positive, aumentando la generalità della trattazione. Possiamo appunto notare che prendendo in considerazione funzioni positive e limitate in un intervallo $[x_0, \infty]$ le due definizioni coincidono. Inoltre se una funzione definita in $\mathbb{R}$, $\mathbb{Q}$ o $\mathbb{Z}$ è continua, allora lo sarà anche in ogni intervallo $[a,b]$ del suo dominio che vogliamo analizzare con la nuova definizione. Eventuali limitazioni della definizione, come quelli che riguardano i valori assumibili da c, saranno introdotte come lemmi.
La formalizzazione in Isabelle di quanto detto risulta:

definition bigo :: "('a ⇒' b :: *linordered_idom*) ⇒ ('a ⇒' b) *set*"
("(1*O*'(_'))") **where**
"*O*(*f* :: ('a ⇒' b)) == {*h*. ∃*c*. ∀*x*. |*h x*| ≤ *c* ∗ |*f x*|}"

dove 'bigo' viene definito di tipo funzione che, ricevendo in input una funzione di tipo $('a \Rightarrow' b)$, restituisce un insieme di funzioni dello stesso tipo. Un aspetto importante di questa definizione è che il codominio delle funzioni trattate deve essere di tipo "linordered_idom", così come viene presentato all'interno della teoria dei reali accennata in 2.1

class linordered_idom = *comm_ring_*1 + *ordered_ab_group_add* + *sgn_if* +
linordered_comm_semiring_strict + *abs_if*

Per spiegare il significato del termine "linordered_idom", verrà suddiviso ed analizzato nelle due parole che lo compongono: linordered e idom. Quest'ultimo risulta semplicemente un'abbreviazione di "integral domain" ossia di dominio di integrità, in accordo con la definizione presentata in 1.2.
Invece il primo termine "linordered" fa riferimento alla presenza di un ordine lineare o totale sul dominio di integrità. Una relazione di ordine ρ viene detta lineare o totale in un insieme A, se $\forall\, a,b \in A$ è sempre possibile affermare $a\,\rho\,b$ o $b\,\rho\,a$. Un classico esempio di una relazione che gode di tale proprietà è la relazione '$\leq$' definita in $\mathbb{Z}$, $\mathbb{Q}$ o $\mathbb{R}$. Preso ad esempio $\mathbb{R}$ e due suoi elementi r_1 e r_2, possiamo infatti sempre dire se valga $r_1 \leq r_2$ o il contrario. L'applicazione di un tale ordine a tutto l'insieme dei reali forma una linea in cui ogni elemento è maggiore di quelli che lo precedono e minore di tutti quelli che lo seguono. Da

qui deriva il termine ordine lineare.
Come già accennato nel sottocapitolo precedente la teoria che contiene le formalizzazioni riguardo ad O-grande prende il nome di BigO.thy e viene introdotta nella libreria Isabelle all'interno della sessione HOL-Library.
Rispetto alla nuova formalizzazione di O-grande presentata, la definizione introdotta nel primo capitolo prevede $c \neq 0$, una condizione utile nello studio degli algoritmi. Questo potrà essere ottenuto mediante i due lemmi successivi:

lemma bigo_pos_const: "$(\exists c :: 'a :: linordered\_idom.\ \forall x.\ |h\,x| \leq c * |f\,x|) \iff (\exists c.\ 0 < c\ \wedge\ (\forall x.\ |h\,x| \leq c * |f\,x|))$"

lemma bigo_alt_def: "$O(f) = \{h.\ \exists c.\ 0 < c\ \wedge\ (\forall x.\ |h\,x| \leq c * |f\,x|)\}$"

La sessione HOL-Library si compone di altre svariate teorie che, supportando la teoria BigO stessa, permettono di eseguire calcoli nella forma $f + g = O(g)$ e $f = g + O(h)$. Per far questo e trattare la somma di funzioni $f + g$ si utilizzerà il concetto di somma puntuale. Prese due funzioni f e g e supposti uguali il loro dominio A e codominio B, definiamo $\forall a \in A\ \ (f+g)(a) = f(a) + g(a)$.
La teoria che esplicita questo concetto in Isabelle:

instantiation "fun" :: (type, plus) plus
begin

definition "$f + g = (\lambda x.\ f\,x + g\,x)$"
instance ..

end

è la "Function_Algebras.thy" che viene direttamente importata come base per la definizione di BigO. Un'altra teoria fondamentale, anch'essa importata in BigO è la "Set_Algebras.thy" che introduce la corrispondente procedura di somma puntuale sugli insiemi, permettendo operazioni del tipo $O(g)$ + $O(f)$. Presi quindi A e B due insiemi qualsiasi di elementi il cui tipo ha definita l'operazione somma,

definiremo $A + B$ come l'insieme della somma dei loro elementi, in Isabelle:

instantiation set :: (plus) plus
begin

definition plus_set :: "'a :: *plus set* ⇒' a *set* ⇒' a *set*"
where set_plus_def : "$A + B = \{ c.\ \exists a \in A.\ \exists b \in B.\ c = a + b \}$"

instance ..

end

In questa formalizzazione si può notare come l'operazione plus_set definita prenda in input due insiemi A e B e restituisca l'insieme formato dagli elementi c ottenuti dalla somma di a e b, elementi rispettivamente di A e B.
Rimanendo sempre all'interno della teoria Set_Algebras.thy troviamo le formalizzazioni necessarie per trattare termini del tipo $f + O(g)$, in accordo con l'interpretazione fornita in 1.1. In particolare viene introdotto l'operatore "$+o$" che prende come argomenti un elemento di tipo a ed un insieme di elementi dello stesso tipo:

definiton elt_set_plus :: "'a :: *plus* ⇒ 'a *set* ⇒ 'a *set*" (**infixl** "$+o$" 70)
where "$a +o\ B\ =\ \{c.\ \exists b \in B.\ c = a + b\}$"

Allo stesso modo viene introdotto l'operatore "$*o$" che si occupa di trattare espressioni del tipo $f * O(g)$:

definiton elt_set_times :: "'a :: *times* ⇒ 'a *set* ⇒ 'a *set*" (**infixl** "$*o$" 80)
where "$a *o\ B\ =\ \{c.\ \exists b \in B.\ c = a * b\}$"

Per usufruire delle formalizzazioni appena introdotte basterà infatti fissare a come elemento di tipo funzione, in cui sia definita la somma o la moltiplicazione, a seconda che si voglia usare $+o$, rispettivamente, $*o$. Fatto questo il secondo parametro e il risultato saranno insiemi di funzioni dello stesso tipo, esattamente come definiti. Facendo sempre riferimento alla trattazione classica di O-grande nel primo capitolo abbiamo affermato che convenzionalmente un'espressione del tipo $x = O(f)$ viene interpretata come $x \in O(f)$. Per far questo introduciamo il simbolo "$= o$"

abbreviation (input) elt_set_eq :: "'a ⇒ 'a *set* ⇒ *bool*" (**infix** "$= o$" 50)
where "$x = o\ A\ \equiv\ x \in A$"

Da qui in poi, per una maggiore comprensione delle proprietà presentate al capitolo successivo, la notazione "$+o$" sarà sostituita con $+_o$, "$*o$" con $*_o$ e "$= o$" con $=_o$.
Nella prima versione del 2004 vi era anche il simbolo "=s" usato per interpretare direttamente l'uguaglianza come relazione di inclusione insiemistica in espressioni del tipo $f + O(g) = O(h)$. In seguito "=s" fu sostituito da una serie di lemmi che garantiscono lo stesso significato senza però l'uso di specifici costrutti sintattici.
Facendo sempre un confronto con la precedente versione possiamo notare tra i cambiamenti più rilevanti la possibilità di trattare forme del tipo $f < g + O(h)$, l'eliminazione del vecchio operatore O-grande "bigoset" sugli insiemi e la generalizzazione dei lemmi riguardanti la somma.
Dato che nelle trattazioni di espressioni contenenti O-grande si usa il simbolo di uguaglianza per denotare situazioni distinte di appartenenza, inclusione ed equivalenza insiemistica ed il significato corretto dipende solamente dal contesto in cui ci troviamo, eviteremo possibili confusioni nel capitolo successivo sostituendo il simbolo '$=$', con la sua corretta interpretazione.

