UNIVERSIDADE da MADEIRA

Departamento de Matemática e Engenharias

Teoria de conjuntos em Isabelle

Helena Paula Nunes Araújo

(Licenciada)

Dissertação para a obtenção do Grau de Mestre em Matemática
(Área de especialização de Matemática para o ensino)

Orientador: Professor Dr. Francisco Miguel A. C. de Sousa Dionísio

Funchal, Julho de 2006

Resumo

A teoria de conjuntos é o estudo da associação entre objectos com uma mesma propriedade, ou seja pertencentes ao mesmo conjunto, utilizando uma notação precisa e definindo uma série de operações e propriedades desses objectos.

Esta teoria teve a sua primeira axiomatização por volta de 1907 com Ernest Zermelo. Desta axiomatização fazem parte os axiomas da extensionalidade, da especificação, do emparelhamento, da união, da potência, do infinito, da escolha e da substituição.

Toda esta axiomatização constitui a primeira parte deste trabalho e baseia-se naquilo que Paul Halmos apresenta no livro "Naive Set Theory".

Ao longo da construção desta teoria surgem propriedades que se podem provar da forma clássica (utilizando o lápis e o papel). Contudo, pode-se utilizar um ambiente de prova denominado *Isabelle* que permite fazer estas provas, quer de forma assistida, quer automática. Neste ambiente está disponível uma representação da teoria de Zermelo-Fraenkel (ZF) como extensão da lógica de primeira ordem FOL (First-Order Logic). É utilizando esta representação que se demonstram algumas dessas propriedades, na segunda parte, apresentando as tácticas e tacticais disponíveis para o efeito.

Palavras Chave

Teoria de Conjuntos; Sistema de Zermelo-Fraenkel; Axiomática; Demostração Automática; Isabelle; Tácticas e tacticais;

Abstract

Set theory is the study of the association between objects sharing one property, that is belonging to the same set, using a precise notation and defining operations and properties of these objects. This theory had its first axiomatization around 1907 with Ernest Zermelo. This axiomatization includes the axioms of extension, specification, pairing, union, power, infinite, choice and substitution.

The first part of this thesis presents this axiomatization and its properties and is based in Paul Halmos book "Naive Set Theory".

The second part of the thesis is dedicated to reproducing the proofs of properties of set theory in computer form. For that purpose the proof environment Isabelle is used. In Isabelle a representation of Zermelo-Fraenkel (ZF) theory is available as an extension of the First-Order Logic (FOL). Proofs are carried in Isabelle/ZF and are, in general, interactive (assisted) but some can be established authomatically. The tactics and tacticals to build proofs and proof strategies are presented.

Key Words

Set Theory; Sistem of Zermelo-Fraenkel ; Axiomatic; Automatic Demonstration; Isabelle; Tactics and tacticals.

Agradecimentos

À Universidade da Madeira (UMa), em particular ao seu Departamento de Matemática e Engenharias (DME).

Ao Professor Doutor Francisco Miguel A. C. de Sousa Dionísio, meu orientador, pelo seu incentivo, apoio e orientação constantes. Pela confiança que sempre demonstrou no meu trabalho.

A todos os professores que leccionaram as cadeiras da parte curricular do meu mestrado.

À Direcção Executiva e todos os meus colegas da Escola Básica dos 2º e 3º Ciclos do Caniço, por todo o apoio manifestado durante a elaboração desta tese.

Ao Rafael pelo seu incansável e carinhoso apoio.

À minha família, pela amizade, apoio e dedicação.

Índice

Introdução

O presente trabalho foi elaborado como dissertação do 2º ano do Mestrado em Matemática (área de especialização de Matemática para o ensino), da Universidade da Madeira. O tema proposto foi teoria de conjuntos em *Isabelle*.

Este trabalho está dividido em duas partes. A primeira contém uma parte da teoria de conjuntos apresentada por Paul Halmos no livro "Naive Set Theory".

A teoria dos conjuntos é a teoria Matemática que trata das propriedades dos conjuntos. Esta teoria surgiu na parte final do século XIX, com o matemático Georg Cantor (1845 - 1918) e baseia-se na ideia de definir conjunto como uma noção primitiva.

Inicialmente no desenvolvimento desta teoria lidava-se com uma noção intuitiva de conjunto. Estes eram considerados como um agregado de elementos que verificavam uma determinada condição, ou seja, se φ é uma condição, então existe um conjunto cujos elementos verificam φ. Este é o *princípio da abstracção*. Contudo, o uso deste sem restrições originou alguns paradoxos, nomeadamente, o paradoxo de Russel. A partir desta altura deu-se por concluída a fase ingénua da teoria dos conjuntos e deu-se início a uma busca de princípios (axiomas) subjacentes à formação de conjuntos. A primeira axiomatização da teoria dos conjuntos deve-se a Ernest Zermelo (1871-1953) por volta de 1907.

O primeiro dos axiomas de Zermelo, o axioma da extensionalidade, fornece uma condição necessária e suficiente para a igualdade entre dois conjuntos. Segue-se o axioma da especificação que, dada uma condição φ, garante que para todo o conjunto A existe um conjunto B formado pelos elementos de A que verificam φ. Ou seja, é possível separar os elementos de A em dois conjuntos: o conjunto de elementos que satisfazem φ e o conjunto dos elementos de A que não satisfazem φ. Ao contrário do princípio da abstracção que leva a contradições, o axioma anterior evita-as, pois limita a formação do conjunto B ao conjunto A.

O axioma da especificação só é eficaz se houver à partida muitos destes conjuntos A, ou seja, só se tem realmente uma teoria de conjuntos se se assegurar a existência de um número razoável de conjuntos à partida. É esse o papel dos chamados axiomas de existência da teoria de Zermelo-Fraenkel (ZF). São eles os seguintes: axioma do emparelhamento, o axioma da união, axioma da potência e o axioma do infinito. Os três primeiros permitem respectivamente formar (com o auxílio do axioma da especificação) os conjuntos $\{a, b\}$, $\cup\mathcal{C}$, $\mathcal{P}(x)$. Por sua vez, o axioma do infinito permite formar o conjunto w dos números naturais. Desta axiomatização fazem ainda parte os axiomas da escolha e da substituição. O primeiro destes garante que o produto cartesiano de uma família não vazia de conjuntos não vazios é não vazio. O axioma da substituição garante que a imagem de um conjunto por uma função é

ainda um conjunto.

Na segunda parte, apresenta-se a demonstração de algumas das propriedades mais importantes referidas ao longo da primeira parte, com recurso a um sistema genérico de prova denominado *Isabelle*, que inclui uma representação de ZF (que é uma extensão de FOL– First-Order Logic).

Capítulo 1

Teoria ingénua de conjuntos

1.1 Introdução

A teoria dos conjuntos desenvolveu-se intensamente, de tal forma que pode dizer-se que todos os ramos da Matemática foram influenciados e enriquecidos por esta teoria. A linguagem matemática utiliza constantemente termos e notações da teoria dos conjuntos, sendo possível definir grande parte dos conceitos matemáticos à custa dos conjuntos. Neste capítulo pretende-se ilustrar precisamente este facto, ou seja, apresenta-se o modo de definir alguns objectos matemáticos, tais como pares ordenados, produto cartesiano, números naturais, usando conceitos da teoria de conjuntos.

1.2 Axioma da extensionalidade

Um dos principais conceitos primitivos[1] da teoria de conjuntos é o conceito de conjunto. Este é de forma intuitiva visto como uma colecção de objectos quaisquer, os quais são chamados elementos ou membros desse conjunto. Um conjunto pode ele próprio ser um elemento de um outro conjunto, neste caso usar-se-á a designação *colecção de conjuntos.*

Além do conceito de conjunto e de elemento, o conceito de pertença é também um conceito primitivo da teoria de conjuntos.

Se x *pertence* a A, ou seja, se x é um elemento de A, onde A é um conjunto, escreve-se simbolicamente

$$x \in A.$$

A negação desta afirmação, x *não pertence* a A é normalmente representada por

$$x \notin A.$$

Uma relação possível entre conjuntos é a *igualdade*, que é denotada por

$$A = B.$$

[1] Um conceito primitivo é um conceito que não é definido à custa de outros, mas que se admite saber o que significa.

Se os conjuntos A e B *não são iguais* escreve-se

$$A \neq B.$$

A igualdade de conjuntos é definida como se segue.

Axioma 1.1 ***(da extensionalidade)*** *- Dois conjuntos A e B são iguais se e só se tiverem exactamente os mesmos elementos, isto é,*

$$(A = B) \Leftrightarrow (\forall_x \ (x \in A \Leftrightarrow x \in B)).$$

Definição 1.2 *Sejam A e B dois conjuntos quaisquer, diz-se que A está contido em B $(A \subseteq B)$, ou que A é um subconjunto de B, ou ainda, que B contém A $(B \supseteq A)$, se e só se todo o elemento de A é um elemento de B, ou seja, $A \subseteq B \Leftrightarrow \forall x$ se $x \in A$ então $x \in B$.*

De acordo com a definição tem-se que qualquer conjunto A é subconjunto de si próprio, ou seja, $A \subseteq A$, que é o mesmo que dizer que a relação "está contido" é *reflexiva.* O mesmo acontece com a igualdade de conjuntos $(A = A)$.

Definição 1.3 *Se A e B são dois conjuntos quaisquer tais que $A \subseteq B$ e $A \neq B$, então diz-se que A está estritamente contido em B, ou seja, A é uma parte própria (ou um subconjunto próprio) de B e que se denota por $A \subset B$.*

Se A, B e C são conjuntos tais que $A \subseteq B$ e $B \subseteq C$, então $A \subseteq C$, que é o mesmo que dizer que a relação "está contido" é *transitiva.* Esta propriedade também é válida no caso da igualdade de conjuntos (se $A = B$ e $B = C$ então $A = C$).

Se A e B são dois conjuntos tais que $A \subseteq B$ e $B \subseteq A$, então A e B têm os mesmos elementos. Logo pelo axioma da extensionalidade $A = B$, que é o mesmo que dizer que a relação "está contido" é *anti-simétrica.* O axioma da extensionalidade pode então ser reformulado do seguinte modo: se A e B são conjuntos, então uma condição necessária e suficiente para se ter $A = B$ é verificar-se simultaneamente $A \subseteq B$ e $B \subseteq A$.

No caso da igualdade tem-se que se $A = B$, então necessariamente $B = A$, ou seja, a igualdade é *simétrica.*

1.3 Axioma da especificação

Todos os axiomas básicos da teoria dos conjuntos, excepto o axioma da extensionalidade, são utilizados para construir novos conjuntos a partir de outros.

A primeira sugestão de definição de conjuntos admitia que sendo φ uma propriedade, então existia o conjunto dos elementos que verificavam φ, ou seja, $\{x : \varphi(x)\}$. No entanto, esta definição pode levar a paradoxos. Por exemplo, considere-se que φ é a condição $x \notin x$ e suponha-se que existe um conjunto $A = \{x : \varphi(x)\}$. Se $A \in A$, então verifica-se $\varphi(A)$ e portanto $A \notin A$. Se $A \notin A$, então A não satisfaz a condição $\varphi(x)$ e logo $A \in A$. Obtém-se assim, que $A \in A$ se e só se $A \notin A$ o

que é uma contradição. Este exemplo ficou conhecido como paradoxo de Russel[2]. Esta definição assume implicitamente a existência de um universo global (conjunto de todos os objectos), o paradoxo de Russel mostra que não se pode assumir a existência desse universo global. A forma de resolver esta questão é a seguinte: dado um conjunto A e uma propriedade φ é possível definir o subconjunto dos elementos de A que satisfazem φ ($\{x \in A \text{ e } \varphi(x)\}$). Este princípio é o axioma seguinte.

Axioma 1.4 ***(da especificação[3])*** *- A todo o conjunto A e a toda a condição $S(x)$ corresponde um conjunto B cujo os elementos são exactamente os elementos x de A que verificam $S(x)$, ou seja, se $S(x)$ é uma condição, então para qualquer A existe um conjunto $B = \{x \in A : S(x)\}$.*

1.4 Pares não ordenados

A partir de agora assume-se que existe de facto um conjunto[4].

Uma consequência de se ter assumido a existência de um conjunto é a de que existe um conjunto sem elementos. De facto, se A é um conjunto, aplicando o axioma da especificação a A com a condição "$x \neq x$" (ou uma outra condição universalmente falsa), o resultado é o conjunto $\{x \in A : x \neq x\}$ e rapidamente se verifica que este não contém elementos. Pelo axioma da extensionalidade tem-se que existe apenas um conjunto nestas condições, ou seja, existe apenas um conjunto sem elementos. Este conjunto é denominado *conjunto vazio* e representa-se por $\varnothing$ ou $\{\}$.

Facilmente, se verifica que o conjunto vazio é subconjunto de todos os conjuntos, simbolicamente

$$\forall_A \quad \varnothing \subseteq A.$$

Para provar esta propriedade basta pensar do seguinte modo. Quer-se provar que todo o elemento do conjunto vazio pertence ao conjunto A, mas uma vez que não existem elementos no conjunto vazio, então a condição é satisfeita.

A teoria dos conjuntos desenvolvida até este momento tem apenas um conjunto e este é vazio. Assim, é necessário um novo axioma que permita construir novos conjuntos.

---

[2] Este paradoxo aparece reformulado pelo próprio Russel, numa versão mais popular. Considere-se uma pequena aldeia que possui um único barbeiro, que obedece à seguinte regra: ele faz a barba de todos os homens da cidade que não se barbeiam e somente a esses.

Considere-se agora a seguinte pergunta: quem faz a barba do barbeiro?

É intrigante considerar que, numa tal aldeia não pode existir um barbeiro assim, pois, se ele se barbear, estará fazendo a barba de alguém que se barbeia, o que não satisfaz a regra, ao passo que, se ele não se barbear, ele terá que ter a barba feita por si próprio, o que também vai contra o estabelecido.

[3] A designação deste axioma é tradução do termo em inglês "specification" utilizado por Halmos. Em português é também denominado axioma da separação.

[4] De facto a existência de um conjunto é imposta pelo axioma 1.44 (do infinito), mais à frente.

Axioma 1.5 ***(do emparelhamento[5])*** - *Para quaisquer dois conjuntos existe sempre um conjunto ao qual ambos pertencem.*

Uma das consequências deste axioma é que para dois conjuntos quaisquer existe um conjunto que os contém como únicos elementos. De facto, se a e b são conjuntos e se A é um conjunto tal que $a \in A$ e $b \in A$, então aplicando o axioma da especificação ao conjunto A, com a condição "$x = a$ ou $x = b$" o resultado é o conjunto

$$\{x \in A : x = a \text{ ou } x = b\}.$$

Como é evidente, este conjunto contém apenas a e b. O axioma da extensionalidade garante que existe apenas um conjunto com esta propriedade. Normalmente, este conjunto é representado por

$$\{a, b\}$$

e denominado *par* ou *par não ordenado* formado por a e b.

Se a é um conjunto, então pode-se formar o par $\{a, a\}$ que é representado por

$$\{a\}$$

e denominado *conjunto singular*[6]. Este conjunto é caraterizado pela condição de que a é o seu único elemento. Daqui conclui-se que, por exemplo, $\varnothing$ e $\{\varnothing\}$ são conjuntos muito diferentes, o conjunto $\varnothing$ não tem elementos e o conjunto $\{\varnothing\}$ tem um único elemento, o $\varnothing$.

O axioma do emparelhamento assegura que todo o conjunto é um elemento de algum outro conjunto e que dois conjuntos são simultaneamente elementos de um mesmo conjunto. Além disso, este axioma garante a existência de muitos conjuntos. Como exemplos considere-se os conjuntos $\varnothing$, $\{\varnothing\}$, $\{\{\varnothing\}\}$, $\{\{\{\varnothing\}\}\}$, ... e os pares tais como $\{\varnothing, \{\varnothing\}\}$, formados por quaisquer dois dos conjuntos anteriores. Pode-se ainda formar outros pares a partir dos pares anteriores $(\{\{\varnothing, \{\varnothing\}\}, \{\{\varnothing\}, \{\{\varnothing\}\}\}\})$ ou misturando um par com um conjunto singular qualquer $(\{\varnothing, \{\varnothing, \{\varnothing\}\}\})$.

[5] A designação deste axioma é tradução do termo em inglês "pairing" utilizado por Halmos. Em português é também denominado axioma dos pares.

[6] Note-se que $\{a, a\}$ representa o mesmo que $\{a\}$, pois pelo axioma da extensionalidade o que realmente interessa são os elementos de cada conjunto.

1.5 Uniões e intersecções

União

Por vezes, é útil juntar todos os elementos de dois ou mais conjuntos num só conjunto. O axioma seguinte permite construir esse tipo de conjunto.