Capitolo 3

Proprietà Base

In questo capitolo saranno presentate e dimostrate in Isabelle le proprietà base indicate originariamente da Avigad e Donnelly per la prima versione della formalizzazione, qua riformulate per quella definitiva ad oggi già facente parte della libreria. Le proprietà saranno prima presentate affiancate dalla loro definizione matematica e dopo dalla loro implementazione in Isabelle. Per una migliore comprensione il capitolo sarà diviso in sottocapitoli facenti riferimento alle due teorie della libreria Isabelle contenenti le proprietà descritte: Set_Algebras e BigO.

3.1 Proprietà in Set_Algebras.thy

Le prime proprietà che tratteremo sono:

Nome della Proprietà	**Definizione Matematica**
set_plus_intro	$(a \in C, b \in D) \implies a+b \in C+D$
set_plus_intro2	$b \in C \implies a+b \in a+C$
set_plus_rearrange	$(a+C)+(b+D) = (a+b)+(C+D)$
set_plus_rearrange2	$a+(b+C) = (a+b)+C$
set_plus_rearrange3	$(a+C)+D = a+(C+D)$
set_plus_rearrange4	$C+(a+D) = a+(C+D)$
set_zero_plus	$0+C = C$

In queste definizioni a e b fanno riferimento ad elementi di tipo "plus", un tipo la cui descrizione prevede il concetto di somma, mentre C e D fanno riferimento ad insiemi di elementi dello stesso tipo. Espressioni della forma "$a+b$" riguardano la somma tra elementi di tipo plus e, dato che stiamo considerando la notazione O-grande, si prenderanno in considerazione funzioni la cui addizione farà riferimento al concetto di somma puntuale definita in 2.2. Espressioni del tipo $a+C$ saranno invece trattate con l'operatore $+_o$, che definisce per l'appunto il concetto di addizione tra un elemento ed un insieme di elementi dello stesso tipo. Infine la somma tra due insiemi $C+D$ sarà considerata come la loro somma puntuale, sempre in accordo con quanto detto in 2.2. Le 4 proprietà di riarrangiamento consentono di riscrivere un termine formato dai tre tipi di addizione appena introdotti, ordinandolo con prima le somme di semplici elementi e poi le somme tra insiemi. Questo permette di verificare più facilmente l'identità di termini, che presentano gli stessi elementi, solamente sommati in ordine diverso. Il termine 0 nell'ultima proprietà non fa riferimento all'insieme contenete solo lo 0, bensì all'elemento 0 stesso. Le dimostrazioni algebriche di queste proprietà seguono direttamente dalle definizioni presentate nei capitoli precedenti e per questo motivo di seguito verranno semplicemente presentate la loro trasposizione in Isabelle (tra un attimo spiegheremo il significato delle keywords usate).

lemma set_plus_intro [intro]: "$a \in C \implies b \in D \implies a+b \in C+D$"
by (auto simp add: set_plus_def)

lemma set_plus_intro2 [intro]: "$b \in C \implies a+b \in a+_o C$"
by (auto simp add: elt_set_plus_def)

lemma set_plus_rearrange: "$(a+_o C)+(b+_o D) = (a+b)+_o (C+D)$"
for a b :: "$'a$ :: *comm_monoid_add*"
apply (auto simp add: elt_set_plus_def set_plus_def ac_simps)
apply (rule_tac x = "$ba+bb$" **in** exI)
apply (auto simp add: ac_simps)
apply (rule_tac x = "$aa+a$" **in** exI)
done

Nell'ultimo lemma a e b vengono considerati di tipo "comm_monoid_add" e nei successivi si farà invece riferimento a "semigroup_add". A livello di sintassi Isa-

belle, la key-word **instance** risulta un modo conveniente per creare un'istanza di una classe di tipo senza la necessità di specificare operazioni, ma semplicemente considerando definizioni già introdotte:

instance set :: (comm_monoid_add) comm_monoid_add
by standard (simp_all add: set_plus_def)

instance set :: (semigroup_add) semigroup_add
by standard (force simp add: set_plus_def add.assoc)

Ritornando alle proprietà restanti:

lemma set_plus_rearrange2: "$a +_o (b +_o C) = (a + b) +_o C$"
for a b :: "$'a$ :: *semigroup_add*"
by (auto simp add: elt_set_plus_def add.assoc)

lemma set_plus_rearrange3: "$(a +_o B) + C = a +_o (B + C)$"
for a :: "$'a$ :: *semigroup_add*"
apply (auto simp add: elt_set_plus_def set_plus_def)
apply (blast intro: ac_simps)
apply (rule_tac x = "$a + aa$" **in** exI)
apply (rule conjI)
apply (rule_tac x = "*aa*" **in** bexI)
apply auto
apply (rule_tac x = "*ba*" **in** bexI)
apply (auto simp add: ac_simps)
done

theorem set_plus_rearrange4: "$C + (a +_o D) = a +_o (C + D)$"
for a :: "$'a$ :: *comm_monoid_add*"
apply (auto simp add: elt_set_plus_def set_plus_def ac_simps)
apply (rule_tac x = "$aa + ba$" **in** exI)
apply (auto simp add: ac_simps)
done

lemma set_zero_plus [simp]: "$0 +_o\ C = C$"
for C :: "$'a$ :: *comm_monoid_add set*"
by (auto simp add: elt_set_plus_def)

Analizziamo ora una delle regole d'istanziazione spesso ricorrenti nelle dimostrazioni dei lemmi: "**apply** (rule_tac $v_1 = t_1$ **and** ... **and** $v_n = t_n$ **in** *theorem*)".
Per fare questo partiamo prima da una sua definizione più semplice e, come spiegato in [9], prendiamo come riferimento una regola generica del tipo:

$$\frac{P_1 \ \ldots \ P_n}{Q}$$

Chiamata questa regola per semplicità R, possiamo affermare che applicando l'istruzione "rule R" si procede con l'unificazione logica di Q con il corrente subgoal andandolo a sostituire con n nuovi subgoals, per l'appunto i vari P_i.
Data questa definizione possiamo notare come la formulazione contenente il "tac" agisca allo stesso modo, permettendo però anche l'istanziazione di variabili mediante la specifica $v_i = t_i$. Questa regola che esegue quindi la risoluzione con anche un'istanziazione esplicita, si applica quando la conclusione definita in *theorem* coincide con la conclusione del subgoal corrente. In questa forma il t_i rappresenta un termine e permette l'instanziazione di v_i che può contenere anche parametri presi direttamente dal subgoal da dimostrare. In particolare nei lemmi precedenti e poi anche in quelli successivi, espressioni del tipo $v_i =$ "$aa + ba$" fanno riferimento a termini che vengono presi dai subgoals correnti e che vengono usati per l'istanziazione della variabile v_i.
Per approfondire e mostrare ulteriori dettagli, prendiamo come riferimento il lemma precedente "set_plus_rearrange3" e riscriviamolo affiancandolo all'evoluzione dei subgoals correnti. Dopo l'esecuzione di ogni istruzione i subgoals richiesti si modificheranno di conseguenza in base alla regola applicata.
Isabelle aggiunge dei nomi nuovi per ogni variabile legata, quando scriviamo "se $\exists x.\ P(x)$ allora ..." la regola di esiste-eliminazione sceglie un nuovo nome a e chiede di provare "se $P(a)$ allora ...".
Per esempio nel lemma "set_plus_rearrange3" l' istruzione rule_tac x = "$a + aa$" assegna ad una nuova variabile ab una somma $a + aa$, con aa a sua volta nuova variabile. La rappresentazione tabellare presentata di seguito permette di chiarire in dettaglio come avviene la dimostrazione del lemma "set_plus_rearrange3" con Isabelle.