Axioma 1.6 ***(da união)*** *- para toda a colecção de conjuntos existe um conjunto que contém todos os elementos que pertencem a pelo menos um dos conjuntos dessa colecção de conjuntos.*

Por outras palavras, o axioma da união diz que para toda a colecção de conjuntos $\mathcal{C}$ existe um conjunto A tal que se $x \in X$ para algum X em $\mathcal{C}$, então $x \in A$. Mas o conjunto A pode conter elementos que não pertençam a nenhum dos conjuntos X da colecção $\mathcal{C}$. Como o que se pretende é garantir que A tenha apenas elementos que pertençam a algum dos conjuntos X de $\mathcal{C}$, aplica-se o axioma da especificação para formar o seguinte conjunto

$$\{x \in A : x \in X \text{ para algum } X \in \mathcal{C}\}.$$

Deste modo, verifica-se que uma condição necessária e suficiente para que x pertença a este conjunto é que x pertença a X, para algum X pertencente à colecção $\mathcal{C}$. Se se representar o novo conjunto por U tem-se,

$$U = \{x : x \in X \text{ para algum } X \in \mathcal{C}\}.$$

O conjunto U é denominado *união* da colecção de conjuntos $\mathcal{C}$, que poderá ser representado por:

$$\begin{aligned} U &= \cup\mathcal{C} \\ &\text{ou} \\ U &= \cup\{X : X \in \mathcal{C}\} \\ &\text{ou} \\ U &= \cup_{X\in\mathcal{C}} X. \end{aligned}$$

Note-se que, o axioma da extensionalidade garante a unicidade deste conjunto.

Notação 1.7 - $\cup\{X : X \in \{A, B\}\} = A \cup B$, *ou seja,* $x \in A \cup B$ *se e só se* x *pertence a* A *ou a* B, *ou a ambos. Assim, tem-se*

$$A \cup B = \{x : x \in A \text{ ou } x \in B\}.$$

Em seguida, apresenta-se uma série de propriedades cujas demonstrações se deixa ao cuidado do leitor.

Algumas dessas demonstrações encontram-se feitas em *Isabelle* na subsecção 2.3.2. As restantes estão na subsecção A.2.1 do apêndice.

Propriedades da união de conjuntos

Sejam A, B e C conjuntos quaisquer.

1. $\cup\{X : X \in \varnothing\} = \varnothing$ ou de forma mais simples, $\cup\varnothing = \varnothing$;
2. $\cup\{X : X \in \{A\}\} = A$ ou de forma mais simples, $\cup\{A\} = A$;
3. $A \cup \varnothing = A$;
4. $A \cup B = B \cup A$ (comutativa);
5. $A \cup (B \cup C) = (A \cup B) \cup C$ (associativa);
6. $A \cup A = A$ (idempotencia);
7. $A \subseteq A \cup B$;
8. $A \subseteq B \Leftrightarrow A \cup B = B$.

Intersecção

Definição 1.8 *Sejam A e B dois conjuntos quaisquer, a intersecção de A e B é o conjunto*

$$A \cap B = \{x \in A : x \in B\} = \{x \in B : x \in A\}.$$

Assim, $x \in A \cap B$ se e só se $x \in A$ e $x \in B$, deste modo tem-se

$$A \cap B = \{x : x \in A \text{ e } x \in B\}.$$

Propriedades da intersecção de conjuntos

1. $A \cap \varnothing = \varnothing$;
2. $A \cap B = B \cap A$ (comutativa);
3. $A \cap (B \cap C) = (A \cap B) \cap C$ (associativa);
4. $A \cap A = A$; (idempotencia)
5. $A \subseteq B \Leftrightarrow A \cap B = A$;
6. $A \subseteq B \wedge A \subseteq C \rightarrow A \subseteq B \cap C$.

As provas destas propriedades ficam ao cuidado do leitor.
Algumas dessas demonstrações encontram-se feitas em *Isabelle* na subsecção 2.3.2. As restantes estão na subsecção A.2.2 do apêndice.

É usual designar por conjuntos *disjuntos* os conjuntos cuja intersecção é vazia.

Propriedades distributivas

1. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$;
2. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.

As provas em *Isabelle* destas propriedades encontram-se na subsecção A.2.3 do apêndice.

Para generalizar a operação de intersecção a classes há que mostrar que dada uma classe $\mathcal{C}$, existe o conjunto que contém exactamente aqueles elementos que pertencem a todos os conjuntos de $\mathcal{C}$. Note-se que esta intersecção não está definida se $\mathcal{C} = \varnothing$. Ou seja, para cada colecção $\mathcal{C}$, diferente de $\varnothing$, existe um conjunto V tal que $x \in V$ se e só se $x \in X$ para todo o X em $\mathcal{C}$. Para provar esta asserção, considere-se um conjunto A qualquer de $\mathcal{C}$ ($\mathcal{C} \neq \varnothing$) e

$$V = \{x \in A : x \in X \text{ para todo } X \text{ em } \mathcal{C}\}.$$

Seja qual for a escolha do conjunto A obtém-se o mesmo conjunto V.

O conjunto V é chamado *intersecção da colecção de conjuntos* C e o axioma da extensionalidade garante a sua unicidade. O conjunto V é usualmente representado por,

$$\cap\{X : X \in \mathcal{C}\}$$
ou
$$\cap_{X \in \mathcal{C}} X.$$

1.6 Complementares

Definição 1.9 *Se A e B são conjuntos, a diferença entre A e B ou como é mais conhecido complementar de B em A é o conjunto $A - B$ definido por*

$$A - B = \{x \in A : x \notin B\}.$$

Assume-se que todos os conjuntos mencionados nesta secção são subconjuntos de E e que todos os complementares serão formados com base em E.

O complementar do conjunto A será representado por A'.

Em seguida, apresenta-se uma série de propriedades cujas demonstrações se deixa ao cuidado do leitor.

Algumas dessas demonstrações encontram-se feitas em *Isabelle* na subsecção 2.3.3. As restantes estão na subsecção A.2.4 do apêndice.

Propriedades

1. $(A')' = A$;
2. $\varnothing' = E$;
3. $E' = \varnothing$;
4. $A \cap A' = \varnothing$;
5. $A \cup A' = E$;
6. $A \subseteq B \Leftrightarrow B' \subseteq A'$;
7. $(A \cup B)' = A' \cap B'$ (lei de De Morgan);
8. $(A \cap B)' = A' \cup B'$ (lei de De Morgan);
9. $A - (B \cup C) = (A - B) \cap (A - C)$ (lei de De Morgan);
10. $A - (B \cap C) = (A - B) \cup (A - C)$ (lei de De Morgan);
11. $A - B = A \cap B'$;
12. $A \subseteq B \Leftrightarrow A - B = \varnothing$;
13. $A - (A - B) = A \cap B$;
14. $A \cap (B - C) = (A \cap B) - (A \cap C)$;
15. $A \cap B \subseteq (A \cap C) \cup (B \cap C')$;
16. $(A \cup C) \cap (B \cup C') \subseteq A \cup B$.

Princípio da dualidade

A cada propriedade verdadeira de subconjuntos de E escrita usando inclusão, igualdade, união, intersecção e complementação, corresponde uma outra propriedade verdadeira (dual da primeira) que se obtém trocando a ordem das inclusões e substituindo uniões por intersecções, intersecções por uniões e cada conjunto pelo seu complementar. Este facto designa-se *"princípio da dualidade para conjuntos"*.

Definição 1.10 - *Se A e B são conjuntos, a soma de A e B é o conjunto $A + B$ definido por*

$$A + B = (A - B) \cup (B - A).$$

Propriedades da soma

1. $A + B = B + A$ (comutativa);
2. $A + (B + C) = (A + B) + C$ (associativa);
3. $A + \varnothing = A$;
4. $A + A = \varnothing$.

As demonstrações destas propriedades deixam-se ao cuidado do leitor. No entanto, estas encontram-se feitas em *Isabelle* na subsecção 2.3.2.

1.7 Conjunto das partes

Axioma 1.11 ***(da potência[7])*** - *Para cada conjunto existe uma colecção de conjuntos que contém entre os seus elementos todos os subconjuntos desse conjunto.*

Por outras palavras, se E é um conjunto, então existe um conjunto (colecção) P tal que se $X \subseteq E$, então $X \in P$.

O conjunto P descrito acima, pode ser maior do que se deseja, pois pode conter outros elementos além dos subconjuntos de E. Para ultrapassar este problema, basta aplicar o axioma da expecificação para formar o conjunto seguinte:

$$\{X \in P : X \subseteq E\}.$$

Uma vez que para todo o X, uma condição necessária e suficiente para que X pertença a este conjunto é que X seja um subconjunto de E, então tem-se:

$$\mathcal{P} = \{X : X \subseteq E\}.$$

O conjunto $\mathcal{P}$ é chamado *conjuntos das partes* de E. O axioma da extensionalidade garante a sua unicidade. A dependência de $\mathcal{P}$ de E é representada por $\mathcal{P}(E)$, em vez de apenas $\mathcal{P}$.

Exemplos:

1. $\mathcal{P}(\varnothing) = \{\varnothing\}$;
2. $\mathcal{P}(\{a\}) = \{\varnothing, \{a\}\}$;
3. $\mathcal{P}(\{a, b\}) = \{\varnothing, \{a\}, \{b\}, \{a, b\}\}$;
4. $\mathcal{P}(\{a, b, c\}) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$.

[7] A designação deste axioma é tradução do termo em inglês "powers" utilizado por Halmos. Em português é também denominado axioma das partes.

Em seguida, apresenta-se uma série de propriedades cujas demonstrações se deixa ao cuidado do leitor.

Algumas dessas demonstrações encontram-se feitas em *Isabelle* na subsecção 2.3.4. As restantes estão na subsecção A.2.5 do apêndice.

Propriedades

1. Se $E \subseteq F$, então $\mathcal{P}(E) \subseteq \mathcal{P}(F)$;
2. $\mathcal{P}(E) \cap \mathcal{P}(F) = \mathcal{P}(E \cap F)$;
3. $\mathcal{P}(E) \cup \mathcal{P}(F) \subseteq \mathcal{P}(E \cup F)$;
4. $\cap \mathcal{P}(E) = \varnothing$;
5. $E = \cup \mathcal{P}(E)$.

1.8 Pares ordenados

Um conceito matemático que pode ser definido utilizando a teoria dos conjuntos é o conceito de par ordenado.

Qual o significado de organizar os elementos de um conjunto A numa certa ordem?

Suponha-se que se tem um conjunto $A = \{a, b, c, d\}$ em que todos os seus elementos são distintos. Ordene-se os elementos de A numa determinada ordem, por exemplo

$$c \quad b \quad d \quad a.$$

Pode-se, para cada posição em particular da lista ordenada, considerar o conjunto de todos os elementos que ocorrem nesse posição ou antes. Assim, obtém-se os conjuntos

$$\{c\} \quad \{c, b\} \quad \{c, b, d\} \quad \{c, b, d, a\}.$$

Considere-se agora a colecção $\mathcal{C}$ que contém todos estes conjuntos.

$$\mathcal{C} = \{\{c\}, \{c, b\}, \{c, b, d\}, \{c, b, d, a\}\}.$$

Será que a partir da colecção $\mathcal{C}$ se pode descobrir a ordem pela qual estavam os elementos do conjunto A?

A resposta é sim. Para tal, em primeiro lugar procura-se um conjunto da colecção $\mathcal{C}$ que esteja contido em todos os outros conjuntos de $\mathcal{C}$, neste caso $\{c\}$. Em seguida, remove-se este de $\mathcal{C}$. Como $\{c\}$ é o primeiro que verifica a condição anterior, então isso quer dizer que c é o primeiro elemento da lista. Agora, entre os restantes conjuntos de $\mathcal{C}$ escolhe-se aquele que está contido em todos os outros e retira-se este de $\mathcal{C}$. Neste caso $\{c, b\}$. Assim, b é o elemento seguinte da lista. Procedendo

do mesmo modo até que $\mathcal{C}$ fique sem elementos, verifica-se que os elementos de A estavam inicialmente ordenados do seguinte modo:

$$c\ b\ \ d\ \ \ a,$$

como era de esperar.

Utilizando este processo, constata-se que é possível passar do conjunto $\mathcal{C}$ para a lista ordenada de elementos de um dado conjunto A. Assim, para cada ordem pode-se associar uma colecção $\mathcal{C}$ de subconjuntos de A, de tal modo que a ordem dada pode ser recuperada de forma única, a partir de $\mathcal{C}$.

A passagem de uma ordem dos elementos do conjunto A para a colecção $\mathcal{C}$ pode ser ilustrada com maior facilidade se o conjunto A tiver apenas dois elementos, por exemplo:

$$A = \{a, b\}.$$

Se neste caso se desejar que a seja o primeiro elemento, então

$$\mathcal{C} = \{\{a\}, \{a, b\}\}.$$

Se por outro lado se quiser b em primeiro lugar, então

$$\mathcal{C} = \{\{b\}, \{a, b\}\}.$$

Este processo de ordenação de elementos de um conjunto A leva à definição de par ordenado. Deste modo, o par ordenado de a e b, com primeira coordenada a e segunda coordenada b é o conjunto (a, b) definido por

$$(a, b) = \{\{a\}, \{a, b\}\}.$$

Proposição 1.12 - *Se (a, b) e (x, y) são pares ordenados e se $(a, b) = (x, y)$, então $a = x$ e $b = y$.*

Prova. Em primeiro lugar, note-se que se $a = b$, então $\{a, b\} = \{a, a\}$ e pelo axioma da extensionalidade, $\{a, a\} = \{a\}$. Portanto, o par ordenado $(a, b) = \{\{a\}, \{a, b\}\} = \{\{a\}, \{a\}\}$. Novamente, pelo axioma da extensionalidade, $\{\{a\}, \{a\}\} = \{\{a\}\}$, ou seja, (a, b) é um conjunto singular. Por outro lado, se $(a, b) = \{\{a\}, \{a, b\}\}$ é um conjunto singular, então $\{a\} = \{a, b\}$, deste modo $b \in \{a\}$, logo $a = b$.

Suponha-se agora que $(a, b) = (x, y)$.

- Se $a = b$, então (a, b) e (x, y) são conjuntos singulares. Uma vez que, $\{x\} \in (a, b)$ e $\{a\} \in (x, y)$, então a, b, x, y são todos iguais, e assim, $x = y$.

- Se $a \neq b$, então tanto (a, b) como (x, y) contêm exactamente um conjunto singular, nomeadamente $\{a\}$ e $\{x\}$ respectivamente, de tal modo que $a = x$. Uma vez que neste caso tanto (a, b) como (x, y), contêm unicamente um par não ordenado, que não é um conjunto singular $\{a, b\}$ e $\{x, y\}$, respectivamente, então $\{a, b\} = \{x, y\}$. Em particular, $b \in \{x, y\}$. Como $b \neq x$, pois caso contrário, $a = x$ e $b = x$, e logo $a = b$, o que contradiz a hipótese, assim, tem-se que ter $b = y$.

■

Se A e B são conjuntos, será que existe um conjunto que contém todos os pares ordenados (a,b), com $a \in A$ e $b \in B$?

A resposta é sim. De facto, se $a \in A$ e $b \in B$, então $\{a\} \subseteq A$ e $\{b\} \subseteq B$ e então $\{a,b\} \subseteq A \cup B$. Uma vez que, também $\{a\} \subseteq A \cup B$ tem-se que, tanto $\{a\}$ e $\{a,b\}$ são elementos de $\mathcal{P}(A \cup B)$. Isto faz com que $\{\{a\},\{a,b\}\}$ seja um subconjunto de $\mathcal{P}(A \cup B)$, o que quer dizer que $\{\{a\},\{a,b\}\}$ é um elemento de $\mathcal{P}(\mathcal{P}(A \cup B))$; por outras palavras $(a,b) \in \mathcal{P}(\mathcal{P}(A \cup B))$ sempre que $a \in A$ e $b \in B$.

Assim, sabendo que existe um conjunto que contém todos os pares ordenados (a,b) com $a \in A$ e $b \in B$, basta aplicar o axioma da especificação e o axioma da extensionalidade para construir o conjunto $A \times B$, que é o conjunto de todos os pares ordenados (a,b) com $a \in A$ e $b \in B$. Este conjunto é denominado *produto cartesiano* de A e B.

$$A \times B = \{x \in \mathcal{P}(\mathcal{P}(A \cup B)) : x = (a,b) \text{ para algum } a \in A \text{ e para algum } b \in B\}.$$

A unicidade deste é garantida pelo axioma da especificação.

O produto cartesiano de dois conjuntos é um conjunto de pares ordenados e o mesmo acontece a todo o subconjunto de um produto cartesiano.

Proposição 1.13 - *Se R é um conjunto tal que todo o elemento de R é um par ordenado, então existem dois conjuntos A e B tais que $R \subseteq A \times B$.*

Prova. Quer-se provar que todo o conjunto de pares ordenados é um subconjunto do produto cartesiano de dois conjuntos. Suponha-se que $x \in R$ tal que $x = \{\{a\}\{a,b\}\}$, para algum a e para algum b. Uma vez que os elementos de R são conjuntos, pode-se considerar a união de conjuntos em R. Como x é um conjunto pertencente a R, então os elementos de x pertencem à união de conjuntos em R. Como $\{a,b\}$ é um dos elementos de x então $\{a,b\} \in \cup R$. Formando a união de conjuntos em $\cup R$ e como $\{a,b\}$ é um desses conjuntos, logo os elementos de $\{a,b\}$ pertencem a essa união e assim, tanto a como b pertencem a $\cup \cup R$, logo $(a,b) \in \cup \cup R \times \cup \cup R$.

Para se ter $R \subseteq A \times B$, pode-se considerar A e B ambos iguais a $\cup \cup R$. ■

Os "menores" conjuntos A e B, que satisfazem a propriedade anterior, são definidos usando o axioma da especificação da seguinte forma:

$$A = \{a \in \cup \cup R : \text{ para algum } b \ ((a,b) \in R)\}$$

e

$$B = \{b \in \cup \cup R : \text{ para algum } a \ ((a,b) \in R)\}.$$

Estes conjuntos são denominados projecções de R na primeira e segunda coordenada respectivamente.

Desta secção, aquilo que é importante reter é que os pares ordenados são determinados de forma única pela sua primeira e segunda coordenadas, que o produto cartesiano pode ser formado e que todo o conjunto de pares ordenados é subconjunto de algum produto cartesiano.

Em seguida, apresenta-se uma série de propriedades cujas demonstrações se deixa ao cuidado do leitor.

Algumas dessas demonstrações encontram-se feitas em *Isabelle* na subsecção 2.3.5. As restantes estão na subsecção A.2.6 do apêndice.

Propriedades

Se A, B, X e Y são conjuntos, então

1. $(A \cup B) \times X = (A \times X) \cup (B \times X)$;
2. $(A \cap B) \times (X \cap Y) = (A \times X) \cap (B \times Y)$;
3. $(A - B) \times X = (A \times X) - (B \times X)$.
4. $A = \varnothing$ ou $B = \varnothing$ se e só se $A \times B = \varnothing$.
5. Se $A \subseteq X$ e $B \subseteq Y$, então $A \times B \subseteq X \times Y$.
6. Se $A \times B \neq \varnothing$ e $A \times B \subseteq X \times Y$, então $A \subseteq X$ e $B \subseteq Y$.

1.9 Relações

No nosso dia-a-dia está-se constantemente a relacionar objectos, pessoas, etc. Por exemplo, quando se diz "a Joana é filha da Rita", está-se a relacionar (associar) a Joana com a Rita.

Para se relacionar dois objectos usa-se normalmente uma condição ou propriedade que esses objectos satisfazem, ou seja, se $P(x,y)$ é essa condição, ela pode definir uma relação formada pelos pares ordenados (x,y), que satisfazem $P(x,y)$.

Exemplo 1.14 - *Se $P(x,y)$ é a condição "x é a capital de y", então (Lisboa, Portugal) é um par ordenado que satisfaz esta condição. A relação definida por $P(x,y)$ é então o conjunto dos pares ordenados que satisfazem esta condição.*

Com este exemplo, facilmente se verifica que para definir uma relação, recorrendo à teoria dos conjuntos, se utilizam os pares ordenados. Outro aspecto importante é a ordem, que neste caso é relevante. Por exemplo, considerando a relação acima referida, verifica-se que o par (Portugal, Lisboa), não a satisfaz.

Assim sendo, uma relação é um conjunto de pares ordenados, ou seja, R é uma relação se cada um dos seus elementos for um par ordenado. Logo, se $z \in R$, então existem x e y tais que $z = (x,y)$.

Notação 1.15 - *Seja R uma relação, então pode-se escrever xRy em vez de $(x,y) \in R$. Quando $(x,y) \notin R$ escreve-se $x \not R y$.*

xRy lê-se "x está em relação com y por R" ou "x está na relação R com y".

O produto cartesiano de dois conjuntos X e Y quaisquer é exemplo de uma relacão.

Na secção anterior, viu-se que associado a todo o conjunto R de pares ordenados, existem dois conjuntos denominados projecções de R na primeira e segunda coordenada. Neste caso, estes conjuntos são conhecidos como domínio e contradomínio de R, que se designa por $dom\,(R)$ e $cod\,(R)$, respectivamente.

Relembre-se que:

$$dom\,(R) = \{x \in X : \text{ para algum } y \ (xRy)\},$$

ou seja, $dom\,(R)$ é o conjunto de todos os x para os quais existe um y tal que, xRy.

$$cod\,(R) = \{y \in X : \text{ para algum } x \ (xRy)\},$$

ou seja, $cod\,(R)$ é o conjunto de todos os y para os quais existe um x tal que, xRy.

Se $R = X \times Y$, então $dom\,(R) = X$ e $cod\,(R) = Y$.

Se R for a relação de igualdade em X, então $dom\,(R) = cod\,(R) = X$.

Definição 1.16 - *Seja R uma relação contida no produto cartesiano $X \times X$.*

1. *R é **reflexiva** se e só se, para todo o $x \in X$, xRx.*
2. *R é **simétrica** se e só se, para todo o $x, y \in X$, se xRy, então yRx.*
3. *R é **transitiva** se e só se, para todo o $x, y, z \in X$, se xRy e yRz, então xRz.*
4. *R é uma **relação de equivalência** se e só se R é reflexiva, simétrica e transitiva.*

Definição 1.17 - *Uma partição de X é uma colecção C, disjunta[8], de subconjuntos não vazios de X, cuja união é o conjunto X.*

Exemplo 1.18 - *Seja $X = \{a, b, c, d, e\}$. Uma possível partição de X é $C = \{\{a, b\}, \{d\}, \{c, e\}\}$*

Definição 1.19 - *Dada uma relação de equivalência R num conjunto X, diz-se que dois elementos x e y de X são equivalentes se e só se xRy.*

Definição 1.20 - *Se R é uma relação de equivalência em X e se $x \in X$, então a classe de equivalência de x segundo R ou módulo R, que se designa por x/R ou $[x]_R$, é o conjunto de todos os elementos y de X que são equivalentes a x, ou seja, todos os elementos y de X tais que xRy. Ao elemento x chama-se representante da classe.*

Exemplo 1.21 - *Se R é a igualdade em X, então cada classe de equivalência é um conjunto singular, ou seja, se $x \in X$ então $x/R = [x]_R = \{x\}$.*

[8] $(\forall_{X_1, X_2 \in C} \ (X_1 \neq X_2 \rightarrow X_1 \cap X_2 = \varnothing))$

Exemplo 1.22 - *Se* $R = \{(a,a),(a,b),(b,a),(b,b),(c,c)\}$ *com* $X = \{a,b,c\}$ *onde* a, b *e* c *são elementos distintos tem-se:*

$$a/R = [a]_R = \{a,b\}\,; \quad b/R = [b]_R = \{a,b\}\,; \quad c/R = [c]_R = \{c\}\,.$$

Teorema 1.23 - *Seja* R *uma relação de equivalência num conjunto* X *e sejam* a *e* b *elementos quaisquer de* X. *Tem-se*

i) $a \in [a]_R$;
ii) aRb *se e só se* $[a]_R = [b]_R$;
iii) $a\not{R}b$ *se e só se* $[a]_R \cap [b]_R = \varnothing$.

O conjunto de todas as classes de equivalência de todos os elementos de X chama-se *conjunto quociente de* X *segundo* R *ou módulo* R, que se designa por X/R.

$$X/R = \{[x]_R : x \in X\}$$

Exemplo 1.24 - *Seja* $R = \{(a,a),(a,b),(b,a),(b,b),(c,c)\}$ *a relação em* $X = \{a,b,c\}$, *com* a, b, c *distintos. Tem-se*

$$X/R = \{[a]_R, [c]_R\} = \{\{a,b\},\{c\}\}\,.$$

Observação 1.25 - *Se* $X = \varnothing$, *então* $X/R = \varnothing$.

Teorema 1.26 - *Seja* R *uma relação de equivalência num conjunto* X, *não vazio. Então* X/R *constitui uma partição de* X, *a partição de* X *induzida por* R.

Prova. Relembrando, uma partição de X é uma colecção $\mathcal{C}$, disjunta, de subconjuntos não vazios de X, cuja união é o conjunto X. Como X é um conjunto não vazio, então nenhuma classe de equivalência de elementos de X é vazia, pois se $x \in X$ então $x \in [x]_R$.

Todo o elemento $x \in X$ pertence a alguma classe de equivalência, nomeadamente à classe $[x]_R$. Por definição de classe de equivalência todas estas são formadas apenas por elementos de X. Logo a união de todas as classes de equivalência originam o conjunto X. Finalmente, por *iii)* do teorema anterior, classes de equivalência distintas são disjuntas. Assim, X/R constitui uma partição de X. ■

No exemplo anterior, X/R constitui uma partição de X, a partição induzida por R.

Considere-se agora a relação $X/\mathcal{C}$, definida em X por

$$x\,(X/\mathcal{C})\,y$$

apenas no caso de x e y pertencerem ao mesmo conjunto da colecção $\mathcal{C}$, onde $\mathcal{C}$ é uma partição de X. Denomine-se $X/\mathcal{C}$ relação induzida pela partição $\mathcal{C}$.

A ligação entre as relações de equivalência e as partições pode ser descrita dizendo que a passagem de $\mathcal{C}$ para $X/\mathcal{C}$ é a passagem inversa de R para X/R.

Lema 1.27 *Se* R *é uma relação de equivalência em* X, *então o conjunto das classes de equivalência é uma partição de* X, *que induz a relação* R. *Se* $\mathcal{C}$ *é uma partição de* X, *então a relação induzida é uma relação de equivalência, cujo conjunto das classes de equivalência é* $\mathcal{C}$.

Prova. Seja R uma relação de equivalência. Uma vez que x pertence a alguma classe de equivalência, (por exemplo $x \in x/R$), então a união de todas as classes de equivalência é todo o X. Se $z \in x/R \cap y/R$, logo xRz e zRy, portanto xRy. Isto significa que se duas classes de equivalência têm um elemento em comum, então são classes idênticas. Por outras plalavras, duas classes de equivalência distintas são sempre disjuntas. Assim, o conjunto das classes de equivalência é um partição. Se dois elementos pertencem ao mesmo conjunto (classe de equivalência) desta partição, então por definição estão na relação R um com o outro. A prova da primeira parte da afirmação está concluída.

Para se provar a segunda parte de afirmação, considere-se uma partição $\mathcal{C}$ e a relação induzida $X/\mathcal{C}$. Uma vez que todo o elemento de X pertence a algum conjunto de $\mathcal{C}$, então tem-se que x pertence ao mesmo conjunto de $\mathcal{C}$ que ele próprio, logo $x\,(X/\mathcal{C})\,x$, ou seja, a reflexiva é verificada. Se $x\,(X/\mathcal{C})\,y$, então x e y pertencem ao mesmo conjunto de $\mathcal{C}$. Logo y e x também pertencem ao mesmo conjunto de $\mathcal{C}$ e assim $y\,(X/\mathcal{C})\,x$, ou seja, a simétrica é também verificada. Se $x\,(X/\mathcal{C})\,y$ e $y\,(X/\mathcal{C})\,z$, então x e y pertencem ao mesmo conjunto de $\mathcal{C}$, e y e z pertencem ao mesmo conjunto de $\mathcal{C}$. Logo x e z também pertencem ao mesmo conjunto de $\mathcal{C}$ e assim $x\,(X/\mathcal{C})\,z$, ou seja, a transitiva é verificada. A classe de equivalência de cada $x \in X$ é o conjunto de $\mathcal{C}$ ao qual x pertence. E a prova está concluída. ■

1.10 Funções

Se X e Y são conjuntos, uma função de X para Y é uma relação f tal que, $dom\,(f) = X$ e, para cada $x \in X$, existe um único elemento $y \in Y$ de modo que $(x,y) \in f$, ou seja, $dom\,(f) = X$ e se $(x,y) \in f$ e $(x,z) \in f$ então $y = z$.

Assim, se f é uma função, representar-se-á y por $f(x)$ e escrever-se-á $f(x) = y$ em vez de $(x,y) \in f$, ou xfy.

y é a imagem de x por f, ou valor de f em x. De modo equivalente pode-se dizer que f transforma x em y.

Ao conjunto f de pares ordenados chama-se de gráfico da função f.

Para indicar que f é uma função (aplicação, correspondência) de X para Y, escreve-se

$$f : X \to Y.$$

Os conjuntos X e Y são designados por *conjunto de partida* (ou *domínio*) e *conjunto de chegada*, respectivamente.

Definição 1.28 - *Ao conjunto de elementos de Y para os quais existe um $x \in X$ tal que, $f(x) = y$, chama-se contradomínio de f, que se designa por $cod\,(f)$.*

Assim, se $f : X \to Y$, então $dom(f) = X$ e $cod(f) \subseteq Y$.

Definição 1.29 - *Seja $f : X \to Y$ e $A \subseteq X$, designa-se por imagem de A por f, o conjunto, que se denota por $f\,(A)$, formado por todos os $y \in Y$ para os quais existe um $x \in A$ tal que, $f\,(x) = y$. Assim,*

$$f\,(A) = \{f\,(x) \in Y : x \in A\}.$$

Definição 1.30 - *Se X é um subconjunto de Y, a função f definida por*[9] $f(x) = x$, *para cada $x \in X$ é chamada inclusão de X em Y.*

Observação 1.31 - *A inclusão de X em X é denominada identidade em X.*

Definição 1.32 - *Se f é uma função de Y para Z, e se $X \subseteq Y$, chama-se restrição de f ao conjunto X, à função, que se designa por $f|X$, assim definida*

$$f|X : X \to Z$$
$$(f|X)(x) = f(x), \text{ para cada } x \in X.$$

Observação 1.33 - $cod(f|X) = f(X)$.

Considere-se agora dois conjuntos X e Y quaisquer. A função f de $X \times Y$ em X, definida por $f((x,y)) = x$ é a projecção de $X \times Y$ em X. Analogamente, a função g de $X \times Y$ em Y, definida por $f((x,y)) = y$ é a projecção de $X \times Y$ em Y.

Observação 1.34 - *A terminologia utilizada nesta secção é ligeiramente diferente de uma usada anteriormente. Assim, se $R = X \times Y$ aquilo que anteriormente era designado por projecção de R na primeira coordenada é agora designado por contradomínio da projecção f.*

Definição 1.35 - *Considere-se uma aplicação $f : X \to Y$ qualquer.*
(a) f é injectiva se e só se quaisquer dois elementos distintos de X têm imagens distintas por f.
(b) f é sobrejectiva se e só se qualquer elemento do conjunto de chegada for imagem por f de algum elemento do conjunto X. Neste caso $cod(f) = Y$.
(c) f é bijectiva se e só se f é injectiva e sobrejectiva.

Definição 1.36 - *Se A é um subconjunto de X, a função característica*[10] *de A é a função χ de X para $\{0,1\}$ tal que* $\begin{cases} \chi(x) = 1 \text{ se } x \in A \\ \chi(x) = 0 \text{ se } x \notin A \end{cases}$.

Um aspecto importante da teoria de conjuntos é que os números são definidos como conjuntos. Assim, como se verá mais adiante $0 = \varnothing$ e $1 = \{\varnothing\}$.

1.11 Famílias

Por vezes, a linguagem e as notações que se introduz para as funções sofre certas modificações.

Suponha-se que x é uma função de um conjunto I para um conjunto X, ou seja, $x : I \to X$. Cada elemento do conjunto I é denominado *índice*, assim, I é o *conjunto*

[9] A frase "a função definida por ..." implica que existe apenas uma função que satisfaz aquela condição.

[10] A função característica permite identificar quais os elementos de X que pertencem a A e quais os que não pertencem.

dos índices. À função x chama-se *família* e o valor da função x no índice i, que se denota por x_i (para cada $i \in I$, $x(i) = x_i$), é denominado *termo* ou *elemento da família.*

Para a família usa-se a notação $(x_i)_{i\in I}$ ou $\{x_i\}_{i\in I}$.

O contradomínio desta função, dado por $cod(x) = \{x_i : i \in I\}$, é por designado por *conjunto indexado*, ou *conjunto de elementos da família.*

Alguns conceitos anteriormente referidos podem ser generalizados a famílias.

Considere-se a família $\{A_i\}$ de subconjuntos de X, que representa a função A de um conjunto de índices I para $\mathcal{P}(X)$. A união do contradomínio da família é chamada união da família $\{A_i\}$, ou união dos conjuntos A_i, que se representa por

$$\cup_{i\in I} A_i \text{ ou } \cup_i A_i,$$

se não houver risco de confusão. Esta união generaliza a união de conjuntos.

Da definição de união decorre que $x \in \cup_i A_i$ se e só se x pertence a A_i, para pelo menos um i.

Considere-se $I = 2$, de tal modo que o contradomínio da família $\{A_i\}$ é o par não ordenado $\{A_0, A_1\}$. Então $\cup_i A_i = A_0 \cup A_1$.

Note-se que não existe perda de generalidade quando se considera uma família de conjuntos em vez de uma colecção arbitrária de conjuntos, pois toda a colecção de conjuntos é o contradomínio de alguma família.

Suponha-se que $\{I_j\}$ é uma família de conjuntos com domínio J. Considere-se agora $K = \cup_j I_j$ e seja $\{A_k\}$ uma família de conjuntos com domínio K. Tem-se então que

$$\cup_{k\in K} A_k = \cup_{j\in J} \left(\cup_{i\in I_j} A_i\right),$$

que é a versão generalizada da lei associativa da união.

Observe-se que a união de conjuntos vazios faz sentido e é vazia, pois se se considerar um x qualquer, $x \in \cup\varnothing$ se e só se $x \in X$ para algum $X \in \varnothing$. Como não existe nenhum X em $\varnothing$, então $x \notin \cup\varnothing$, logo $\cup\varnothing = \varnothing$.

Pelo contrário, a intersecção de conjuntos vazios não faz sentido, pois se se considerar um x qualquer, $x \in \cap\varnothing$ se e só se $x \in X$ para todo o $X \in \varnothing$. Verifica-se que qualquer x satisfaz esta condição, uma vez que se assim não fosse existiria um $X \in \varnothing$ tal que $x \notin X$.