Lemma Isabelle	**Output Subgoal**
lemma set_plus_rearrange3: "$(a +_o B) + C = a +_o (B + C)$"	**goal** (1 subgoal): 1. $a +_o B + C = a +_o (B + C)$
for a :: "'a :: *semigroup_add*"	**goal** (1 subgoal): 1. $a +_o B + C = a +_o (B + C)$
apply (auto simp add: elt_set_plus_def set_plus_def)	**goal** (2 subgoals): 1. $\bigwedge b\ ba.\quad b \in B \Longrightarrow ba \in C \Longrightarrow \exists bb.\ (\exists a \in B.\ \exists b \in C.\ bb = a + b) \wedge a + b + ba = a + bb$ 2. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow \exists ab.\ (\exists b \in B.\ ab = a + b) \wedge (\exists b \in C.\ a + (aa + ba) = ab + b)$
apply (blast intro: ac_simps)	**goal** (1 subgoal): 1. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow \exists ab.\ (\exists b \in B.\ ab = a + b) \wedge (\exists b \in C.\ a + (aa + ba) = ab + b)$
apply (rule_tac x = "$a + aa$" **in** exI)	**goal** (1 subgoal): 1. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow (\exists b \in B.\ a + aa = a + b) \wedge (\exists b \in C.\ a + (aa + ba) = a + aa + b)$
apply (rule conjI)	**goal** (2 subgoals): 1. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow \exists b \in B.\ a + aa = a + b$ 2. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow \exists b \in C.\ a + (aa + ba) = a + aa + b$
apply (rule_tac x = "aa" **in** bexI)	**goal** (3 subgoals): 1. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow a + aa = a + aa$ 2. $\bigwedge a\ ba.\quad a \in B \Longrightarrow ba \in C \Longrightarrow a \in B$ 3. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow \exists b \in C.\ a + (aa + ba) = a + aa + b$
apply auto	**goal** (1 subgoal): 1. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow \exists b \in C.\ a + (aa + ba) = a + aa + b$
apply (rule_tac x = "ba" **in** bexI)	**goal** (2 subgoals): 1. $\bigwedge aa\ ba.\quad aa \in B \Longrightarrow ba \in C \Longrightarrow a + (aa + ba) = a + aa + ba$ 2. $\bigwedge a\ ba.\quad a \in B \Longrightarrow ba \in C \Longrightarrow ba \in C$
apply (auto simp add: ac_simps)	**goal:** No subgoals!
done	

Dato che ragionare con espressioni contenti l' O-grande, riguarda spesso ragionamenti circa l'inclusione insiemistica, come il determinare quale tra più insiemi di funzioni sia il più grande, e possa di conseguenza includere gli altri, le seguenti proprietà sulla monotonicità dell'inclusione risultano centrali:

Nome della Proprietà	**Definizione Matematica**
set_plus_mono	$C \subseteq D \implies a + C \subseteq a + D$
set_plus_mono2	$(C \subseteq D, E \subseteq F) \implies C + E \subseteq D + F$
set_plus_mono3	$a \in C \implies a + D \subseteq C + D$
set_plus_mono4	$a \in C \implies a + D \subseteq D + C$
set_plus_mono5	$(a \in C, B \subseteq D) \implies a + B \subseteq C + D$

Le prime quattro di queste proprietà vengono dichiarate direttamente al ragionatore automatico, fornendogli un utile supporto nei calcoli asintotici. Visto che le dimostrazioni matematiche delle proprietà risultano, come in questo caso, spesso ovvie d'ora in avanti ci soffermeremo sempre sulla loro formalizzazione Isabelle, e verranno presentate le spiegazioni solo di quelle proprietà che potrebbero non essere di immediata comprensione.

lemma set_plus_mono [intro!]: "$C \subseteq D \implies a +_o C \implies a +_o D$"
by (auto simp add: elt_set_plus_def)

lemma set_plus_mono2 [intro]: "$C \subseteq D \implies E \subseteq F \implies C + E \subseteq D + F$"
for C D E F :: "'a :: *plus set*"
by (auto simp add: set_plus_def)

lemma set_plus_mono3 [intro]: "$a \in C \implies a +_o D \subseteq C + D$"
by (auto simp add: elt_set_plus_def set_plus_def)

lemma set_plus_mono4 [intro]: "$a \in C \implies a +_o D \subseteq D + C$"
for a :: "'a :: *comm_monoid_add*"
by (auto simp add: elt_set_plus_def set_plus_def ac_simps)

lemma set_plus_mono5: "$a \in C \implies B \subseteq D \implies a +_o B \subseteq C + D$"
apply (subgoal_tac "$a +_o B \subseteq a +_o D$")
apply (erule order_trans)
apply (erule set_plus_mono3)
apply (erule set_plus_mono)
done

I primi due lemmi presentati in questo capitolo e le prime quattro proprietà appena introdotte sono caratterizzati dall'etichetta **[intro]**, utilizzata per denotare regole introduttive, in questo caso dichiarate ad Isabelle come regole di semplificazione che potrà adottare nelle dimostrazioni successive. Nelle dimostrazioni dei subgoals di lemmi o teoremi attraverso il comando "**apply** (auto)" si istruisce Isabelle a cercare di provare automaticamente tutti i subgoals previsti mediante semplificazione, utilizzando meccanismi di base e tutte le regole che gli abbiamo fornito. In particolare nella costruzione di una dimostrazione l'utilizzo dei vari comandi **apply** sotto un qualsiasi lemma permettono la manipolazione dei subgoals che Isabelle ci indica come necessari per provare tale lemma, mediante l'utilizzo di regole già introdotte o semplicemente tramite regole di riscrittura o induzione. La semplificazione dei lemmi così proposta potrà permettere ad Isabelle di concludere automaticamente i passi restanti e il lemma stesso potrà terminare con la key-word **done**.
Introduciamo ora delle proprietà che riguardano il concetto di moltiplicazione su insiemi, in accordo con la definizione di "$*_o$" in 2.2. Assumendo quindi che il tipo considerato sia un anello commutativo con identità e che quindi l'operazione di moltiplicazione goda della proprietà associativa e commutativa, saranno presentati dei concetti riguardo alla distributività del prodotto sulla somma:

Nome della Proprietà	**Definizione Matematica**
set_times_plus_distrib	$a*(b+C) = a*b+a*C$
set_times_plus_distrib2	$a*(B+C) = a*B+a*C$
set_times_plus_distrib3	$(a+C)*D \subseteq a*D+C*D$

lemma set_times_plus_distrib: "$a *_o (b +_o C) = (a*b) +_o (a *_o C)$"
for a b :: "'a :: *semiring*"

by (auto simp add: elt_set_plus_def elt_set_times_def ring_distribs)

lemma set_times_plus_distrib2: "$a *_o (B + C) = (a *_o B) + (a *_o C)$"
for a :: "'a :: *semiring*"
apply (auto simp add: set_plus_def elt_set_times_def ring_distribs)
apply blast
apply (rule_tac x = "$b + bb$" **in** exI)
apply (auto simp add: ring_distribs)
done

lemma set_times_plus_distrib3: "$(a +_o C) * D \subseteq a *_o D + C * D$"
for a :: "'a :: *semiring*"
apply (auto simp: elt_set_plus_def elt_set_times_def set_times_def
set_plus_def ring_distribs)
apply auto
done

Precisiamo che il tipo "*semiring*" considerato nei lemmi precedenti, riguarda la classe di semianello esattamente come presentata in 2.1, da cui verrà generato il concetto assiomatico di anello proposto da Markus Wenzel e Gertrud Bauer. L'ultima proprietà che introduciamo in questo sottocapitolo, è un lemma che correla la sottrazione classica tra elementi con la somma "$+_o$" :

Nome della Proprietà	**Definizione Matematica**
set_minus_plus	$(a - b) \in C = a \in (b + C)$

La dimostrazione di questo lemma prevede l'utilizzo del tipo "*ab_group_add*" per a e b definitio in Function_Algebras.thy già citata in 2.1 per la formalizzazione del concetto di somma puntuale

instance "fun" :: (type, ab_group_add) ab_group_add
by standard simp_all

lemma set_minus_plus: "$a - b \in C \iff a \in b +_o C$"
for a b :: "$'a$ :: *ab_group_add*"
apply (rule iffI)
apply (rule set_minus_imp_plus)
apply assumption
apply (rule set_plus_imp_minus)
apply assumption
done

3.2 Proprietà in BigO.thy

In questo sottocapitolo saranno invece analizzate tutte le proprietà base inserite all'interno della teoria BigO:

Nome della Proprietà	**Definizione Matematica**
bigo_elt_subset	$f \in O(g) \implies O(f) \leq O(g)$
bigo_elt_subset2	$f \in g + O(h) \implies O(f) \subseteq O(g) + O(h)$
bigo_refl	$f \in O(f)$

In queste proprietà le variabili f e g rappresentano funzioni il cui codominio sia un anello ordinato non degenere (1.2). Rispetto alla prima formulazione di queste proprietà noteremo il maggiore utilizzo di espressioni tra insiemi contenente il simbolo "$\leq$". In questa circostanza, e nei futuri teoremi, il simbolo viene interpretato come generalizzazione ed espansione agli insiemi di una relazione presente tra gli elementi che compongono gli insiemi stessi, ossia le funzioni.

lemma bigo_elt_subset [intro]: "$f \in O(g) \implies (f) \leq O(g)$"
apply (auto simp add: bigo_alt_def)
apply (rule_tac x = "$ca * c$" **in** exI)
apply (rule conjI)
apply simp
apply (rule allI)
apply (drule_tac x = "xa" **in** spec)+

apply (subgoal_tac "$ca * |f\ xa| \leq ca * (c * |g\ xa|)$")
  apply (erule order_trans)
  apply (simp add: ac_simps)
apply (rule mult_left_mono, assumption)
apply (rule order_less_imp_le, assumption)
done

Spesso gli script contenenti lunghe sequenze di **apply** risultano difficili da leggere, soprattutto per quanto riguarda dimostrazioni lunghe, per le quali si preferisce l'utilizzo di Isar. Isar è il linguaggio formale di Isabelle previsto per la formalizzazione delle dimostrazione dato che permette di strutturarle in maniera tale da poter essere lette e comprese senza dover ricorrere all'esecuzione, verificando direttamente tutto ciò che si sta provando in ogni punto. Le istruzioni contenenti **have** sono passi intermedi che ci avvicinano verso il risultato finale espresso dall'istruzione **show**. Quest'ultimo sarà verificato, ovviamente solamente quando ogni step intermedio della dimostrazione avrà esito positivo.

lemma bigo_elt_subset2 [intro]:
  assumes $*$: "$f \in g +_o O(h)$"
  shows "$O(f) \subseteq O(g) + O(h)$"
proof −
  note $*$
  also have "$g +_o O(h) \subseteq O(g) + O(h)$"
    by (auto del: subsetI)
  also have "$... = O(\lambda x.\ |g\,x|) + O(\lambda x.\ |h\,x|)$"
    by (subst bigo_abs3 [symmetric])+ (rule refl)
  also have "$... = O(\lambda x.\ |g\,x|) + (\lambda x.\ |h\,x|)$"
    by (rule bigo_plus_eq [symmetric]) auto
  finally have "$f \in ...$"**.**
  then have "$O(f) \subseteq ...$"
    by (elim bigo_elt_subset)
  also have "$... = O(\lambda x.\ |gx|) + O(\lambda x.\ |hx|)$"
    by (rule bigo_plus_eq, auto)
  finally show ?thesis

by (simp flip: bigo_abs3)
qed

lemma bigo_refl [intro]: "$f \in O(f)$"
apply (auto simp add: bigo_def)
apply (rule_tac x = 1 **in** exI)
apply simp
done

Le proprietà che saranno ora presentate riguardano la definizione stessa dell' O-grande in Isabelle e per questo già accennate nel paragrafo 2.2, nel quale si presenta la formalizzazione.

Nome della Proprietà	**Definizione Matematica**
bigo_pos_const	$(\exists c\ \forall x\,(\lvert h(x)\rvert \leq c * \lvert f(x)\rvert)) \iff$ $(\exists c\ (0 < c \wedge (\forall x\ \lvert h(x)\rvert \leq c * \lvert f(x)\rvert)))$
bigo_alt_def	$O(f) = \{h \mid \exists c\,(0 < c\ \wedge\ \forall x\,(\lvert h(x) \leq c * \lvert g(x)\rvert))\}$

La prima proprietà vincola l'esistenza del fattore costante c ad essere un valore maggiore di 0, mentre solamente dal nome della seconda proprietà, si può intuire che si tratti di una definizione alternativa (alt_def) della formalizzazione originaria di O-grande del paragrafo 2.2. In entrambe proprietà si considera, senza alcuna perdita di generalità, che c sia strettamente positivo e questo potrà tornare utile nei successivi lemmi, per semplificare future dimostrazioni.

lemma bigo_pos_const:
"$(\exists c ::' a ::$ linordered_idom. $\forall x.\ \lvert h\,x\rvert \leq c * \lvert f\,x\rvert) \iff$
$(\exists c.\ 0 < c\ \wedge\ (\forall x.\ \lvert h\,x\rvert \leq c * \lvert f x\rvert))$"
apply auto
apply (case_tac "$c = 0$")
apply simp
apply (rule_tac x = "1" **in** exI)
apply simp
apply (rule_tac x = "$\lvert c\rvert$" **in** exI)

apply auto
apply (subgoal_tac "$c * |fx| \leq |c| * |fx|$")
  apply (erule_tac x = x **in** allE)
  apply force
apply (rule mult_right_mono)
  apply (rule abs_ge_self)
apply (rule abs_ge_zero)
done

lemma bigo_alt_def: "$O(f) = \{h.\ \exists c.\ 0 < c \wedge (\forall x.\ |hx| \leq c * |fx|)\}$"
  by (auto simp add: bigo_def bigo_pos_const)

Come si evince chiaramente in quest'ultimo lemma, la definizione alternativa di O-grande viene semplicemente ottenuta combinando quella che è la definizione originale di O-grande e il lemma che lo precede. La definizione alternativa, infatti, non si discosta molto dall'originale se non appunto per la restrizione sui valori assumibili da c.
Saranno ora introdotte delle proprietà che saranno utili per i calcoli. Verranno presentate prima quelle che riguardano la somma e poi quelle per la moltiplicazione e la sottrazione. Espresse a questo livello di generalità, le loro prove possono fare affidamento solo su proprietà, come la disuguaglianza triangolare, che sono vere in ogni anello ordinato non degenere.

Nome della Proprietà	**Definizione Matematica**
bigo_plus_self_subset	$O(f)+O(f) \subseteq O(f)$
bigo_plus_idemp	$O(f)+O(f) = O(f)$
bigo_plus_subset	$O(f+g) \subseteq O(f)+O(g)$
bigo_plus_subset2	$(A \subseteq O(f),\ B \subseteq O(f)) \Longrightarrow A+B \subseteq O(f)$
bigo_plus_eq	$(\forall x\ (0 \leq f(x)),\ \forall x\ (0 \leq g(x))) \Longrightarrow$ $O(f+g) = O(f)+O(g)$
bigo_plus_absorb_lemma1	$f \in O(g) \Longrightarrow f+O(g) \subseteq O(g)$
bigo_plus_absorb_lemma2	$f \in O(g) \Longrightarrow O(g) \subseteq f+O(g)$
bigo_plus_absorb	$f \in O(g) \Longrightarrow f+O(g) = O(g)$

bigo_plus_absorb2	$(f \in O(g), A \subseteq O(g)) \implies f + A \subseteq O(g)$
bigo_add_commute_imp	$(f \in g + O(h)) \implies g \in f + O(h)$
bigo_add_commute	$(f \in g + O(h)) \iff g \in f + O(h)$

Il fatto che la relazione di inclusione insiemistica non possa essere sostituita dall'uguaglianza, in proprietà come "*bigo_plus_subset*" risiede in alcune eccezioni che non lo rendono possibile. Nel caso in cui infatti prendessimo in considerazione $g = -f$ le proprietà risulterebbero errate e per questo motivo si vede spesso la necessità di introdurre proprietà più generiche contente l'inclusione ed alcune più specifiche contenti l'uguaglianza. Quest'ultime, applicate a funzioni che soddisfano determinate condizioni, possono aiutare maggiormente in fase di semplificazione.