Existem formulações da teoria de conjuntos em que é sempre pressuposto um universo fixo U, ou seja, todos os conjuntos referidos são subconjuntos desse universo. Nesse caso, $\cap\varnothing = U$. No entanto, não é essa a formulação apresentada aqui. Por outro lado, sabe-se que não existe um universo "global" (ver paradoxo de Russel, página 4). Assim sendo, à excepção deste caso, a notação e terminologia para o caso da intersecção é paralela à união em todos os aspectos. Deste modo, se $\{A_i\}$ é uma família não vazia[11] de conjuntos, a intersecção do contradomínio da família é denominado intersecção da família $\{A_i\}$,ou intersecção dos conjuntos A_i, que se denota por

$$\cap_{i\in I} A_i \text{ ou } \cap_i A_i,$$

caso não haja risco de confusão.

[11] Por família não vazia, entendemos, família cujo domínio I não é vazio.

Da definição de intersecção decorre que se $I \neq \varnothing$, então uma condição necessária e suficiente para que x pertença a $\cap_i A_i$ é que x pertença a A_i, para todo o $i \in I$.

Se $\{A_i\}$ é a família de subconjuntos de X e $B \subseteq X$, então

$$B \cap \cup_i A_i = \cup_i (B \cap A_i)$$

e

$$B \cup \cap_i A_i = \cap_i (B \cup A_i),$$

que são as generalizações das propriedades distributivas, referidas na secção dedicada às uniões e intersecções.

Utilizando a notação de família generaliza-se o produto cartesiano como se segue.

O produto cartesiano de dois conjuntos X e Y foi definido como o conjunto de todos os pares ordenados (x, y) com $x \in X$ e $y \in Y$. Considere-se um par não ordenado qualquer $\{a, b\}$ com $a \neq b$. Considere-se o conjunto Z de todas as famílias z, indexadas por $\{a, b\}$, tal que $z_a \in X$ e $z_b \in Y$. Se a função f de Z para $X \times Y$ é definida por $f(z) = (z_a, z_b)$, então f é uma função injectiva. A diferença entre Z e $X \times Y$ é meramente uma questão de notação.

Se $\{X_i\}$ é uma família de conjuntos, com $i \in I$, o produto cartesiano da família é, por definição, o conjunto de todas as famílias $\{x_i\}$, com $x_i \in X_i$, para cada $i \in I$, que se denota por $\prod_{i \in I} X_i$.

Se todo o X_i for igual a X, então $\prod_{i \in I} X_i = X^I$. Se $I = \{a, b\}$, com $a \neq b$, então $\prod_{i \in I} X_i = X_a \times X_b$, e se $I = \{a\}$, então $\prod_{i \in I} X_i = X_a$. Os triplos ordenados, os quádruplos ordenados, etc., podem ser definidos como famílias cujos conjuntos de índices são triplos, quádruplos não ordenados, etc.

1.12 Inversa e composta

Dada uma função f de X para Y, a *inversa*[12] de f, que se denota por f^{-1}, é a função de $\mathcal{P}(Y)$ para $\mathcal{P}(X)$, tal que se $B \subseteq Y$, então

$$f^{-1}(B) = \{x \in X : f(x) \in B\}.$$

Por outras palavras, $f^{-1}(B)$ é o conjunto formado por todos os elementos de X cuja imagem por f está em B. O conjunto $f^{-1}(B)$ é denominado *imagem inversa*, ou *pré-imagem* de B, por f.

Uma condição necessária e suficiente para que f seja uma função de X em Y, é que a imagem inversa por f, de cada subconjunto não vazio de Y, seja um subconjunto não vazio de X.

Uma condição necessária e suficiente para que f seja injectiva, é que a imagem inversa por f, de cada conjunto singular de $cod(f)$, seja um conjunto singular de X. Assim, para funções f injectivas, pode dizer-se que $f^{-1}(y) = x$ se e só se $f(x) = y$.

Seja f uma função de X para Y.

[12] Note-se que não se deve confundir esta inversa de f com a função que normalmente é designada por função inversa quando existe. A função f^{-1} de $\mathcal{P}(y)$ para $\mathcal{P}(x)$ tem a vantagem técnica de ser sempre possível defini-la.

Proposição 1.37 - *Se $B \subseteq Y$, então*

$$f\left(f^{-1}(B)\right) \subseteq B.$$

Prova. *Se $y \in f\left(f^{-1}(B)\right)$, então $y = f(x)$ para algum $x \in f^{-1}(B)$. Mas por definição $f^{-1}(B) = \{x \in X : f(x) \in B\}$, logo $f(x) \in B$, ou seja, $y \in B$.* ■

Proposição 1.38 - *Se f é uma função de X para Y, onde $cod(f) = Y$, então*

$$f\left(f^{-1}(B)\right) = B.$$

Prova. *Se $y \in B$, então $y = f(x)$ para algum $x \in X$,ou seja, para algum $x \in f^{-1}(B)$. Isto significa que $y \in f\left(f^{-1}(B)\right)$.* ■

Proposição 1.39 - *Se $A \subseteq X$, então*

$$A \subseteq f^{-1}(f(A)).$$

Prova. *Se $x \in A$, então $f(x) \in f(A)$, o que significa que $x \in f^{-1}(f(A))$.* ■

Proposição 1.40 - *Se f é injectiva, então*

$$A = f^{-1}(f(A)).$$

Prova. *Se $x \in f^{-1}(f(A))$, então $f(x) \in f(A)$ e portanto $f(x) = f(u)$ para algum $u \in A$, o que implica que $x = u$, pois f é injectiva. Logo $x \in A$.* ■

Proposição 1.41 - *Se $\{B_i\}$ é uma família de subconjuntos de Y, então*

$$f^{-1}(\cup_i B_i) = \cup_i f^{-1}(B_i)$$

e

$$f^{-1}(\cap_i B_i) = \cap_i f^{-1}(B_i).$$

No fundo, ao se estudar as funções inversas, verifica-se que aquilo que uma função faz pode ser desfeito.

O próximo passo, será verificar que o que duas funções fazem, pode muitas vezes ser feito num só passo, ou seja, por uma só função. Esta é conhecida por função composta.

Se f é uma função de X para Y e g uma função de Y para Z, então todo o elemento do contradomínio de f pertence ao domínio de g, e consequentemente $g(f(x))$ tem significado para cada $x \in X$.

Definição 1.42 - *Sendo $f : X \to Y$ e $g : Y \to Z$, chama-se função composta de f e g à função de X para Z, que se denota por $g \circ f$, definida por $g \circ f(x) = g(f(x))$ para todo o $x \in X$.*

Observe-se que a ordem é importante, pois para que $g \circ f$ seja definida é necessário que o contradomínio de f esteja contido no domínio de g, o que pode acontecer sem que o contrário se verifique, simultaneamente. Mesmo que tanto $g \circ f$ e $f \circ g$ estejam definidas, o que acontece se se considerar, por exemplo as funções f de X para Y e g de Y para X, as funções $g \circ f$ e $f \circ g$ não são necessariamente iguais. Por outras palavras, a composição de funções não é necessariamente comutativa. No entanto, a composição de funções é sempre associativa. Se f é uma função de X para Y, g uma função de Y para Z e h uma função de Z para V, facilmente se prova que

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

A relação entre a inversa e a composição é muito importante. Se $f : X \rightarrow Y$ e $g : Y \rightarrow Z$, então $f^{-1} : P(Y) \rightarrow P(X)$ e $g^{-1} : P(Z) \rightarrow P(Y)$. Neste caso, tem-se que $f^{-1} \circ g^{-1}$ é a inversa de $g \circ f$.

A inversa e a composição de funções são casos especiais de operações similares para as relações. Assim, em particular, associada a toda a relação R de X para Y existe uma relação inversa R^{-1} de Y para X. Por definição, $yR^{-1}x$ significa que xRy.

Seja R uma relação de X para Y e S uma relação de Y para Z. A relação composta T de X para Z, denotada por $S \circ R$, é definida de tal modo que, xTz se e só se existe um $y \in Y$ tal que xRy e ySz. Se tanto R e S são funções, então xRy e ySz podem ser reescritos como $R(x) = y$ e $S(y) = z$, respectivamente. Assim, $S(R(x)) = z$ se e só se xTz, de tal modo que a função composta é de facto um caso especial da relação composta.

A composição de funções está relacionada com a inversa de acordo com a seguinte igualdade

$$(SR)^{-1} = R^{-1}S^{-1}.$$

No caso das relações serem funções $(f \circ g) = g^{-1} \circ f^{-1}$ é um caso particular da igualdade acima.

1.13 Números naturais

O próximo objectivo é definir os números naturais utilizando a teoria de conjuntos.

Definição 1.43 - *Para todo o conjunto x, define-se o sucessor de x, denotado por $suc(x)$, como sendo o conjunto obtido juntando a x todos os elementos de x. Por outras palavras, $suc(x) = x \cup \{x\}$.*

Pode-se definir $0, 1$ e 2 da seguinte forma: $0 = \varnothing$, $1 = \{\varnothing\}$, $2 = \{\varnothing, \{\varnothing\}\}$. Em geral, cada número natural é definido como sendo igual ao conjunto de todos os seus

antecessores. Assim tem-se,

$$\begin{aligned}
0 &= \varnothing \\
1 &= suc(0) = \{\varnothing\} = \{0\}; \\
2 &= suc(1) = \{\varnothing, \{\varnothing\}\} = \{0,1\}; \\
3 &= suc(2) = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\} = \{0,1,2\}; \\
4 &= suc(3) = \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}, \{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}\} = \{0,1,2,3\} \\
&\vdots
\end{aligned}$$

A definição dos números naturais sugere a existência de um conjunto infinito.

Axioma 1.44 ***(do infinito)*** *- existe um conjunto que contém o zero e todos os sucessores de cada um dos seus elementos.*

Pode-se definir que um conjunto A é um conjunto sucessor se $0 \in A$ e $suc(x) \in A$, sempre que $x \in A$. Assim, o axioma anterior diz que existe um tal conjunto A.

Proposição 1.45 *A intersecção de toda a família, não vazia, de conjuntos sucessores é um conjunto sucessor.*

Prova. Seja A_i uma família não vazia de conjuntos sucessores, com $i \in I$. Tem-se que $\varnothing \in A_i$ para todo o $i \in I$ e para todo o $i \in I$, $suc(x) \in A_i$, sempre que $x \in A_i$. Logo $\varnothing \in \cap A_i$ e $suc(x) \in \cap A_i$ sempre que $x \in \cap A_i$. Assim, $\cap A_i$ é um conjunto sucessor. ■

Da proposição anterior decorre que a intersecção de todos os conjuntos sucessores contidos em A é um conjunto sucessor w.

Proposição 1.46 *- Fixado um conjunto A, o conjunto w é subconjunto de todo o conjunto sucessor contido em A.*

Prova. Sejam A e B conjuntos sucessores quaisquer, então $A \cap B$ é também um conjunto sucessor. Uma vez que, $A \cap B \subseteq A$ e w é a intersecção de todos os conjuntos sucessores contidos em A, então $w \subseteq A \cap B$. Consequentemente, $w \subseteq B$. ■

Note-se que w não depende da escolha inicial do conjunto sucessor A. O axioma da extensionalidade garante a unicidade do conjunto w.

Cada um dos elementos do conjunto sucessor w é designado por conjunto natural ou natural, ou ainda, número natural, sendo este conjunto denominado conjunto dos números naturais. O conjunto sucessor w é também, representado por $\mathbb{N}_0$.

Definição 1.47 *- Uma família $\{x_i\}$ cujo conjunto de índices é um número natural é denominada sequência finita.*

Definição 1.48 *- Uma família $\{x_i\}$ cujo conjunto de índices é o conjunto de todos os números naturais é designada sequência infinita.*

Se $\{A_i\}$ é uma sequência de conjuntos, cujo conjunto de índices é o número natural $suc(n)$, então a união das sequências é denotada por

$$\bigcup_{i=0}^{n} A_i \text{ ou } A_0 \cup ... \cup A_n.$$

Se o conjunto de índices é o conjunto w, então a união das sequências é representada por

$$\bigcup_{i=0}^{\infty} A_i \text{ ou } A_0 \cup A_1 \cup ....$$

No caso da intersecção, tem-se:

$$\bigcap_{i=0}^{n} A_i \text{ ou } A_0 \cap ... \cap A_n$$

e

$$\bigcap_{i=0}^{\infty} A_i \text{ ou } A_0 \cap A_1 \cap ...,$$

respectivamente.

Para o produto cartesiano tem-se

$$\prod_{i=0}^{n} A_i \text{ ou } A_0 \times ... \times A_n$$

e

$$\prod_{i=0}^{\infty} A_i \text{ ou } A_0 \times A_1 \times ...,$$

respectivamente.

1.14 Axiomas de Peano

O que se sabe de mais importante sobre o conjunto de todos os números naturais (w), é que este é o único conjunto sucessor que é subconjunto de todo o conjunto sucessor.

Dizer que w é um conjunto sucessor significa que:

Proposição 1.49 I) $0 \in w$;

Proposição 1.50 II) *se* $n \in w$, *então* $suc(n) \in w$, *onde* $suc(n) = n \cup \{n\}$.

O facto de w ser o **menor** conjunto sucessor significa que:

Proposição 1.51 III) *se* $0 \in S$ *e se* $suc(n) \in S$ *sempre que* $n \in S$, *então* $w \subseteq S$. *Se além disso* $S \subseteq w$, *então* $S = w$.

Esta propriedade é conhecida como **princípio de indução matemática** e é um método de prova, usado para estabelecer que uma determinada afirmação é verdadeira, para todos os números naturais. Assim, seja P uma afirmação e $S = \{n : P(n)\}$. Se $0 \in S$, ou seja, $P(0)$ [13]. E se $n \in S$, então $suc(n) \in S$, ou seja, se $P(n)$ então $P(suc(n))$ [14]. Pode-se concluir que $S = w$, ou seja, que todos os números naturais satisfazem P. Assim, tem-se duas técnicas alternativas equivalentes para estabelecer $\forall_n P(n)$.

Proposição 1.52 IV) *$suc(n) \neq 0$, para todo o $n \in w$.*

Prova. Uma vez que $n \subseteq suc(n)$ e $0 = \varnothing$ tem-se $suc(n) \neq 0$, para todo o $n \in w$. ■

Proposição 1.53 V) *se $n, m \in w$ e se $suc(n) = suc(m)$, então $n = m$.*

Para se poder provar a proposição **V)** é necessário estabelecer as duas proposições seguintes.

Proposição 1.54 *i) Nenhum número natural é subconjunto de nenhum dos seus elementos.*

A prova desta proposição é uma aplicação do princípio de indução matemática

Prova. Seja $S = \{n \in w :$ para todo o $x \in n$, $n \nsubseteq x\}$.

Base de indução: $0 \in S$, pois é verdade que para todo o $x \in \varnothing$, $\varnothing \nsubseteq x$.

Passo de indução: Assuimindo que para todo o $x \in n$ tem-se $n \nsubseteq x$, quer-se provar que para todo o $y \in suc(n)$, $suc(n) \nsubseteq y$, ou seja, para todo o $y \in suc(n)$, $n \cup \{n\} \nsubseteq y$.

Suponha-se com vista a um absurdo, que existe um $y \in n \cup \{n\}$ tal que $n \cup \{n\} \subseteq y$.

Se $y \in n \cup \{n\}$, tem-se dois casos possíveis: $y \in n$, ou $y = n$.

Se $y \in n$, como $n \cup \{n\} \subseteq y$, então $n \subseteq y$, tem-se deste modo que existe um $y \in n$, tal que $n \subseteq y$, o que contradiz a hipótese de indução.

Se $y = n$, como $n \cup \{n\} \subseteq y$, então, $n \cup \{n\} \subseteq n$, o que implica que $n \in n$ e assim, $n \cup \{n\} = n \subseteq n$, o que contradiz a hipótese de indução. ■

Proposição 1.55 *ii) Todo o elemento de um número natural é um subconjunto desse número natural.*

Definição 1.56 - *Um conjunto E é transitivo se $x \in y$ e $y \in E$, então $x \in E$.*

Assim, a proposição 1.55 *ii)* diz que todo o número natural, é transitivo.

Prova. Seja $S = \{n \in w :$ para todo o $x \in n$, $x \subseteq n\}$.

Base de indução: $0 \in S$, pois é verdade que para todo o $x \in \varnothing$, $x \subseteq \varnothing$.

[13] Este passo do príncipio é designado por Base de Indução.

[14] Este passo é conhecido como Passo de Indução, sendo $P(n)$ a Hipótese de Indução e $P(suc(n))$ a Tese de Indução.

Passo de indução: Assumindo que para todo o $x \in n$, $x \subseteq n$, quer-se concluir que para todo o $y \in suc(n)$, $y \subseteq suc(n)$, ou seja, para todo o $y \in n \cup \{n\}$, $y \subseteq n \cup \{n\}$.

Se $y \in n \cup \{n\}$, tem-se dois casos possíveis: $y \in n$, ou $y = n$.

Se $y \in n$, então $y \subseteq n \subseteq n \cup \{n\}$.