lemma bigo_plus_self_subset [intro]: "$O(f) + O(f) \subseteq O(f)$"
apply (auto simp add: bigo_alt_def set_plus_def)
apply (rule_tac x = "$c + ca$" **in** exI)
apply auto
apply (simp add: ring_distribs func_plus)
apply (rule order_trans)
apply (rule abs_triangle_ineq)
apply (rule add_mono)
apply force
apply force
done

lemma bigo_plus_idemp [simp]: "$O(f) + O(f) = O(f)$"
apply (rule equalityI)
apply(rule bigo_plus_self_subset)
apply (rule set_zero_plus2)
apply (rule bigo_zero)
done

lemma bigo_plus_subset [intro]: "$O(f+g) \subseteq O(f)+O(g)$"
apply (rule subsetI)
apply (auto simp add: bigo_def bigo_pos_const func_plus set_plus_def)
apply (subst bigo_pos_const [symmetric])+
apply (rule_tac x = "$\lambda n.\ if\ |g\,n| \leq |f\,n|\ then\ x\,n\ else\ 0$" **in** exI)
apply (rule conjI)
apply (rule_tac x = "$c+c$" **in** exI)
apply (clarsimp)
apply (subgoal_tac"$c*|f\,xa+g\,xa| \leq (c+c)*|f\,xa|$")
apply (erule_tac x = xa **in** allE)
apply (erule order_trans)
apply (simp)
apply (subgoal_tac "$c*|f\,xa+g\,xa| \leq c*(|f\,xa|+|g\,xa|)$")
apply (erule order_trans)
apply (simp add: ring_distribs)
apply (rule mult_left_mono)
apply (simp add: abs_triangle_ineq)
apply (simp add: order_less_le)
apply (rule_tac x = "$\lambda n.\ if\ |f\,n| < |g\,n|\ then\ x\,n\ else\ 0$" **in** exI)
apply (rule conjI)
apply (rule_tac x = "$c+c$" **in** exI)
apply auto
apply (subgoal_tac "$c*|f\,xa+g\,xa| \leq (c+c)*|g\,xa|$")
apply (erule_tac x = xa **in** allE)
apply (erule order_trans)
apply simp
apply (subgoal_tac "$c*|f\,xa+g\,xa| \leq c*(|f\,xa|+|g\,xa|)$")
apply (erule order_trans)
apply (simp add: ring_distribs)
apply (rule mult_left_mono)
apply (rule abs_triangle_ineq)

apply (simp add: order_less_le)
done

lemma bigo_plus_subset2 [intro]: "$A \subseteq O(f) \Longrightarrow B \subseteq O(f) \Longrightarrow A + B \subseteq O(f)$"
apply (subgoal_tac "$A + B \subseteq O(f) + O(f)$")
apply (erule order_trans)
apply simp
apply (auto del: subsetI simp del: bigo_plus_idemp)
done

lemma bigo_plus_eq: "$\forall x.\ 0 \leq f\,x \Longrightarrow \forall x.\ 0 \leq g\,x \Longrightarrow O(f+g) = O(f) + O(g)$"
apply (rule equalityI)
apply (rule bigo_plus_subset)
apply (simp add: bigo_alt_def set_plus_def func_plus)
apply clarify
apply (rule_tac x = "*max c ca*" **in** exI)
apply (rule conjI)
apply (subgoal_tac "$c \leq max\ c\ ca$")
apply (erule order_less_le_trans)
apply assumption
apply (rule max.cobounded1)
apply clarify
apply (drule_tac x = "*xa*" **in** in spec)+
apply (subgoal_tac "$0 \leq f\,xa + g\,xa$")
apply (simp add: ring_distribs)
apply (subgoal_tac "$|a\ xa + b\,xa| \leq |a\ xa| + |b\ xa|$")
apply (subgoal_tac "$|a\ xa| + |b\ xa| \leq max\ c\,ca * f\,xa + max\ c\,ca * g\,xa$")
apply force
apply (rule add_mono)
apply (subgoal_tac "$c * f\,xa \leq max\ c\,ca * f\,xa$")
apply force

```
        apply (rule mult_right_mono)
          apply (rule max.cobounded1)
        apply assumption
      apply (subgoal_tac "ca * g xa ≤ max c ca * g xa")
        apply force
      apply (rule mult_right_mono)
        apply (rule max.cobounded2)
      apply assumption
    apply (rule abs_triangle_ineq)
  apply (rule add_nonneg_nonneg)
    apply assumption+
  done

lemma bigo_plus_absorb_lemma1:
  assumes *: "f ∈ O(g)"
  shows "f +o O(g) ⊆ O(g)"
proof −
  have "f ∈ O(f)" by auto
  then have "f +o O(g) ⊆ O(f) + O(g)"
    by (auto del: subsetI)
  also have "... ⊆ O(g) + O(g)"
  proof −
    from * have "O(f) ⊆ O(g)"
      by (auto del: subsetI)
    then show ?thesis
      by (auto del: subsetI)
  qed
  also have "... ⊆ O(g)" by simp
  finally show ?thesis .
qed
```

lemma bigo_plus_absorb_lemma2:
  assumes $*$: "$f \in O(g)$"
  shows "$O(g) \subseteq f +_o O(g)$"
proof −
  from $*$ **have** "$-f \in O(g)$"
    by auto
  then have "$-f +_o O(g) \subseteq O(g)$"
    by (elim bigo_plus_absorb_lemma1)
  then have "$f +_o (-f +_o O(g)) \subseteq f +_o O(g)$"
    by auto
  also have "$f +_o (-f +_o O(g)) = O(g)$"
    by (simp add: set_plus_rearranges)
  finally show ?thesis .
qed

lemma bigo_plus_absorb [simp]: "$f \in O(g) \Longrightarrow f +_o O(g) = O(g)$"
  apply apply (rule equalityI)
    apply (erule bigo_plus_absorb_lemma1)
  apply (erule bigo_plus_absorb_lemma2)
  done

lemma bigo_plus_absorb2 [intro]: "$f \in O(g) \Longrightarrow A \subseteq O(g) \Longrightarrow f +_o A \subseteq O(g)$"
  apply (subgoal_tac "$f +_o A \subseteq f +_o O(g)$")
    apply force+
  done

lemma bigo_add_commute_imp: "$f \in g +_o O(h) \Longrightarrow g \in f +_o O(h)$"
  apply (subst set_minus_plus [symmetric])
  apply (subgoal_tac "$g - f = -(f - g)$")
    apply (erule ssubst)
    apply (rule bigo_minus)

apply (subst set_minus_plus)
apply assumption
apply (simp add: ac_simps)
done

lemma bigo_add_commute: "$f \in g +_o O(h) \iff g \in f +_o O(h)$"
apply (rule iffI)
apply (erule bigo_add_commute_imp)+
done

Come precedentemente annunciato a queste proprietà seguiranno le corrispettive riguardanti la moltiplicazione e poi quelle per la sottrazione. Alcune tra queste non sono altro che la trasposizione degli scenari prima analizzati, in cui però al posto dell'operazione somma troviamo la moltiplicazione. La dimostrazione e quindi l'utilizzo di queste proprietà risulterà rilevante nella verifica dei lemmi contenti molti calcoli.

Nome della Proprietà	**Definizione Matematica**
bigo_mult	$O(f) * O(g) \subseteq O(f * g)$
bigo_mult2	$f * O(g) \subseteq O(f * g)$
bigo_mult3	$(f \in O(h), g \in O(j)) \implies f * g \in O(h * j)$
bigo_mult4	$f \in k + O(h) \implies g * f \in (g * k) + O(g * h)$
bigo_mult5	$\forall x.\ f\,x \neq 0 \implies O(f * g) \subseteq f * O(g)$
bigo_mult6	$\forall x.\ f\,x \neq 0 \implies O(f * g) = f * O(g)$
bigo_mult7	$\forall x.\ f\,x \neq 0 \implies O(f * g) \subseteq O(f) * (O(g)$
bigo_mult8	$\forall x.\ f\,x \neq 0 \implies O(f * g) = O(f) * O(g)$

lemma bigo_mult [intro]: "$O(f) * O(g) \subseteq O(f * g)$"
lemma (rule subsetI)
lemma (subst bigo_def)
lemma (auto simp add: bigo_alt_def set_times_def func_times)
lemma (rule_tac x = "$c * ca$" **in** exI)
apply (rule allI)

apply (erule_tac x = x **in** allE)+
apply (subgoal_tac "$c * ca * |f\,x * g\,x| = (c * |f\,x|) * (ca * |g\,x|)$")
apply (erule ssubst)
apply (subst abs_mult)
apply (rule mult_mono)
apply assumption+
apply auto
apply (simp add: ac_simps abs_mult)
done

lemma bigo_mult2 [intro]: "$f *_o O(g) \subseteq O(f * g)$"
apply auto simp add: bigo_def elt_set_times_def func_times abs_mult)
apply (rule_tac x = c **in** exI)
apply auto
apply (drule_tac x = x **in** spec)
apply (subgoal_tac "$|f\,x| * |b\,x| \leq |f\,x| * (c * |g\,x|)$")
apply (force simp add: ac_simps)
apply (rule mult_left_mono, assumption)
apply (rule abs_ge_zero)
done

lemma bigo_mult3: "$f \in O(h) \Longrightarrow g \in O(j) \Longrightarrow f * g \in O(h * j)$"
apply (rule subsetD)
apply (rule bigo_mult)
apply (erule set_times_intro, assumption)
done

lemma bigo_mult4 [intro]: "$f \in k +_o O(h) \Longrightarrow g * f \in (g * k) +_o O(g * h)$"
apply (drule set_plus_imp_minus)
apply (rule set_minus_imp_plus)
apply (drule bigo_mult3 [**where** g = g **and** j = g])