Se $y = n$, então $y \subseteq n \cup \{n\}$. ■

Prova. V) Suponha-se que n e m são números naturais e que $suc(n) = suc(m)$. Uma vez que $n \in suc(n)$, então $n \in suc(m)$. Assim, tem-se dois casos possíveis: $n \in m$, ou $n = m$. Por outro lado, como $m \in suc(m)$, então $m \in suc(n)$ e portanto tem-se $m \in n$, ou $n = m$.

Se $n \neq m$, então $n \in m$ e $m \in n$. Como por $ii)$ n é um conjunto transitivo, então $n \in n$. Logo $n \subseteq n$, o que contradiz $i)$. Portanto $n = m$. ■

As proposições **I)**,**II)**,**III)**,**IV)** e **V)** são conhecidas como os **Axiomas de Peano**. A partir destes axiomas, juntando os axiomas da teoria dos conjuntos, é possível definir os números inteiros, os números racionais, os números reais, os números complexos, e derivar as sua aritmética e propriedades analíticas. Contudo, este não é o objectivo deste trabalho.

O princípio de indução matemática é muitas vezes utilizado não apenas para provar afirmações, mas para definir funções.

Suponha-se que f é a função de um conjunto X para o mesmo conjunto X, e que $a \in X$. Parece natural definir uma sequência infinita $\{u(n)\}$ de elementos de X (ou seja, uma função u de w para X), de tal modo que:

$$\begin{aligned} u(0) &= a; \\ u(1) &= f(u(0)); \\ u(2) &= f(f(u(0))) = f(u(1)); \\ u(3) &= f(f(f(u(0)))) = f(u(2)); \end{aligned}$$

e assim por diante.

Para justificar a definição indutiva necessita-se do resultado seguinte que estabelece a existência da função u.

Teorema 1.57 ***(da Recursão)*** *- Se a é um elemento de um conjunto X, e se f é uma função de X para X, então existe uma função u de w para X tal que, $u(0) = a$ e para todo $n \in w$, $u(suc(n)) = f(u(n))$.*

Prova. Recorde-se que uma função de w para X é um certo tipo de subconjunto de $w \times X$, assim, construir-se-á u como um conjunto de pares ordenados. Para tal, considere-se a colecção $\mathcal{C}$ de todos os subconjuntos A de $w \times X$, para os quais $(0, a) \in A$ e $(suc(n), f(x)) \in A$ sempre que $(n, x) \in A$. Ou seja, $\mathcal{C} = \{A : A \subseteq w \times X \text{ e } (0, a) \in A, \text{ e se } (n, x) \in A, \text{ então } (suc(n), f(x)) \in A\}$. Uma vez que $w \times X$ tem estas propriedades, então a colecção $\mathcal{C}$ é não vazia. Pode-se assim, formar a intersecção u de todos os conjuntos da colecção $\mathcal{C}$, ou seja, $u = \cap \mathcal{C}$.

Facilmente, se verifica que $u \in \mathcal{C}$, falta provar que u é uma função. Tem-se assim, que provar que para cada número natural n existe no máximo um elemento $x \in X$ tal que $(n, x) \in u$ (ou seja, se tanto (n, x) e (n, y) pertencem a u, então $x = y$).

A prova será feita por indução.

Seja S o conjunto de todos os números naturais n para os quais se verifica que $(n,x) \in u$, para no máximo um x.

Provar-se-á que $0 \in S$ e que se $n \in S$, então $suc(n) \in S$.

Base de indução: $0 \in S$?

Se $0 \notin S$, então $(0,b) \in u$ para algum $b \neq a$. Considere-se, neste caso, o conjunto $u - \{(0,b)\}$. Observe-se que este conjunto ainda contém $(0,a)$, pois $b \neq a$. E que se este conjunto contém (n,x), então também contém $(suc(n), f(x))$, isto porque $suc(n) \neq 0$. Logo $(0,b) \neq (suc(n), f(x))$. Por outras palavras, $u - \{(0,b)\} \in \mathcal{C}$, o que contradiz a facto de u ser o menor conjunto de $\mathcal{C}$, (pois u é a intersecção de todos os conjuntos de $\mathcal{C}$), logo $0 \in S$.

Passo de indução: Suponha-se que $n \in S$, quer-se provar que $suc(n) \in S$.

Se $n \in S$, então existe um único elemento $x \in X$ tal que $(n,x) \in u$. Como $(n,x) \in u$, segue que $(suc(n), f(x)) \in u$.

Se $suc(n) \notin S$, então $(suc(n), y) \in u$, para algum $y \neq f(x)$. Considere-se, neste caso, o conjunto $u - \{(suc(n), y)\}$. Observe-se que este conjunto contém $(0,a)$, uma vez que $suc(n) \neq 0$. E que se este conjunto contém (m,t), então também contém $(suc(m), f(t))$. De facto, se $m = n$, então $t = x$. Assim, o conjunto $u - \{(suc(n), y)\}$ contém $(suc(n), f(x))$ porque $f(x) \neq y$.

Por outro lado, se $m \neq n$, então a razão pela qual o conjunto $u - \{(suc(n), y)\}$ contém $(suc(m), f(t))$ é que $suc(m) \neq suc(n)$. Ou seja, $u - \{(suc(n), y)\} \in \mathcal{C}$, o que contradiz o facto de u ser o menor conjunto de $\mathcal{C}$, logo $suc(n) \in S$. ■

1.15 Aritmética

Soma

A introdução à adição dos números naturais é um exemplo de uma definição por indução. De facto, do teorema da recursão decorre que, para cada número natural m existe uma função S_m de w para w, tal que,

$$S_m(0) = m \tag{1.1}$$

e

$$S_m(suc(n)) = suc(S_m(n)) \tag{1.2}$$

para todo o natural n.

Por definição o valor de $S_m(n)$ é a soma $m + n$.

Propriedades

Associativa

$$(k + m) + n = k + (m + n), \quad \forall_{k,m,n \in w} \tag{1.3}$$

Prova. A prova será feita por indução em n.

Base de indução: se $n = 0$, então

$$(k+m)+0 \underset{(1.1)}{=} k+m \underset{(1.1)}{=} k+(m+0)\,.$$

Hipótese de indução: $(k+m)+n = k+(m+n)\,,\ \forall_{k,m,n\in w}$.
Tese de indução: $(k+m)+suc\,(n) = k+(m+suc\,(n))\,,\ \forall_{k,m,n\in w}$.

$$\begin{aligned}(k+m)+suc\,(n) &\underset{(1.2)}{=} suc\,((k+m)+n)\\ &\underset{\text{H.I.}}{=} suc\,(k+(m+n))\\ &\underset{(1.2)}{=} k+suc\,(m+n)\\ &\underset{(1.2)}{=} k+(m+suc\,(n))\,.\end{aligned}$$

■

Comutativa

$$m+n = n+m,\ \ \forall_{m,n\in w}. \tag{1.4}$$

Para provar esta propriedade, em primeiro lugar, tem-se que provar as duas igualdades seguintes.

$$(i)\ \ 0+n = n\ \ \forall_{n\in w};$$

$$(ii)\ \ suc\,(m)+n = suc\,(m+n)\ \ \forall_{m,n\in w}.$$

Prova. (i) A prova será feita por indução em n.
Base de indução: se $n = 0$, então $0+0 \underset{(1.1)}{=} 0$.
Hipótese de indução: $0+n = n,\ \forall_{n\in w}$.
Tese de indução: $0+suc\,(n) = suc\,(n)\ \ \forall_{n\in w}$.

$$0+suc\,(n) \underset{(1.2)}{=} suc\,(0+n) \underset{\text{H.I.}}{=} suc\,(n)\,.$$

■

Prova. (ii) A prova será feita por indução em n.
Base de indução: se $n = 0$, então

$$suc\,(m)+0 \underset{(1.2)}{=} suc\,(m+0)\,.$$

Hipótese de indução: $suc\,(m)+n = suc\,(m+n)\ \ \forall_{m,n\in w}$.
Tese de indução: $suc\,(m)+suc\,(n) = suc\,(m+suc\,(n))\ \ \forall_{m,n\in w}$.

$$\begin{aligned}suc\,(m)+suc\,(n) &\underset{(1.2)}{=} suc\,(suc\,(m)+n)\\ &\underset{\text{H.I.}}{=} suc\,(suc\,(m+n))\\ &\underset{(1.2)}{=} suc\,(m+suc\,(n))\,.\end{aligned}$$

■

Está-se agora em condições de provar a propriedade comutativa.

Prova. A prova será feita por indução em m.
Base de indução: se $m = 0$, então

$$0 + n \underset{(i)}{=} n \underset{(1.1)}{=} n + 0$$

Hipótese de indução: $m + n = n + m \quad \forall_{m,n \in w}$.
Tese de indução: $suc(m) + n = n + suc(m) \quad \forall_{m,n \in w}$.

$$\begin{aligned} suc(m) + n &\underset{(ii)}{=} suc(m + n) \\ &\underset{\text{H.I.}}{=} suc(n + m) \\ &\underset{(1.2)}{=} n + suc(m). \end{aligned}$$

■

Multiplicação

Para definir a multiplicação, aplica-se o teorema da recursão para construir a função P_m tal que, $\forall_{m \in w}$

$$P_m(0) = 0 \tag{1.5}$$

e

$$P_m(suc(n)) = P_m(n) + m, \ \forall_{n \in w}. \tag{1.6}$$

O valor de $P_m(n)$ é por definição o produto $m \cdot n$.

Antes de enunciar e provar algumas propriedades da multiplicação, é útil provar a seguinte igualdade.

$$\forall_{n \in w} \ suc(n) = 1 + n. \tag{1.7}$$

Prova. A prova será feita por indução em n.
Base de indução: se $n = 0$, como $suc(0) = 1$ por definição e $1 + 0 \underset{1.1}{=} 1$, então $suc(0) = 1 + 0$.
Hipótese de indução: $suc(n) = 1 + n$.
Tese de indução: $suc(suc(n)) = 1 + suc(n)$.

$$\begin{aligned} suc(suc(n)) &\underset{\text{H.I.}}{=} suc(1 + n) \\ &\underset{(1.2)}{=} 1 + suc(n). \end{aligned}$$

■

Propriedades

Distributiva à direita da multiplicação em relação à adição

$$k \cdot (m+n) = k \cdot m + k \cdot n, \ \forall_{k,m,n \in w}. \tag{1.8}$$

Prova. A prova será feita por indução em n.

Base de indução: se $n = 0$, então $k \cdot (m+0) \underset{(1.1)}{=} k \cdot m$. Por outro lado, $k \cdot m + k \cdot 0 \underset{(1.5)}{=} k \cdot m + 0 \underset{(1.1)}{=} k \cdot m$.

Hipótese de indução: $k \cdot (m+n) = k \cdot m + k \cdot n$

Tese de indução: $k \cdot (m + suc(n)) = k \cdot m + k \cdot suc(n)$.

$$\begin{aligned} k \cdot (m + suc(n)) &\underset{(1.2)}{=} k \cdot suc(m+n) \\ &\underset{(1.6)}{=} k \cdot (m+n) + k \\ &\underset{\text{H.I.}}{=} k \cdot m + k \cdot n + k \\ &\underset{(1.6)}{=} k \cdot m + k \cdot suc(n). \end{aligned}$$

■

Associativa

$$(k \cdot m) \cdot n = k \cdot (m \cdot n) \quad \forall_{k,m,n \in w}.$$

Prova. A prova será feita por indução em n.

Base de indução: se $n = 0$, então $(k \cdot m) \cdot 0 \underset{(1.5)}{=} 0$. Por outro lado, se $n = 0$, então $k \cdot (m \cdot 0) \underset{(1.5)}{=} k \cdot 0 \underset{(1.5)}{=} 0$.

Hipótese de indução: $(k \cdot m) \cdot n = k \cdot (m \cdot n) \quad \forall_{k,m,n \in w}$.

Tese de indução: $(k \cdot m) \cdot suc(n) = k \cdot (m \cdot suc(n)) \quad \forall_{k,m,n \in w}$.

$$\begin{aligned} (k \cdot m) \cdot suc(n) &\underset{(1.6)}{=} (k \cdot m) \cdot n + k \cdot m \\ &\underset{\text{H.I.}}{=} k \cdot (m \cdot n) + k \cdot m \\ &\underset{(1.8)}{=} k \cdot (m \cdot n + m) \\ &\underset{(1.6)}{=} k \cdot (m \cdot suc(n)). \end{aligned}$$

■

Comutativa

$$m \cdot n = n \cdot m \quad \forall_{m,n \in w}. \tag{1.9}$$

Antes de provar esta propriedade é necessário estabelecer, por indução, as igualdades seguintes:

$$(i) \quad 0 \cdot m = 0 \quad \forall_{m \in w}.$$

$$(ii)\quad suc(n)\cdot m = n\cdot m + m \quad \forall_{m,n\in w}.$$

Prova. (i) A prova será feita por indução em m.
Base de indução: se $m=0$, então $0\cdot 0 \underset{(1.5)}{=} 0$.
Hipótese de indução: $0\cdot m = 0 \quad \forall_{m\in w}$.
Tese de indução: $0\cdot suc(m) = 0 \quad \forall_{m\in w}$.

$$0\cdot suc(m) \underset{(1.6)}{=} 0\cdot m + 0 \underset{\text{H.I.}}{=} 0+0 \underset{(1.1)}{=} 0.$$

■

Prova. (ii) A prova será feita por indução em m.
Base de indução: se $m=0$, então $suc(n)\cdot 0 \underset{(1.5)}{=} 0$. Por outro lado, $0\cdot n+0 \underset{(1.5)}{=} 0+0 \underset{(1.1)}{=} 0$.
Hipótese de indução: $suc(n)\cdot m = n\cdot m + m \quad \forall_{m,n\in w}$.
Tese de indução: $suc(n)\cdot suc(m) = n\cdot suc(m) + suc(m) \quad \forall_{m,n\in w}$.

$$\begin{aligned} suc(n)\cdot suc(m) &\underset{(1.6)}{=} suc(n)\cdot m + suc(n) \\ &\underset{\text{H.I.}}{=} n\cdot m + m + suc(n) \\ &\underset{(1.2)}{=} n\cdot m + suc(m+n) \\ &\underset{(1.4)}{=} n\cdot m + suc(n+m) \\ &\underset{(1.2)}{=} n\cdot m + n + suc(m) \\ &\underset{(1.6)}{=} n\cdot suc(m) + suc(m). \end{aligned}$$

■

Agora está-se em condicões de provar a propriedade comutativa da multiplicação.
Prova. A prova será feita por indução em n.
Base de indução: se $n=0$, então $m\cdot 0 \underset{(1.5)}{=} 0 \underset{(i)}{=} 0\cdot m$.
Hipótese de indução: $m\cdot n = n\cdot m \quad \forall_{m,n\in w}$.
Tese de indução: $m\cdot suc(n) = suc(n)\cdot m \quad \forall_{m,n\in w}$.

$$\begin{aligned} m\cdot suc(n) &\underset{(1.6)}{=} m\cdot n + m \\ &\underset{\text{H.I.}}{=} n\cdot m + m \\ &\underset{(ii)}{=} suc(n) + m. \end{aligned}$$

■

Elemento neutro da multiplicação

$$1\cdot n = n \quad \forall_{n\in w} \tag{1.10}$$

Prova. A prova será feita por indução em n.

Base de indução: se $n = 0$, então $1 \cdot 0 \underset{(1.5)}{=} 0$.
Hipótese de indução: $1 \cdot n = n \quad \forall_{n \in w}$
Tese de indução: $1 \cdot suc(n) = suc(n) \quad \forall_{n \in w}$

$$\begin{aligned} 1 \cdot suc(n) &\underset{(1.6)}{=} 1 \cdot n + 1 \\ &\underset{\text{H.I.}}{=} n + 1 \\ &\underset{(1.7)}{=} suc(n). \end{aligned}$$

■

Distributiva à esquerda da multiplicação em relação à adição

$$(m + n) \cdot k = m \cdot k + n \cdot k, \ \forall_{k,m,n \in w}.$$

Prova. A prova será feita por indução em n.
Base de indução: se $n = 0$, então $(m + 0) \cdot k \underset{1.1}{=} m \cdot k$. Por outro lado, $m \cdot k + 0 \cdot k \underset{(1.9)}{=} m \cdot k + k \cdot 0 \underset{(1.5)}{=} m \cdot k + 0 \underset{(1.1)}{=} m \cdot k$.
Hipótese de indução: $(m + n) \cdot k = m \cdot k + n \cdot k$.
Tese de indução: $(m + suc(n)) \cdot k = m \cdot k + suc(n) \cdot k$.

$$\begin{aligned} (m + suc(n)) \cdot k &\underset{(1.2)}{=} suc(m + n) \cdot k \\ &\underset{(1.9)}{=} k \cdot suc(m + n) \\ &\underset{(1.6)}{=} k \cdot (m + n) + k \\ &\underset{(1.9)}{=} (m + n) \cdot k + k \\ &\underset{\text{H.I.}}{=} m \cdot k + n \cdot k + k \\ &\underset{(1.9)}{=} m \cdot k + k \cdot n + k \\ &\underset{(1.6)}{=} m \cdot k + k \cdot suc(n) \\ &\underset{(1.9)}{=} m \cdot k + suc(n) \cdot k. \end{aligned}$$

■

Potências

Para definir o conceito de potência, aplica-se novamente o teorema da recursão para construir a função E_m tal que

$$E_m(0) = 1 \quad \forall_{m \in w \setminus \{0\}} \tag{1.11}$$

e

$$E_m(suc(n)) = E_m(n) \cdot m \quad \forall_{m,n \in w}. \tag{1.12}$$

O valor de $E_m(n)$ é por definição a potência m^n.