apply (auto simp add: algebra_simps)
done

lemma bigo_mult5:
fixes f :: "$'a \Longrightarrow 'b :: linordered\_field$"
assumes "$\forall x.\ f\,x \neq 0$"
shows "$O(f * g) \subseteq f *_o O(g)$"
proof
fix h
assume "$h \in O(f * g)$"
then have "$(\lambda x.\ 1 / (f\,x)) * h \in (\lambda x.\ 1 / f\,x) *_o O(f * g)$"
by auto
also have "$\ldots \subseteq O((\lambda x.\ 1 / f\,x) * (f * g))$"
by (rule bigo_mult2)
also have "$(\lambda x.\ 1 / f\,x) * (f * g) = g$"
apply (simp add: func_times)
apply (rule ext)
apply (simp add: assms nonzero_divide_eq_eq ac_simps)
done
finally have "$(\lambda x.\ (1 ::' b) / f\,x) * h \in O(g)$" .
then have "$f * ((\lambda x.\ (1 ::' b) / f\,x) * h) \in f *_o O(g)$"
by auto
also have "$f * ((\lambda x.\ (1 ::' b) / f\,x) * h) = h$"
apply (simp add: func_times)
apply (rule ext)
apply (simp add: assms nonzero_divide_eq_eq ac_simps)
done
finally show "$h \in f *_o O(g)$" .
qed

lemma bigo_mult6: "$\forall x.\ f\,x \neq 0 \implies O(f * g) = f *_o O(g)$"
for f :: "$'a \implies 'b :: linordered\_field$"
apply (rule equalityI)
apply (erule bigo_mult5)
apply (rule bigo_mult2)
done

lemma bigo_mult7: "$\forall x.\ f\,x \neq 0 \implies O(f * g) \subseteq O(f) * O(g)$"
for f :: "$'a \implies 'b :: linordered\_field$"
apply (subst bigo_mult6)
apply assumption
apply (rule set_times_mono3)
apply (rule bigo_refl)
done

lemma bigo_mult8: "$\forall x.\ f\,x \neq 0 \implies O(f * g) = O(f) * O(g)$"
for f :: "$'a \implies 'b :: linordered\_field$"
apply (rule equalityI)
apply (erule bigo_mult7)
apply (rule bigo_mult)
done

Passiamo ora alle proprietà che riguardano la sottrazione:

Nome della Proprietà	**Definizione Matematica**
bigo_minus	$f \in O(g) \implies -f \in O(g)$
bigo_minus2	$f \in g + O(h) \implies -f \in -g + O(h)$
bigo_minus3	$O(-f) = O(f)$

lemma bigo_minus [intro]: "$f \in O(g) \implies -f \in O(g)$"
by (auto simp add: bigo_def fun_Compl_def)

lemma bigo_minus2: "$f \in g +_o O(h) \implies -f \in -g +_o O(h)$"
apply (rule set_minus_imp_plus)
apply (drule set_plus_imp_minus)
apply (drule bigo_minus)
apply simp
done

lemma bigo_minus3: "$O(-f) = O(f)$"
by (auto simp add: bigo_def fun_Compl_def)

Mostrare che una funzione appartenga o meno ad un particolare insieme O-grande è spesso solo una questione di definizioni. A tale scopo i teoremi che seguono offrono delle scorciatoie:

Nome della Proprietà	**Definizione Matematica**
bigo_bounded_alt	$\forall x\ (0 \le f(x), f(x) \le c * g\,x) \implies f \in O(g)$
bigo_bounded	$\forall x\ (0 \le f(x), f(x) \le g\,x) \implies f \in O(g)$
bigo_bounded2	$\forall x\ (g(x) \le f(x), f(x) \le g(x) + h(x)) \implies f \in g + O(h)$

lemma bigo_bounded_alt: "$\forall x.\ 0 \le f\,x \implies \forall x.\ f\,x \le c * g\,x \implies f \in O(g)$"
apply (auto simp add: bigo_def)
apply (rule_tac x = "$|c|$" **in** exI)
apply auto
apply (drule_tac x = x **in** spec)+
apply (simp flip: abs_mult)
done

lemma bigo_bounded: "$\forall x.\ 0 \le f\,x \implies \forall x.\ f\,x \le g\,x \implies f \in O(g)$"
apply (erule bigo_bounded_alt [of f 1 g])
apply simp

done

lemma bigo_bounded2: "$\forall x.\ lb\,x \leq f\,x \implies \forall x.\ f\,x \leq lb\,x + g\,x \implies f \in lb +_o O(g)$"
apply (rule set_minus_imp_plus)
apply (rule bigo_bounded)
apply (auto simp add: fun_Compl_def func_plus)
apply (drule_tac x = x **in** spec)+
apply force
done

Alcuni dei teoremi che vedremo ora verranno presi in considerazioni solamente in anelli ordinati con la proprietà aggiuntiva: $\forall c \neq 0$ esiste un 'd' t.c $cd \geq 1$. Queste condizioni, sotto le quali le proprietà risulteranno corrette, valgono in tutti i campi di nostro interesse ($\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$). Al fine di portare una trattazione più esplicativa verranno anche presentati dei risultati più semplici, che serviranno come step intermedi nella dimostrazioni più complesse. Ad esempio le prime quattro proprietà altro non fanno che dichiarare l'appartenenza di ogni funzioni costante 'c' ad $O(1)$ e la relazione insiemistica esistente tra $O(1)$ e $O(c)$, $\forall c$.
Ciò che accomuna tutte queste proprietà è la parola "*const*" presente nei loro nomi, che palesa l'intenzione di raggruppare le proprietà riguardanti le funzioni costanti. Altro aspetto comune è l'utilizzo delle funzioni lambda per trattare funzioni anonime esattamente come accennato nel capitolo 1.2. Ricordiamo che espressioni del tipo $\lambda x.\ c$ indicano semplicemente una funzione costante c, in quanto il corpo della funzione segue l'espressione "$\lambda x.$" che denota invece la variabile in questione.

Nome della Proprietà	**Definizione Matematica**
bigo_const1	$(\lambda x.\ c) \in O(\lambda x.\ 1)$
bigo_const2	$O(\lambda x.\ c) \subseteq O(\lambda x.\ 1)$
bigo_const3	$c \neq 0 \implies (\lambda x.\ 1) \in O(\lambda x.\ c)$
bigo_const4	$c \neq 0 \implies O(\lambda x.\ 1) \subseteq O(\lambda x.\ c)$
bigo_const	$c \neq 0 \implies O(\lambda x.\ c) = O(\lambda x.\ 1)$
bigo_const_mult1	$(\lambda x.\ c * f\,x) \in O(f)$

bigo_const_mult2	$O(\lambda x.\ c * f\,x) \subseteq O(f)$
bigo_const_mult3	$c \neq 0 \Longrightarrow f \in O(\lambda x.\ c * f\,x)$
bigo_const_mult4	$c \neq 0 \Longrightarrow O(f) \subseteq O(\lambda x.\ c * f\,x)$
bigo_const_mult	$c \neq 0 \Longrightarrow O(\lambda x.\ c * f\,x)) = O(f)$
bigo_const_mult5	$c \neq 0 \Longrightarrow (\lambda x.\ c) * O(f) = O(f)$
bigo_const_mult6	$(\lambda x.\ c) * O(f)) \subseteq O(f)$
bigo_const_mult7	$f = O(g) \Longrightarrow (\lambda x.\ c * f\,x)) = O(g)$

lemma bigo_const1: "$(\lambda x.\ c) \in O(\lambda x.\ 1)$"
  by (auto simp add: bigo_def ac_simps)

lemma bigo_const2 [intro]: "$O(\lambda x.\ c) \subseteq O(\lambda x.\ 1)$"
  apply (rule bigo_elt_subset)
  apply (rule bigo_const1)
  done

lemma bigo_const3: "$c \neq 0 \Longrightarrow (\lambda x.\ 1) \in O(\lambda x.\ c)$"
  for c :: "$'a :: linordered\_field$"
  apply (simp add: bigo_def)
  apply (rule_tac x = "$|inverse\ c|$" **in** exI)
  apply (simp flip: abs_mult)
  done

lemma bigo_const4: "$c \neq 0 \Longrightarrow O(\lambda x.\ 1) \subseteq O(\lambda x.\ c)$"
  for c :: "$'a :: linordered\_field$"
  apply (rule bigo_elt_subset)
  apply (rule bigo_const3)
  apply assumption
  done