Propriedades

$$1^n = 1 \tag{1.13}$$

Prova. A prova será feita por indução em n.
Base de indução: se $n = 0$, então $1^0 \underset{(1.11)}{=} 1$.

Hipótese de indução: $1^n = 1$.
Tese de indução: $1^{suc(n)} = 1$.

$$1^{suc(n)} \underset{(1.12)}{=} 1^n \cdot 1 \underset{\text{H.I.}}{=} 1 \cdot 1 \underset{(1.10)}{=} 1.$$

■

Multiplicação de potências com a mesma base

$$m^n \cdot m^k = m^{n+k} \quad \forall_{m,n,k \in w}. \tag{1.14}$$

Prova. A prova será feita por indução em n.
Base de indução: se $n = 0$, então $m^0 \cdot m^k \underset{(1.11)}{=} 1 \cdot m^k \underset{(1.10)}{=} m^k \underset{(1.1)}{=} m^{0+k}$.

Hipótese de indução: $m^n \cdot m^k = m^{n+k}$.
Tese de indução: $m^{suc(n)} \cdot m^k = m^{suc(n)+k}$.

$$\begin{aligned}
m^{suc(n)} \cdot m^k &\underset{(1.12)}{=} m^k \cdot m \cdot m^k \\
&\underset{(1.9)}{=} m^n \cdot m^k \cdot m \\
&\underset{\text{H.I.}}{=} m^{n+k} \cdot m \\
&\underset{(1.12)}{=} m^{suc(n+k)} \\
&\underset{(1.4)}{=} m^{suc(k+n)} \\
&\underset{(1.2)}{=} m^{k+suc(n)} \\
&\underset{(1.4)}{=} m^{suc(n)+k}.
\end{aligned}$$

■

Multiplicação de potências com o mesmo expoente

$$m^n \cdot p^n = (m \cdot p)^n \quad \forall_{m,n,p \in w}.$$

Prova. A prova será feita por indução em n.
Base de indução: se $n = 0$, então $m^0 \cdot p^0 \underset{(1.11)}{=} 1 \cdot 1 \underset{(1.10)}{=} 1 \underset{(1.11)}{=} (m \cdot p)^0$.

Hipótese de indução: $m^n \cdot p^n = (m \cdot p)^n$

Tese de indução: $m^{suc(n)} \cdot p^{suc(n)} = (m \cdot p)^{suc(n)}$

$$\begin{aligned} m^{suc(n)} \cdot p^{suc(n)} &\underset{(1.12)}{=} m^n \cdot m \cdot p^n \cdot p \\ &\underset{(1.9)}{=} m^n \cdot p^n \cdot m \cdot p \\ &\underset{\text{H.I.}}{=} (m \cdot p)^n \cdot (m \cdot p) \\ &\underset{(1.12)}{=} (m \cdot p)^{suc(n)}. \end{aligned}$$

■

Potência de potência

$$(m^n)^k = m^{n \cdot k} \quad \forall_{k,m,n \in w}.$$

Prova. A prova será feita por indução em k.

Base de indução: se $k = 0$, então $(m^n)^0 \underset{(1.11)}{=} 1$. Por outro lado $m^{n \cdot 0} \underset{(1.5)}{=} m^0 \underset{(1.11)}{=} 1$.

Hipótese de indução: $(m^n)^k = m^{n \cdot k}$

Tese de indução: $(m^n)^{suc(k)} = m^{n \cdot suc(k)}$

$$\begin{aligned} (m^n)^{suc(k)} &\underset{(1.12)}{=} (m^n)^k \cdot m^n \\ &\underset{\text{H.I.}}{=} m^{n \cdot k} \cdot m^n \\ &\underset{(1.14)}{=} m^{n \cdot k + n} \\ &\underset{(1.2)}{=} m^{n \cdot suc(k)}. \end{aligned}$$

■

1.16 Cardinalidade

O próximo assunto, que mereçe alguma atenção, é a teoria da ordenação no conjunto dos números naturais. Com este objectivo, examine-se o conceito de pertença entre os números naturais.

Definição 1.58 - *Diz-se que dois números naturais m e n são comparáveis se $m \in n$, ou $m = n$, ou $n \in m$.*

Axioma 1.59 - *Dois números naturais são sempre comparáveis.*

Antes de se provar este axioma é necessário introduzir alguma notação.

Para cada $n \in w$, $S(n)$ é o conjunto de todo o $m \in w$ tal que m e n são comparáveis. Seja S o conjunto de todos os n para os quais $S(n) = w$.

Prova. Inicia-se a prova mostrando que $S(0) = w$, ou seja, $0 \in S$.

Obviamente, $0 \in S(0)$, pois $0 = 0$. Se $m \in S(0)$,então, uma vez que $m \in 0$ é impossível, ou $m = 0$ e neste caso $0 \in suc(m)$, ou $0 \in m$ e também neste caso $0 \in suc(m)$. Portanto, em todos os casos, se $m \in S(0)$, então $suc(m) \in S(0)$, o que prova que $S(0) = w$.

Conclui-se a prova mostrando que se $S(n) = w$, então $S(suc(n)) = w$.

$0 \in S(suc(n))$, uma vez que $suc(n) \in S(0)$.

Falta provar que se $m \in S(suc(n))$, então $suc(n) \in S(suc(n))$.

Como $m \in S(suc(n))$, então ou $suc(n) \in m$ e neste caso $suc(n) \in suc(m)$, ou $suc(n) = m$ e também neste caso $suc(n) \in suc(m)$, ou $m \in suc(n)$. Neste último caso, ou $m = n$ e assim, $suc(m) = suc(n)$, ou $m \in n$. Este último caso divide-se de acordo com o comportamento de $suc(m)$ e n. Uma vez que para $suc(m) \in S(m)$ tem-se, $n \in suc(m)$, ou $n = suc(m)$, ou $suc(m) \in n$. A primeira possibilidade é imcompatível com $m \in n$, pois se $n \in suc(m)$, então $n \in m$ ou $m = n$, de tal modo que em cada caso $n \subset m$ e sabe-se que nenhum número natural é subconjunto de um dos seus elementos.

Ambas as possibilidades que restam, implicam que $suc(m) \in suc(n)$ e deste modo a prova está completa. ■

O axioma anterior implica que se $m, n \in w$, então uma das três possibilidades ($m \in n$, $m = n$, $n \in m$) é verificada e é fácil verificar que de facto apenas uma delas é verificada. Uma outra consequência é que se n e m são números naturais distintos, uma condição necessária e suficiente para $m \in n$ é que $m \subset n$.

Se $m \in n$, ou se de forma equivalente, $m \subset n$, escreve-se $m < n$, que se lê "m é menor que n".

Se m é menor ou igual a n, escreve-se $m \leq n$. Note-se que " $<$ " e " $\leq$ " são relações em w.

A relação $\leq$ é reflexiva e transitiva, mas não é simétrica. A relação $<$ é apenas transitiva.

Se $m \leq n$ e $n \leq m$, então $m = n$.

Definição 1.60 - *Dois conjuntos E e F são equivalentes, simbolicamente $E \sim F$, se existe uma correspondência injectiva entre eles.*

Proposição 1.61 - *Todo o subconjunto próprio de um número natural n é equivalente a algum número natural menor, (isto é, a algum elemento de n).*

Prova. A prova será feita por indução em n.

Seja $S = \{n$: para todo o n, para todo o m, se $m \subseteq n$, $m \neq n$, então existe $x \in n$ tal que $m \sim x\}$.

Base de indução: $0 \in S$, pois é verdade que para todo o m, se $m \subseteq 0$, $m \neq 0$, então existe $x \in 0$, tal que $m \sim x$.

Passo de indução: Assumindo que $n \in S$, quere-se provar que $suc(n) = n \cup \{n\} \in S$, ou seja, quere-se provar que se $m \subseteq n \cup \{n\}$ e $m \neq n \cup \{n\}$, então existe $x \in n \cup \{n\}$ tal que $m \sim x$.

Como $m \subseteq n \cup \{n\}$, tem-se três casos possíveis: $m \subset n$, $m = n$, ou $m \subseteq n \cup \{n\}$.

Se $m \subset n$, então por hipótese de indução, existe $x \in n \cup \{n\}$ tal que $m \sim x$.

Se $m = n$, então neste caso $x = n \in n \cup \{n\}$ e consequentemente $m \sim x$.

Se $m \subseteq n \cup \{n\}$, logo $n \in m$. Seja $m' = m \cap n$, por hipótese de indução, como $m' \subseteq n$, então existe $x' \in n$, tal que $m' \sim x'$. Considere-se $x = suc(x') \in n \cup \{n\}$, pois $x' \in n$ e assim, $m \sim x$. ■

Definição 1.62 *Um conjunto A é um conjunto finito se for equivalente a algum número natural, caso contrário, é um conjunto infinito.*

Proposição 1.63 *Um conjunto pode ser equivalente a no máximo um número natural.*

Um conjunto finito nunca é equivalente a um seu subconjunto próprio, ou seja, desde que se considere conjuntos finitos, o todo é sempre maior que qualquer uma das suas partes.

Uma vez que todo o subconjunto de um número natural é equivalente a um número natural, tem-se também que todo o subconjunto de um conjunto finito é finito.

O número de elementos de um conjunto finito A é, por definição, o único número natural equivalente a A. Este denota-se por $\sharp(A)$.

Proposição 1.64 *Se A e B são conjuntos finitos tais que $A \subseteq B$, então $\sharp(A) \leq \sharp(B)$.*

Proposição 1.65 *Se A e B são conjuntos finitos, então $A \cup B$ é finito.*

Proposição 1.66 *Se A e B são conjuntos finitos disjuntos, então $\sharp(A \cup B) = \sharp(A) + \sharp(B)$.*

Proposição 1.67 *Se A e B são conjuntos finitos, então também o são $A \times B$ e A^B.*

Proposição 1.68 *Se A e B são conjuntos finitos, então $\sharp(A \times B) = \sharp(A) \cdot \sharp(B)$ e $\sharp\left(A^B\right) = \sharp(A)^{\sharp(B)}$.*

Proposição 1.69 *Se A é um conjunto finito, então $\mathcal{P}(A)$ é finito*

Proposição 1.70 *Se A é um conjunto finito, então $\sharp(\mathcal{P}(A)) = 2^{\sharp(A)}$.*

Prova. A prova será feita por indução em $n = \sharp A$.

Pretende-se provar que qualquer que seja o conjunto A, se A tem n elementos então $\mathcal{P}(A)$ tem 2^n elementos.

Base de indução: Se $n = 0$ então o conjunto A não tem elementos, ou seja, $A = \varnothing$. Neste caso $\mathcal{P}(\varnothing) = \{\varnothing\}$ que tem $1 = 2^0$ elementos.

Hipótese de indução: Qualquer que seja o conjunto A, se A tem n elementos então $\mathcal{P}(A)$ tem 2^n elementos.

Tese de indução: Qualquer que seja o conjunto A, se A tem $n+1$ elementos então $\mathcal{P}(A)$ tem 2^{n+1} elementos.

Considere-se então o conjunto A qualquer com $n+1$ elementos. Fixe-se agora um elemento x de A e considere-se o conjunto B formado pelos restantes elementos de A. Como B tem n elementos então por hipótese de indução $\mathcal{P}(B)$ tem 2^n elementos.

Por outro lado, os subconjuntos de A são os subconjuntos de B mais os conjuntos que se obtêm juntando a estes o x. Assim, o número de elementos de $\mathcal{P}(A)$ é igual ao dobro do número de elementos de $\mathcal{P}(B)$. Logo, como $\mathcal{P}(B)$ tem 2^n elementos então $\mathcal{P}(A)$ tem $2 \times 2^n = 2^{n+1}$ elementos. ■

1.17 Ordem

Definição 1.71 *Uma relação R num conjunto X é* ***anti-simétrica*** *se, para todo $x, y \in X$, xRy e yRx, então $x = y$.*

Definição 1.72 *Uma* ***relação de ordem parcial*** *ou simplesmente* ***ordem*** *num conjunto X, é uma relação em X, reflexiva, anti-simétrica e transitiva.*

O símbolo mais utilizado para representar esta relação é o símbolo da desigualdade ($\leq$). Assim, uma relação de ordem parcial em X pode ser definida como a relação $\leq$ em X tal que, $\forall_{x,y,z \in X}$ tem-se:

(i) $x \leq x$;
(ii) se $x \leq y$ e $y \leq x$, então $x = y$;
(iii) se $x \leq y$ e $y \leq z$, então $x \leq z$.

Definição 1.73 *Se uma relação é uma relação de ordem parcial e para todo o $x, y \in X$, ou $x \leq y$ ou $y \leq x$, então a relação $\leq$ é uma* ***relação de ordem total.***

Observação 1.74 *Frequentemente, um conjunto totalmente ordenado é conhecido como* ***cadeia.***

Definição 1.75 - *Um conjunto parcialmente ordenado é um par ordenado $(X, \leq)$ onde X é um conjunto e a relação $\leq$ é uma relação de ordem parcial em X.*

Dir-se-á que "X é um conjunto parcialmente ordenado" em vez de "seja X o domínio de uma ordem parcial".

Suponha-se que X é um conjunto parcialmente ordenado e que x e y são elementos de X. Escreve-se $y \geq x$ quando $x \leq y$. Por outras palavras, $\geq$ é a inversa da relação $\leq$.

Se $x \leq y$ e $x \neq y$, escreve-se $x < y$, que se lê "x menor que y" ou "x é o **predecessor** de y". Nas mesmas condições escrever-se $y > x$ que se lê "y maior que x" ou "y é o **sucessor** de x".

A relação $<$ é tal que

(i) para nenhum elemento x e y, $x < y$ e $y < x$ se verificam simultaneamente e
(ii) se $x < y$ e $y < z$ então $x < z$, ou seja, a relação $<$ é transitiva.

Se X é um conjunto parcialmente ordenado, e se $a \in X$, o conjunto $\{x \in X : x < a\}$ é o **segmento inicial** determinado por a, que se denotará por $s(a)$.

O conjunto $\{x \in X : x \leq a\}$ é o **segmento inicial fraco** determinado por a, e será representado por $\overline{s}(a)$.

Seja X um conjunto parcialmente ordenado e E um subconjunto qualquer de X.

Definição 1.76 *Um elemento a de X é um minorante de E se e só se $\forall_{e \in E}$ $a \leq e$.*

Esta definição diz que um minorante de E é um elemento de X que é menor ou igual que todos os elementos de E.

Exemplo 1.77 *Seja $X = w$ e $E = \{4,5,6,7,8\}$. Qualquer número, pertencente a w, menor ou igual a 4 é um minorante de E.*

Definição 1.78 *Um elemento a de X é um majorante de E se e só se $\forall_{e\in E} \ e \leq a$.*

Neste caso, um majorante de E é um elemento de X que é maior ou igual que todos os elementos de E

Exemplo 1.79 *Seja $X = w$ e $E = \{4,5,6,7,8\}$. Qualquer número, pertencente a w, maior ou igual a 8 é um majorante de E.*

Definição 1.80 *Diz-se que E é minorado ou limitado inferiormente (em X), se e só se existir um minorante de E em X, ou seja, $\exists_{a\in X} \ \forall_{e\in E} \ a \leq e$.*

Definição 1.81 *Diz-se que E é majorado ou limitado superiormente (em X), se e só se E tiver pelo menos um majorante em X, ou seja, $\exists_{a\in X} \ \forall_{e\in E} \ e \leq a$.*

Definição 1.82 *Diz-se que E é limitado (em X) se e só se for minorado e majorado, ou seja, se e só se $\exists_{a,b\in X} \ \forall_{e\in E} \ a \leq e \leq b$.*

Exemplo 1.83 *Seja $X = w$ e $E = \{4,5,6,7,8\}$. O conjunto E é minorado e majorado, sendo portanto limitado.*

Definição 1.84 *Diz-se que um elemento a (de X) é um elemento minimal de E se e só se $a \in E$ e $\forall_{e\in E}$ (se $e \leq a$ então $a = e$).*

Exemplo 1.85 *Seja $X = w$ e $E = \{4,5,6,7,8\}$. O 4 é um elemento minimal de E.*

Definição 1.86 *Diz-se que um elemento a (de X) é um elemento maximal de E se e só se $a \in E$ e $\forall_{e\in E}$ (se $a \leq e$ então $a = e$).*

Exemplo 1.87 *Seja $X = w$ e $E = \{4,5,6,7,8\}$. O 8 é um elemento maximal de E.*

Definição 1.88 *Diz-se que um elemento a (de X) é mínimo de E se e só se $a \in E$ e $\forall_{e\in E} \ a \leq e$.*

Por outras palavras, um mínimo de E é um elemento de E que é menor ou igual que todos os elementos de E.

Observação 1.89 *Um subconjunto E de X pode não admitir mínimo, mas se admitir este é único.*

Exemplo 1.90 *Seja $X = w$ e $E = \{4,5,6,7,8\}$. 4 é o mínimo de E.*

Verifica-se ainda, que o mínimo de E (quando existe) é um minorante de E, e que qualquer minorante de E, que pertença a E, é um mínimo de E.