lemma bigo_const [simp]: "$c \neq 0 \Longrightarrow O(\lambda x.\ c) = O(\lambda x.\ 1)$"

for c :: "'a :: $linordered\_field$"
apply (rule equalityI)
apply (rule bigo_const2)
apply (rule bigo_const4)
apply assumption
done

lemma bigo_const_mult1: "$(\lambda x.\ c * f\,x) \in O(f)$"
apply simp add: bigo_def)
apply (rule_tac x = "$|c|$" **in** exI)
apply (auto simp flip: abs_mult)
done

lemma bigo_const_mult2: "$O(\lambda x.\ c * f\,x) \subseteq O(f)$"
apply (rule bigo_elt_subset)
apply (rule bigo_const_mult1)
done

lemma bigo_const_mult3: "$c \neq 0 \Longrightarrow f \in O(\lambda x.\ c * f\,x)$"
for c :: "'a :: $linordered\_field$"
apply (simp add: bigo_def)
apply (rule_tac x = "$|inverse\ c|$" **in** exI)
apply (simp add: abs_mult mult.assoc [symmetric])
done

lemma bigo_const_mult4: "$c \neq 0 \Longrightarrow O(f) \subseteq O(\lambda x.\ c * f\,x)$"
for c :: "'a :: $linordered\_field$"
apply (rule bigo_elt_subset)
apply (rule bigo_const_mult3)
apply assumption
done

lemma bigo_const_mult [simp]: "$c \neq 0 \Longrightarrow O(\lambda x.\ c * f\,x) = O(f)$"
  for c :: "'a :: $linordered\_field$"
  apply (rule equalityI)
    apply (rule bigo_const_mult2)
  apply (erule bigo_const_mult4)
  done

lemma bigo_const_mult5 [simp]: "$c \neq 0 \Longrightarrow (\lambda x.\ c) *_o O(f) = O(f)$"
  for c :: "'a :: $linordered\_field$"
  apply (auto del: subsetI)
    apply (rule order_trans)
      apply (rule bigo_mult2)
    apply (simp add: func_times)
  apply (auto intro!: simp add: bigo_def elt_set_times_def func_times)
  apply (rule_tac x = "$\lambda y.\ inverse\ c * x\,y$" **in** exI)
  apply (simp add: mult.assoc [symmetric] abs_mult)
  apply (rule_tac x = "$|inverse\ c| * ca$" **in** exI)
  apply auto
  done

lemma bigo_const_mult6 [intro]: "$(\lambda x.\ c) *_o O(f) \subseteq O(f)$"
  apply (auto intro!: simp add: bigo_def elt_set_times_def func_times)
  apply (rule_tac x = "$ca * |c|$" **in** exI)
  apply (rule allI)
  apply (subgoal_tac "$ca * |c| * |f\,x| = |c| * (ca * |f\,x|)$")
    apply (erule ssubst)
    apply (subst abs_mult)
    apply (rule mult_left_mono)
      apply (erule spec)
    apply simp

apply (simp add: ac_simps)
done

lemma bigo_const_mult7 [intro]:
assumes *: "$f =_o O(g)$"
shows "$(\lambda x.\ c * f\,x) =_o O(g)$"
proof −
from $*$ **have** "$(\lambda x.\ c) * f =_o (\lambda x.\ c) *_o O(g)$"
by auto
also have "$(\lambda x.\ c) * f = (\lambda x.\ c * f\,x)$"
by (simp add: func_times)
also have "$(\lambda x.\ c) *_o O(g) \subseteq O(g)$"
by (auto del: subsetI)
finally show ?thesis .
qed

Le ultime proprietà base presentate nella trattazione di Avigad e Donnelly [6] e che saranno ora introdotte, riguardano applicazioni più specifiche della notazione O-grande, ricordando che con l'espressione $f = O(g)$ s'intende $f \in O(g)$.
Nello scritto originale si faceva riferimento ad una funzione "*sumr*", inclusa nella libreria HOL-Complex, che permetteva di svolgere calcoli del tipo $\sum_{m \le i < n} f(i)$ semplicemente scrivendo *sumr m n f*. In questa notazione *m* e *n* sono elementi di $\mathbb{N}$ ed f deve essere una funzione definita da $\mathbb{N}$ a $\mathbb{R}$.
Con la versione finale di queste teorie, ad oggi parte della libreria ufficiale di Isabelle, molte delle vecchie proprietà sono state modificate ed in questo caso, ad esempio, la funzione *sumr* è stata sostituita con il simbolo esplicito di sommatoria, che troveremo sempre in queste formulazioni.

Nome della Proprietà	**Definizione Matematica**
bigo_sum_main	$\exists c\,\forall x\,\forall y \in A\,x\ (0 \le h\,x\,y, \lvert f\,x\,y\rvert \le c * h\,x\,y) \Longrightarrow$ $\lambda x.\ \sum_{y \in A\,x}\ f\,x\,y = O(\lambda x.\ \sum_{y \in A\,x}\ h\,x\,y)$
bigo_sum1	$\exists c\,\forall x,y\ (0 \le h\,x\,y, \lvert f\,x\,y\rvert \le c * h\,x\,y) \Longrightarrow$ $\lambda x.\ \sum_{y \in A\,x}\ f\,x\,y = O(\lambda x.\ \sum_{y \in A\,x}\ h\,x\,y)$

bigo_sum2	$\exists c\, \forall y\ (0 \leq hy, \lvert fy\rvert \leq c * hy) \Longrightarrow$ $\lambda x.\ \sum_{y \in Ax}\ fy = O(\lambda x.\ \sum_{y \in Ax}\ hy)$
bigo_sum3	$f = O(h) \Longrightarrow$ $\lambda x.\ \sum_{y \in Ax}\ lxy * f(kxy) = O(\lambda x.\ \sum_{y \in Ax}\ \lvert lxy * h(kxy)\rvert)$
bigo_sum4	$f = g + O(h) \Longrightarrow \lambda x.\ \sum_{y \in Ax}\ lxy * f(kxy) =$ $\lambda x.\ \sum_{y \in Ax}\ lxy * g(kxy) + O(\lambda x.\ \sum_{y \in Ax}\ \lvert lxy * h(kxy)\rvert)$
bigo_sum5	$\forall x,y\ (f = O(h),\ 0 \leq lxy,\ 0 \leq hx) \Longrightarrow$ $\lambda x.\ \sum_{y \in Ax}\ lxy * f(kxy) = O(\lambda x.\ \sum_{y \in Ax}\ lxy * h(kxy))$
bigo_sum6	$\forall x,y\ (f = g + O(h),\ 0 \leq lxy,\ 0 \leq hx) \Longrightarrow$ $\lambda x.\ \sum_{y \in Ax}\ lxy * f(kxy) =$ $\lambda x.\ \sum_{y \in Ax}\ lxy * g(kxy) + O(\lambda x.\ \sum_{y \in Ax}\ lxy * h(kxy))$

lemma bigo_sum_main: "$\forall x.\, \forall y \in Ax.\ 0 \leq hxy \Longrightarrow$
$\exists c.\ \forall x.\ \forall y \in Ax.\ \lvert fxy\rvert \leq c * hxy \Longrightarrow$
$(\lambda x.\ \sum y \in Ax.\ fxy) =_o O(\lambda x.\ \sum y \in Ax.\ hxy)$"

apply (auto simp add: bigo_def)
apply (rule_tac x = "$\lvert c\rvert$" **in** exI)
apply (subst abs_of_nonneg) back back
apply (rule sum_nonneg)
apply force
apply (subst sum_distrib_left)
appply (rule allI)
apply (rule order_trans)
apply (rule sum_abs)
apply (rule sum_mono)
apply (rule order_trans)
apply (drule spec)+
apply (drule bspec)+
apply assumption+
apply (drule bspec)
apply assumption+
apply (rule mult_right_mono)

apply (rule abs_ge_self)
apply force
done

lemma bigo_sum1: "$\forall x\, y.\ 0 \leq h\, x\, y \Longrightarrow$
$\exists c.\ \forall x\, y.\ |f\, x\, y| \leq c * h\, x\, y \Longrightarrow$
$(\lambda x.\ \sum y \in A\, x.\ f\, x\, y) =_o O(\lambda x.\ \sum y \in A\, x.\ h\, x\, y)$"
apply (rule bigo_sum_main)
apply force
apply clarsimp
apply (rule_tac x = c **in** exI)
apply force
done