Tem-se ainda, que um mínimo de E é sempre um elemento minimal de E, mas o recíproco pode não se verificar, ou seja, um elemento minimal pode não ser mínimo.

Exemplo 1.91 *Seja* $X = \{1, 2, 3, \alpha, \beta, \gamma\}$ *e* R *a relação definida do seguinte modo:* $R = \{(1,1); (1,2); (1,3); (2,2); (2,3); (3,3); (\alpha,\alpha); (\alpha,\beta); (\alpha,\gamma); (\beta,\beta); (\beta,\gamma); (\gamma,\gamma)\}$. 1 *é um elemento minimal pois* $\forall_{x\in X}$ *se* $xR1$ *então* $1 = x$, *mas não é mínimo pois* $1 \not R \alpha$, $1 \not R \beta$ *e* $1 \not R \gamma$.

Definição 1.92 *Diz-se que um elemento* a *(de* X*) é máximo de* E *se e só se* $a \in E$ *e* $\forall_{e\in E}\ \ e \leq a$.

Observação 1.93 *Um subconjunto* E *de* X *pode não admitir máximo, mas se admitir este é único.*

Exemplo 1.94 *Seja* $X = w$ *e* $E = \{4, 5, 6, 7, 8\}$. 8 *é o máximo de* E.

Definição 1.95 *Diz-se que um elemento* a *de* X *é ínfimo de* E *se e só se* a *é minorante de* E *e qualquer outro minorante de* E *precede* a *de acordo com a ordem* $\leq$, *ou seja,* $\forall_{e\in E}\ \ a \leq e$ *e* $\forall_{x\in X}\ \ (\text{se } \forall_{e\in E}\ \ x \leq e \text{ então } x \leq a)$.

Assim, um ínfimo de um subconjunto E de X é o maior dos minorantes de E, ou seja, é o máximo do subconjunto de X formado por todos os minorantes de E em X. Como o máximo de um subconjunto de X pode não exsitir, mas se existir é único, então um subconjunto E de X pode não admitir ínfimo, mas se admitir é único.

Além disso, como se referiu acima, qualquer minorante de E que pertença a E é um mínimo de E. Logo, como o ínfimo de E é um minorante de E, se o ínfimo existir e pertencer a E, então o ínfimo de E coincide com o mínimo de E.

Por outro lado, se existir o mínimo de E ele coincide com o seu ínfimo.

Exemplo 1.96 *Seja* $X = w$ *e* $E = \{4, 5, 6, 7, 8\}$. 4 *é o ínfimo de* E.

Definição 1.97 *Diz-se que um elemento* a *de* X *é supremo de* E *se e só se* a *é majorante de* E *e qualquer outro majorante de* E *sucede* a *de acordo com a ordem* $\leq$, *ou seja,* $\forall_{e\in E}\ \ e \leq a$ *e* $\forall_{x\in X}\ \ (\text{se } \forall_{e\in E}\ \ e \leq x \text{ então } a \leq x)$.

Exemplo 1.98 *Seja* $X = w$ *e* $E = \{4, 5, 6, 7, 8\}$. 8 *é o supremo de* E.

1.18 Axioma da escolha

Sabe-se que qualquer conjunto ou é vazio, ou não o é. Se um conjunto é não vazio, sabe-se, por definição de conjunto vazio, que esse conjunto tem um elemento.

Se X e Y são conjuntos e se um deles é vazio, então o produto cartesiano $X \times Y$ é vazio. Por outro lado, se tanto X como Y não são vazios, então existe um elemento $x \in X$ e existe um elemento $y \in Y$, e assim, $(x, y) \in X \times Y$, deste modo, o produto cartesiano $X \times Y$ é não vazio. O recíproco também é verdade: se $X \times Y \neq \varnothing$ então existe $x \in X$ e $y \in Y$.

Se $\{x_i\}$ é uma sequência finita de conjuntos, onde $i \in n$, então uma condição necessária e suficiente para que o seu produto cartesiano seja vazio é que pelo menos um dos conjuntos seja vazio. Generalizando esta afirmação para famílias infinitas, obtém-se o seguinte axioma.

Axioma 1.99 ***(da escolha)*** **-** *O produto cartesiano de uma família não vazia de conjuntos não vazios é não vazio.*

Por outras palavras, se $\{X_i\}$ é uma família de conjuntos, não vazios, indexados por um conjunto não vazio I, então existe uma família $\{x_i\}$, $i \in I$, tal que $x_i \in X_i$ para cada $i \in I$.

Suponha-se que $\mathcal{C}$ é uma colecção não vazia de conjuntos não vazios. A colecção $\mathcal{C}$ pode ser vista como uma família, ou seja, pode ser convertida num conjunto indexado se se usar a própria colecção $\mathcal{C}$ como conjunto de índices e a identidade em $\mathcal{C}$, como indexante. Assim, o axioma da escolha diz que o produto cartesiano dos conjuntos da colecção $\mathcal{C}$ tem pelo menos um elemento. Um elemento de tal produto cartesiano é, por definição, uma função (família, conjunto indexado) cujo domínio é o conjunto de índices ($\mathcal{C}$) e cujo valor em cada índice pertence ao conjunto que origina esse índice. Assim, pode-se concluir que existe uma função f com domínio $\mathcal{C}$ tal que se $A \in \mathcal{C}$, então $f(A) \in A$. Esta função pode ser vista como sendo uma função que "escolhe" um elemento de cada um dos conjuntos da colecção $\mathcal{C}$ e é por este facto que o axioma se denomina "Axioma da escolha". Uma função com estas características é chamada *função escolha.*

Antes de se enunciar o axioma da escolha, já se podia garantir que, se a colecção de conjuntos for finita, então é possível "escolher" um elemento de cada conjunto, isto como consequência da definição de produto cartesiano. O papel do axioma anterior é garantir que tal é possível para uma colecção qualquer.

Apresenta-se agora um exemplo da aplicação do axioma da escolha.

Proposição 1.100 *Se um conjunto é infinito, então tem um subconjunto equivalente a w.*

Informalmente, a prova desta afirmação é feita do seguinte modo. Seja X um conjunto infinito, então X é não vazio, (ou seja, não é equivalente a $0 = \varnothing$), logo tem um elemento, por exemplo x_0. Uma vez que X não é equivalente a 1, então o conjunto $X - \{x_0\}$ é não vazio e portanto tem um elemento, x_1. O próximo passo é dizer que $X - \{x_0, x_1\}$ é não vazio e portanto tem um elemento x_2. Repita-se este

argumento infinitamente. O resultado é uma sequência infinita $\{x_n\}$ de elementos distintos de X. Deste modo, encontra-se um subconjunto de X equivalente a w.

Prova. Seja f a função escolha para X, ou seja, f é uma função de uma colecção de todos os subconjuntos não vazios de X para X, tal que $f(A) \in A$, para todo o A pertencente ao domínio de f. Seja $\mathcal{C}$ a colecção de todos os subconjuntos finitos de X. Uma vez que X é infinito, se $A \in \mathcal{C}$, então $X - A$ é um conjunto não vazio e portanto $X - A$ pertence ao domínio de f (pois $X - A$ é um subconjunto de X, não vazio). Seja g uma função de $\mathcal{C}$ para $\mathcal{C}$ definida por $g(A) = A \cup \{f(X - A)\}$. Por outras palavras, $g(A)$ é obtido juntando a A o elemento que f escolhe de $X - A$. Aplica-se agora o teorema da recursão à função g. Resulta deste modo que existe uma função U de w para $\mathcal{C}$ tal que $U(0) = \varnothing$ e $U(suc(n)) = g(U(n)) = U(n) \cup \{f(X - U(n))\}$ para todo o número natural n.

Asserção: se $v(n) = f(X - U(n))$, então v é um correspondência injectiva de w para X e assim, w é equivalente a algum subconjunto de X, nomeadamente o contradomínio de v. Para provar esta asserção, tem-se de fazer uma série de observações.

1- $v(n) \notin U(n)$ para todo o n.

2- $v(n) \in U(suc(n))$ para todo o n.

3- Se n e m são números naturais e $n \leq m$, então $U(n) \subseteq U(m)$.

4- Se n e m são números naturais e $n < m$, então $v(n) \neq v(m)$, pois $v(n) \in U(m)$, mas $v(m) \notin U(m)$.

A 4ª observação implica que a função v transforma números naturais distintos em elementos distintos de X. Deste modo, encontrou-se um subconjunto de X (contradomínio de v) que é equivalente a w e a prova está completa. ■

Como consequência deste resultado, tem-se a seguinte proposição.

Proposição 1.101 *Um conjunto é infinito se e só se é equivalente a um seu subconjunto próprio.*

Prova. Para se provar esta afirmação, tem-se que considerar dois casos.

1- Se X é um conjunto infinito, então X é equivalente a um seu subconjunto próprio. Para se provar esta afirmação, considere-se um conjunto infinito X e seja v uma correspondência injectiva de w para X. Se x é o contradomínio de v, $x = v(n)$ define-se $h(x) = v(suc(n))$. Se x não é o contradomínio de v, escreve-se $h(x) = x$. Verifica-se que h é uma correspondência injectiva de X para X. Uma vez que o contradomínio de h é um subconjunto próprio de X, pois não contém $v(0)$, a prova está completa.

2- Se X é equivalente a um seu subconjunto próprio, então X é um conjunto infinito. Esta afirmação é equivalente a dizer que, se X é um conjunto finito, então X não pode ser equivalente a um seu subconjunto próprio. Este facto decorre da definição de subconjunto próprio e do próprio conceito de equivalência entre conjuntos. ■

1.19 Lema de Zorn

Lema 1.102 ***(de Zorn)*** *- Se X é um conjunto parcialmente ordenado tal que toda a cadeia em X tem um majorante, então X contém um elemento maximal.*

Este lema diz que se se tiver um conjunto parcialmente ordenado X de tal modo que toda a cadeia em X[15] tem um majorante, então pode-se concluir que existe um elemento a em X tal que, se $a \leq x$, então $a = x$.

A ideia base da prova deste lema é a seguinte.

Uma vez que por hipótese o conjunto X é não vazio, então tem um elemento x_0. Se x_0 é maximal, então a prova está concluída, senão existe um elemento x_1 estritamente maior que x_0. Se x_1 é maximal, a prova termina, senão continua-se. Se se repetir este argumento infinitamente este levará a um elemento maximal.

Prova. Seja $\leq$ uma relação de ordem parcial qualquer em X.

O primeiro passo da prova é substituir a ordem parcial $\leq$ pela inclusão. Mais precisamente, considere-se para cada $x \in X$, o segmento inicial $\overline{s}(x) = \{y \in X : y \leq x\}$. Considere-se agora a função $\overline{s}$ de X para $\mathcal{P}(X)$ que a cada x faz corresponder $\overline{s}(x)$. O contradomínio $\overline{S}$ da função $\overline{s}$ é parcialmente ordenado pela inclusão. Verifica-se que $\overline{s}$ é injectiva e que $\overline{s}(x) \subseteq \overline{s}(y)$ se e só se $x \leq y$. Assim, a tarefa de encontrar um elemento maximal em X é a mesma que encontrar um elemento maximal em $\overline{S}$, ou seja, a cada cadeia maximal de $\overline{S}$ corresponde um elemento maximal de X,ou ainda x é um elemento maximal em X se e só se $\overline{s}(x)$ é um elemento maximal em $\overline{S}$.

Antes de provar a afirmação anterior é necessário mostrar que $\overline{s}(y) = \overline{s}(x)$ se e só se $x = y$.

Considere-se que $\overline{s}(y) = \overline{s}(x)$. Assim, $\overline{s}(y) \supseteq \overline{s}(x)$ e portanto $y \geq x$. Por outro lado, $\overline{s}(y) \subseteq \overline{s}(x)$, logo $y \leq x$. Então $x = y$ pois X é parcialmente ordenado.

Se $x = y$, então obviamente $\overline{s}(x) = \overline{s}(y)$.

1- Suponha-se que x é um elemento maximal em X, quer-se provar que $\overline{s}(x)$ é um elemento maximal em $\overline{S}$. Seja $\mathcal{C} \in \overline{S}$ tal que $\mathcal{C} \supseteq \overline{s}(x)$. Se $\mathcal{C} \in \overline{S}$, então existe um e um só y tal que $\mathcal{C} = \overline{s}(y)$. Assim, como $\overline{s}(y) \supseteq \overline{s}(x)$, então $y \geq x$. Como x é maximal em X, logo $x = y$ e portanto $\mathcal{C} = \overline{s}(x)$, ou seja, $\overline{s}(x) = \overline{s}(y)$. Logo $\overline{s}(x)$ é um elemento maximal em $\overline{S}$.

2- Suponha-se que $\mathcal{C}$ é um elemento maximal em $\overline{S}$. Considere-se que $\mathcal{C} = \overline{s}(x)$. Quer-se provar que x é um elemento maximal em X. Seja $y \geq x$. Então $\overline{s}(y) \supseteq \overline{s}(x)$. Como $\overline{s}(x)$ é um elemento maximal em $\overline{S}$, logo $\overline{s}(x) = \overline{s}(y)$, ou seja, $x = y$.

Além disso, toda a cadeia em X tem majorante se e só se toda a cadeia em $\overline{S}$ tem majorante. A prova desta afirmação é a seguinte.

Assuma-se que toda a cadeia em X tem majorante. Quer-se provar que toda a cadeia em $\overline{S}$ tem majorante. Seja $\underset{\shortparallel}{\mathcal{C}_1} \subseteq \underset{\shortparallel}{\mathcal{C}_2} \subseteq \underset{\shortparallel}{\mathcal{C}_3}$ uma cadeia em $\overline{S}$. Então
$s(x_1) \quad s(x_2) \quad s(x_3)$
existe uma cadeia $x_1 \leq x_2 \leq x_3 \ldots$ em X. Por hipótese esta cadeia tem majorante y. Como $x_1 \leq y$, então $s(x_1) = \mathcal{C}_1 \subseteq \overline{s}(y)$. O mesmo acontece para qualquer x_i que se considere. Portanto $\overline{s}(y)$ é majorante de $\overline{S}$.

[15] Uma cadeia em X é um subconjunto de X totalmente ordenado.

Assuma-se agora que toda a cadeia em $\overline{S}$ tem majorante. Quer-se provar que toda a cadeia em X tem majorante. Seja $x_1 \leq x_2 \leq x_3 ...$ em X. Então existe uma cadeia $s(x_1) \subseteq s(x_2) \subseteq s(x_3) ...$ em $\overline{S}$ que por hipótese tem majorante $\mathcal{C}$. Seja $\mathcal{C} = \overline{s}(y)$. Como $s(x_1) \subseteq \overline{s}(y)$, então $x_1 \leq y$. O mesmo acontece para qualquer $s(x_i)$ que se considere. Logo y é majorante de $x_1 \leq x_2 \leq x_3 ....$

Seja $\mathcal{X}$ o conjunto de todas as cadeias em X. Verifica-se que todo o elemento de $\mathcal{X}$ está contido em $\overline{s}(x)$ para algum $x \in X$, pois por hipótese toda a cadeia tem majorante. Por outras palavras, dada uma cadeia A, seja x um seu majorante, então $A \subseteq \overline{s}(x)$. Como o conjunto $\varnothing$ é (vacuosamente) uma cadeia tem-se que $\varnothing \in \mathcal{X}$ e assim, a colecção $\mathcal{X}$ é uma colecção de conjuntos não vazia parcialmente ordenada pela inclusão. Além disso, se $\mathcal{C}$ é uma cadeia em $\mathcal{X}$, então a união de todos os conjuntos em $\mathcal{C}$ pertence a $\mathcal{X}$, ou seja, $\cup\mathcal{C}$ é uma cadeia em X. Para verificá-lo considere-se $x_1, x_2 \in \cup\mathcal{C}$. Então x_1 e x_2 pertencem a pelo menos uma das cadeias de $\mathcal{C}$, sejam $\mathcal{C}_1$ e $\mathcal{C}_2$ essas cadeias, respectivamente. Como $\mathcal{C}$ é uma cadeia ou $\mathcal{C}_1 = \mathcal{C}_2$ ou $\mathcal{C}_1 \subset \mathcal{C}_2$ ou $\mathcal{C}_2 \subset \mathcal{C}_1$. Deste modo verifica-se que x_1 e x_2 pertencem à mesma cadeia de $\mathcal{C}$ ($\mathcal{C}_1$ ou $\mathcal{C}_2$). Seja $\mathcal{C}_i$ essa cadeia. Como $\mathcal{C}_i$ é uma cadeia e $x_1, x_2 \in \mathcal{C}_i$, então $x_1 = x_2$, ou $x_1 < x_2$, ou $x_2 < x_1$. Mostrou-se que dados x_1 e x_2 arbitrários em $\cup\mathcal{C}$, então $x_1 = x_2$, ou $x_1 < x_2$, ou $x_2 < x_1$, ou seja, $\cup\mathcal{C}$ é uma cadeia em X.