lemma bigo_sum2: "$\forall y.\ 0 \leq h\, y \Longrightarrow$
$\exists c.\ \forall y.\ |f\, y| \leq c * (h\, y) \Longrightarrow$
$(\lambda x.\ \sum y \in A\, x.\ f\, y) =_o O(\lambda x.\ \sum y \in A\, x.\ h\, y)$"
by (rule bigo_sum1) auto

lemma bigo_sum3: "$f =_o O(h) \Longrightarrow$
$(\lambda x.\ \sum y \in A\, x.\ l\, x\, y * f\, (k\, x\, y)) =_o O(\lambda x.\ \sum y \in A\, x.\ |l\, x\, y * h\, (k\, x\, y)|)$"
apply (rule bigo_sum1)
apply (rule allI)+
apply (rule abs_ge_zero)
apply (unfold bigo_def)
apply auto
apply (rule_tac x = c **in** exI)
apply (rule allI)+
apply (subst abs_mult)+
apply (subst mult.left_commute)
apply (rule mult_left_mono)

apply (erule spec)
apply (rule abs_ge_zero)
done

lemma bigo_sum4: "$f =_o g +_o O(h) \Longrightarrow$
$(\lambda x.\ \sum y \in A\, x.\ l\, x\, y * f\,(k\, x\, y)) =_o$
$(\lambda x.\ \sum y \in A\, x.\ l\, x\, y * g\,(k\, x\, y)) +_o$
$O(\lambda x.\ \sum y \in A\, x.\ |l\, x\, y * h\,(k\, x\, y)|)$"
apply (rule set_minus_imp_plus)
apply (subst fun_diff_def)
apply (subst sum_subtractf [symmetric])
apply (subst right_diff_distrib [symmetric])
apply (rule bigo_sum3)
apply (subst fun_diff_def [symmetric])
apply (erule set_plus_imp_minus)
done

lemma bigo_sum5: "$f =_o O(h) \Longrightarrow \forall x\, y.\ 0 \leq l\, x\, y \Longrightarrow$
$\forall x.\ 0 \leq h\, x \Longrightarrow$
$(\lambda x.\ \sum y \in A\, x.\ l\, x\, y * f\,(k\, x\, y)) =_o$
$O(\lambda x.\ \sum y \in A\, x.\ l\, x\, y * h\,(k\, x\, y))$"
apply (subgoal_tac "$(\lambda x.\ \sum y \in A\, x.\ l\, x\, y * h\,(k\, x\, y)) =$
$(\lambda x.\ \sum y \in A\, x.\ |l\, x\, y * h\,(k\, x\, y)|)$")
apply (erule ssubst)
apply (erule bigo_sum3)
apply (rule ext)
apply (rule sum.cong)
apply (rule refl)
apply (subst abs_of_nonneg)
apply auto
done

lemma bigo_sum6: "$f =_o g +_o O(h) \Longrightarrow \forall x\,y.\ 0 \leq l\,x\,y \Longrightarrow$
$\forall x.\ 0 \leq h\,x \Longrightarrow$
$(\lambda x.\ \sum y \in A\,x.\ l\,x\,y * f\,(k\,x\,y)) =_o$
$(\lambda x.\ \sum y \in A\,x.\ l\,x\,y * g\,(k\,x\,y)) +_o$
$O(\lambda x.\ \sum y \in A\,x.\ l\,x\,y * h\,(k\,x\,y))$"

apply (rule set_minus_imp_plus)
apply (subst fun_diff_def)
apply (subst sum_subtractf [symmetric])
apply (subst right_diff_distrib [symmetric])
apply (rule bigo_sum5)
apply (subst fun_diff_def [symmetric])
apply (drule set_plus_imp_minus)
apply auto
done

In questo capitolo sono state trattate molte delle proprietà presenti nella libreria e che concorrono alla formalizzazione in Isabelle della notazione O-grande. In particolare siamo partiti delle proprietà proposte nel pdf di Avigad e Donnelly del 2004 che sono state mostrate nella loro versione corrente. Alcune sono rimaste simili, altre identiche, mentre molte hanno subito modifiche sostanziali. Nell'ottica di fare una trattazione il più completa possibile, sono stati presentati anche lemmi secondari, non nominati nel pdf originale, ma parte delle dimostrazione di alcune proprietà. Nel caso in cui si volesse constatare con mano quanto affermato si può far riferimento a [5] per ottenere l'elenco di tutte le sessioni che fanno parte della libreria HOL, oppure ad [8] per visionare direttamente la teoria BigO.

Conclusione

Ad oggi gli studi che riguardano i tempi di calcolo e la complessità algoritmica vengono svolti con ragionamenti informali, che non richiedono l'utilizzo di dimostratori come Isabelle. Le implicazioni dirette dell'utilizzo di questa formalizzazione potranno però permettere di controllare ennesime varianti di ragionamenti ricorrenti per evitare errori dovuti alla moltiplicazione del lavoro. Ricordando inoltre che Isabelle basa tutta la sua libreria su costruzioni incrementali, in cui ogni nuova teoria deve poggiare su quelle precedenti, allo stesso modo quelle presentate in questa trattazione potranno in futuro risultare tasselli fondamentali di compiti sempre di maggiore rilevanza.
Nonostante la completezza delle conclusioni a cui si è arrivati dopo anni di continue modifiche della proposta originaria di Avigad e Donnelly, una revisione è ancora possibile. Nello specifico in base a quanto constatato dal professor Berardi e dal sottoscritto, le istruzioni usate per dimostrare i lemmi in Isabelle nella formulazione classica corrente risultano spesso di difficile comprensione senza l'utilizzo dell'applicativo che mostra i subgoals necessari. Sebbene questa problematica sia molto estesa e affligga la maggior parte dei lemmi delle teorie contenute nella libreria, il problema potrebbe essere parzialmente risolto andando ad utilizzare ovunque sia possibile dimostrazioni con Isar. Si può constatare con mano questa problematica osservando la dimostrazione realizzata con Isar di pagina 34. Mentre un lemma strutturato in questa maniera risulta immediatamente comprensibile, per capire a fondo la dimostrazione di un lemma classico come "set_plus_rearrange3", preso a pagina 28 come esempio, siamo stati costretti ad introdurre una tabella nella pagina successiva che evidenziasse l'output dell'esecuzione di ogni istruzione. Tutto questo è causato dalla possibilità di applicare codice Isabelle per semplificare il subgoal iniziale in più subgoals diversi di più facile risoluzione, la cui formulazione prevede però spesso l'introduzione automatica di nuove variabili. Dal momento che si possono applicare regole logiche direttamente a variabili prese dai subgoals presentati, si può far riferimento a quelle

nuove variabili appena introdotte. Queste variabili non facenti parte della formulazione originaria del lemma saranno di difficile comprensione per il lettore.
Un possibile sviluppo futuro di questa tesi potrebbe essere quindi il tentativo di riscrivere i lemmi qua presentati utilizzando però Isar come risolutore.

Bibliografia

[1] Thomas H. Cormen, Charles E. Leiserson e Ronald L. Rivest. *Introduzione agli algoritmi e strutture dati*. McGraw-Hill Education, terza edizione, 2010.

[2] Giulia Maria Piacentini. *Algebra un approccio algoritmico*. Zanichelli, 1996.

[3] Tobias Nipkow, Gerwin Klein. *Concrete Semantics with Isabelle/HOL*. Springer-Verlag, 2019.

[4] Carlo Domenico Pagani, Sandro Salsa. *Analisi matematica 1*. Zanichelli, seconda edizione, 2015.

[5] Libreria HOL:
`https://isabelle.in.tum.de/library/HOL`

[6] Jeremy Avigad e Kevin Donnelly. *Formalizing O notation in Isabelle/HOL*. In D. Basin and M. Rusiowitch, editors, Automated Reasoning: second international conference, IJCAR 2004, pages 357–371. Springer, 2004.

[7] R. L. Graham, D. E. Knuth, O. Patashnik. *Concrete mathematics, a foundation for computer-science*. Addison-Welsey, second edition, 1994.

[8] BigO Theory:
`https://isabelle.in.tum.de/library/HOL/HOL-Library/BigO.html`

[9] Tobias Nipkow, Lawrence C. Paulson, Markus Wenzel. *A Proof Assistant for Higher-Order Logic*. Springer-Verlag, 2008.