Antes de prosseguir a prova, repare-se que uma vez que cada elemento de $\mathcal{X}$ está contido em algum conjunto de $\overline{S}$, então se se passar de $\overline{S}$ para $\mathcal{X}$ não se estará a introduzir nenhum novo elemento maximal, ou seja, se M for maximal em $\mathcal{X}$ também é maximal em $\overline{S}$. A prova desta afirmação é a seguinte. Assuma-se que M é maximal em $\mathcal{X}$. Por hipótese todas as cadeias têm majorante m. É fácil ver que $M \subseteq \overline{s}(m)$. Como $\overline{s}(m) \in \mathcal{X}$ e M é maximal em $\mathcal{X}$, então $M = \overline{s}(m)$, logo $M \in \overline{S}$. Além disso, M também é maximal em $\overline{S}$.

Uma vantagem técnica de $\mathcal{X}$ é que se pode caracterizar os majorantes das cadeias. De facto, se C for uma cadeia em $\mathcal{X}$ então $\cup C$ é o (menor) majorante da cadeia.

Outra vantagem técnica de $\mathcal{X}$ é que contém todos os subconjuntos de cada um dos seus conjuntos. Isto faz com que seja possível alargar um conjunto não maximal de $\mathcal{X}$, acrescentado a este um elemento de cada vez.

Verifica-se de seguida que $\mathcal{X}$ tem elemento maximal. De facto vai-se verificar que todas as colecções que satisfazem as duas condições seguintes têm elemento maximal. Para tal considere-se uma colecção não vazia $\mathcal{X}'$ de subconjuntos não vazios de X, sujeita a duas condições:

1. se $A \subseteq B$ e $B \in \mathcal{X}'$, então $A \in \mathcal{X}'$;
2. se $\mathcal{C}$ é uma cadeia em $\mathcal{X}'$, então $\cup\mathcal{C} \in \mathcal{X}'$.

Note-se que o conjunto $\mathcal{X}$ das cadeias em X verifica as condições anteriores, como se mostra de seguida.

1. se $A \subseteq B$ e $B \in \mathcal{X}$, então é fácil concluir que A é uma cadeia de X, ou seja, $A \in \mathcal{X}$.
2. se $\mathcal{C}$ é uma cadeia em $\mathcal{X}$, então $\cup\mathcal{C} \in \mathcal{X}$, como já se viu anteriormente.

A primeira condição implica que $\varnothing \in \mathcal{X}'$. Agora pretende-se provar que existe em $\mathcal{X}'$ um conjunto maximal.

Seja f a função escolha para X, ou seja, $f : \mathcal{P}(X) \setminus \varnothing \to X$ tal que $f(A) \in A$ para todo $A \in \mathcal{P}(X) \setminus \varnothing$.

Para cada $A \in \mathcal{X}'$, seja $\widehat{A} = \{x \in X : A \cup \{x\} \in \mathcal{X}'\}$, ou seja, $\widehat{A}$ é o conjunto dos elementos de X que se podem acrescentar um a um ao conjunto A de modo a

que o conjunto assim obtido ainda pertença a $\mathcal{X}'$.

Seja g uma função de $\mathcal{X}'$ para $\mathcal{X}'$ definida como se segue:

$$\begin{aligned} \text{se } \widehat{A} - A &\neq \varnothing \text{ então } g(A) = A \cup \left\{ f\left(\widehat{A} - A\right) \right\} \\ \text{se } \widehat{A} - A &= \varnothing \text{ então } g(A) = A. \end{aligned}$$

Esta função tem como propriedade fundamental o facto de $g(A)$ (que contém sempre A) conter no máximo mais um elemento que A (o elemento que a função escolha f "escolhe" do conjunto $\widehat{A} - A$, quando este é não vazio).

Da definição de $\widehat{A}$ vem que $\widehat{A} - A = \varnothing$ se e só se A for maximal. A prova desta afirmação é a seguinte.

1- Se $\widehat{A} - A = \varnothing$ então A é maximal.

Suponha-se com vista a um absurdo que $\widehat{A} - A = \varnothing$ e A não é maximal. Então existe $B \neq A$ e $B \supseteq A$. Ou seja, existe um $b \in B$, tal que $b \notin A$. Assim, $A \cup \{b\} \subseteq B$ e $A \cup \{b\} \in \mathcal{X}'$ o que contaria o facto de $\widehat{A} - A = \varnothing$. Logo A é maximal.

2- Se A é maximal, então $\widehat{A} - A = \varnothing$.

Suponha-se com vista a um absurdo que A é maximal e $\widehat{A} - A \neq \varnothing$. Então existe $b \in X$ tal que $b \in \widehat{A}$ e $b \notin A$. Logo $A \cup \{b\} \in \mathcal{X}'$ e $A \cup \{b\} \supseteq A$, contrariando o facto de A ser maximal. Portanto $\widehat{A} - A = \varnothing$.

Assim, tem-se que provar que existe em $\mathcal{X}'$ um conjunto A tal que $g(A) = A$.

Agora para facilitar a exposição da prova introduz-se a definição seguinte.

Definição 1.103 *Uma subcolecção J de $\mathcal{X}'$ diz-se uma torre se:*

i) $\varnothing \in J$;
ii) se $A \in J$, *então* $g(A) \in J$;
iii) se $\mathcal{C}$ *é uma cadeia em* J, *então* $\cup \mathcal{C} \in J$.

Há pelo menos uma torre que é a colecção $\mathcal{X}'$ toda, pois esta verifica as três condições anteriores, como se pode ver de seguida.

i) $\varnothing \in \mathcal{X}'$;

ii) se $A \in \mathcal{X}'$, tem-se dois casos possíveis:

se $\widehat{A} - A = \varnothing$ então $g(A) = A \in \mathcal{X}'$.

se $\widehat{A} - A \neq \varnothing$ então $g(A) = A \cup \left\{ f\left(\widehat{A} - A\right) \right\}$. Por definição de $\widehat{A}$, o elemento que a função f "escolhe" em $\widehat{A} - A$ (que obviamente pertence a $\widehat{A}$) é um elemento que se se juntar ao conjunto A, faz com que o conjunto assim formado ainda pertença a $\mathcal{X}'$, ou seja, $A \cup \left\{ f\left(\widehat{A} - A\right) \right\} \in \mathcal{X}'$;

iii) se $\mathcal{C}$ é uma cadeia em $\mathcal{X}'$, então $\cup \mathcal{C} \in \mathcal{X}'$, por definição de $\mathcal{X}'$.

Seja T a colecção de todas as torres em $\mathcal{X}'$ e $J_0 = \cap T$. Então J_0 é uma torre e é a menor, pois verifica as três condições anteriormente referidas, como se pode ver de seguida.

i) como $\varnothing$ pertence a cada elemento de T, então $\varnothing$ pertence a J_0;

ii) se $A \in J_0$, então A pertence a todas as torres de T, assim $g(A)$ também pertence a todas as torres de T e portanto pertence a J_0;

iii) se $\mathcal{C}$ é uma cadeia em J_0, logo $\mathcal{C}$ é uma cadeia em cada $J \in T$, assim, para todo o $J \in T$, $\cup \mathcal{C} \in J$, então $\cup \mathcal{C} \in J_0$.

iv) Seja J uma torre qualquer de T, então obviamente $J_0 = \cap T \subseteq J$.

O próximo objectivo é provar que J_0 é uma cadeia. Para tal introduz-se a definição seguinte.

Definição 1.104 *Um conjunto $C \in J_0$ é comparável se para todo $A \in J_0$, $A \subseteq C$ ou $C \subseteq A$.*

Assim, dizer que J_0 é uma cadeia significa que todos os conjuntos em J_0 são comparáveis.

Seja C um conjunto comparável arbitrário.

Suponha-se que $A \in J_0$, $A \subseteq C$ e $A \neq C$. Uma vez que C é comparável, então $g(A) \subseteq C$ ou $C \subseteq g(A)$. Se $C \subseteq g(A)$ tem-se dois casos possíveis: $C = g(A)$ e neste caso obviamente $g(A) \subseteq C$, ou $C \subset g(A)$. O último caso não se verifica pois se assim não fosse, tinha-se $A \subset C \subset g(A)$ e logo $g(A) - A$ tinha mais do que um elemento, o que contradiz a definição da função g. Concluí-se assim, que $g(A) \subseteq C$.

Considere-se a colecção $\mathcal{U}$ de todos os conjuntos $A \in J_0$ para os quais $A \subseteq C$ ou $g(C) \subseteq A$, ou seja, $\mathcal{U} = \{A \in J_0 : A \subseteq C \text{ ou } g(C) \subseteq A\}$. A colecção $\mathcal{U}$ é mais "exigente" que a colecção de conjuntos em J_0 comparável a $g(C)$, pois se $A \in \mathcal{U}$, então, uma vez que $C \subseteq g(C)$, $A \subseteq g(C)$ ou $g(C) \subseteq A$.

Prova-se, em seguida, que $\mathcal{U}$ é uma torre.

i) Como $\varnothing \in J_0$ e $\varnothing \subseteq C$, então $\varnothing \in \mathcal{U}$;

ii) Quer-se provar que se $A \in \mathcal{U}$, então $g(A) \in \mathcal{U}$. Para tal considere-se os três casos seguintes:

1- se $A \subseteq C$ e $A \neq C$, então pelo que se apresentou acima $g(A) \subseteq C$. Assim, $g(A) \in \mathcal{U}$.

2- se $A = C$, então $g(A) = g(C)$ e obviamente $g(C) \subseteq g(A)$, logo $g(A) \in \mathcal{U}$.

3- se $g(C) \subseteq A$, então $g(C) \subseteq g(A)$, logo $g(A) \in \mathcal{U}$.

iii) Da definição de $\mathcal{U}$ decorre que se $\mathcal{C}$ é uma cadeia em $\mathcal{U}$, então $\cup\mathcal{C} \in \mathcal{U}$. Não é difícil ver que qualquer união de conjuntos em $\mathcal{U}$ pertence a $\mathcal{U}$.

Concluí-se assim, que $\mathcal{U}$ é uma torre contida em J_0. Mas como J_0 é a menor torre, então $\mathcal{U} = J_0$.

Das considerações acima concluí-se que para cada conjunto comparável C em J_0, o conjunto $g(C)$ é também comparável em J_0. Isto porque, se C é comparável em $J_0 = \mathcal{U}$, então para todo o $A \in J_0$, $A \subseteq C$ ou $g(C) \subseteq A$. Se $A \subseteq C$, então $A \subseteq g(C)$. Logo tem-se $A \subseteq g(C)$ ou $g(C) \subseteq A$, para todo o $A \in J_0$.

Sabe-se que $\varnothing$ é comparável e que a função g tansforma conjuntos comparáveis em conjuntos comparáveis. Além disso, a união de uma cadeia de comparáveis é comparável. Logo os conjuntos comparáveis em J_0 formam uma torre e portanto o conjunto dos conjuntos comparáveis em J_0 é igual a J_0. Assim, fica provado que J_0 é uma cadeia.

Se J_0 é uma cadeia, então $A = \cup J_0 \in J_0$. Uma vez que a união contém todos os conjuntos pertencentes a J_0, então $g(A) \subseteq A$. Como se tem sempre $A \subseteq g(A)$, então $A = g(A)$, ou seja, A é maximal em X. ■

Capítulo 2

Teoria de conjuntos em *Isabelle*

2.1 Introdução

Isabelle é um sistema genérico interactivo de prova que permite a prova de teoremas matemáticos, com um diálogo constante entre o utilizador e o computador. Foi desenvolvido por Lawrence C. Paulson (Universidade de Cambridge, Reino Unido) e Tobias Nipkow (Universidade Técnica de Munique, Alemanha). A sua primeira versão foi apresentada em 1986 e tem vindo a ser actualizado desde então. Este sistema permite ao utilizador representar e utilizar sistemas dedutivos para diferentes lógicas. Estas lógicas tanto podem ser as lógicas que estão disponíveis no sistema ou outras lógicas criadas pelo utilizador.

Para representar um sistema dedutivo para uma determinada lógica é necessário codificar as fórmulas e regras de inferência numa outra lógica, a *metalógica*. As fórmulas da metalógica são denominadas *metafórmulas* e a lógica que se pretende definir designa-se *lógica objecto*. Para se obter a linguagem da lógica objecto é necessário declarar as funções que representam as fórmulas objecto, os conectivos e os operadores relevantes. As fórmulas da lógica objecto são entendidas como metafórmulas atómicas. As regras de inferência da lógica objecto são codificadas como axiomas da metalógica (metafórmulas). A representação de um sistema dedutivo para a lógica objecto designa-se por *teoria*.

Depois de definir a representação de um sistema dedutivo, para uma determinada lógica, numa teoria, estabelecem-se metafórmulas que decorrem dessa teoria. Ao se estabelecerem estas metafórmulas está-se a mostrar que determinadas fórmulas decorrem, nesse sistema, de determinadas hipóteses. O processo de estabelecer metafórmulas em *Isabelle* é designado por *demonstração*. Uma demonstração pode ser automática ou assistida. Em qualquer um dos casos o sistema verifica a correcção de cada passo da prova.

Neste capítulo tem-se por objectivo fazer uma pequena introdução à teoria FOL (First-Order Logic) e à teoria *Isabelle* que codifica a teoria de conjuntos de *Zermelo-Fraenkel*. Esta teoria é designada por teoria ZF e implementa a teoria de *Zermelo-Fraenkel* como uma extensão da teoria FOL.

Nas secções seguintes, apresentam-se as definições e regras em *Isabelle* para alguns dos assuntos tratados no capítulo anterior. Apresenta-se também as provas

em *Isabelle* das propriedades da união, intersecção, complementação, conjunto das partes e pares ordenados (produto cartesiano), apresentadas no capítulo anterior. No entanto, algumas destas provas são remetidas para o apêndice, uma vez que os raciocínios utilizados são análogos. Para facilitar a leitura será mantida a numeração das referidas propriedades, quer neste capítulo, quer no apêndice.

Além disso, faz-se a prova por indução de algumas propriedades apresentadas no capítulo 1, na secção dedicada aos números naturais e à sua aritmética.

2.2 FOL

A lógica FOL é a *lógica clássica* e é obtida a partir da *lógica clássica intuicionista* (IFOL) juntando a regra *classical*.

2.2.1 Sintaxe

Constantes

Designação	Nome	Símbolo
Negação	Not	~
Tautologia	True	$\top$
Absurdo	False	$\bot$

Conectivos

Designação	Nome	Símbolo
Conjunção	conj	&
Disjunção	disj	|
Implicação	imp	--> [1]
Equivalência	iff	<->
Igualdade		=

Quantificadores

Designação	Nome	Símbolo
Universal	All	ALL
Existencial	Ex	EX
	Ex1	EX!

[1] O símbolo de implicação, em Isabelle, é representado com dois traços, no entanto ao longo do texto irá aparecer muitas vezes apenas com um traço. Isto deve-se à formatação automática do sistema.

2.2.2 Definições de abreviaturas

Designação	Definição
True_def	True == False -> False
not_def	~P == P -> False
iff_def	P <-> Q == (P -> Q) & (Q -> P)
ex1_def	Ex1(P) == EX x. P(x) & (ALL y. P(y) -> y = x)

2.2.3 Regras

Igualdade

Designação	Regra
refl	a = a
subst	[| a = b; P(a) |] ==> P(b)

Conjunção, disjunção e absurdo

Designação	Regra
conjI	[| P; Q |] ==> P & Q
conjunct1	P & Q ==> P
conjunct2	P & Q ==> Q
disjI1	P ==> P|Q
disjI2	Q ==> P|Q
disjE	[|P|Q; P ==> R; Q ==> R|]==> R
impI	(P ==> Q) ==> P -> Q
mp	[| P -> Q; P |] ==> Q
FalseE	False ==> P

Quantificadores

Designação	Regra
allI	(!!x. P(x)) ==> ALL x. P(x)
spec	ALL x. P(x) ==> P(x)
exI	P(x) ==> EX x. P(x)
exE	[| EX x. P(x); !!x. P(x) ==> R |] ==> R

Regras derivadas

Designação	Regra
sym	a = b ==> b = a
trans	[| a = b; b = c |] ==> a = c
ssubst	[| b = a; P(a) |] ==> P(b)
TrueI	True
notI	(P ==> False) ==> ~P
notE	[| ~P; P |] ==> R
iffI	[| P ==> Q; Q ==> P |] ==> P <-> Q
iffE	[| P <-> Q; [| P -> Q; Q -> P |] ==> R |] ==> R
iffD1	[| P <-> Q; P |] ==> Q
iffD2	[| P <-> Q; Q |] ==> P
ex1I	[| P(a); !!x. P(x) ==> x = a |] ==> EX! x. P(x)
ex1E	[| EX! x. P(x); !!x. [| P(x); ALL y. P(y) -> y = x |] ==> R |]==>R
excluded_middle	~P | P
disjCI	(~Q ==> P) ==> P | Q
exCI	(ALL x. ~P(x) ==> P(a)) ==> EX x. P(x)
impCE	[| P -> Q; ~P ==> R; Q ==> R |] ==> R
iffCE	[| P <-> Q; [| P; Q |] ==> R; [| ~P; ~Q |] ==> R |] ==> R
notnotD	~~P ==> P
swap	[| ~Pa; ~P ==> Pa |] ==> P

Classical

Designação	Regra
classical	(~P ==> P) ==> P

2.2.4 Exemplos de provas

Para iniciar a prova de proposições em *Isabelle* FOL à que em primeiro lugar iniciar o *Isabelle* indicando que se pretendo utilizar o FOL. Para tal basta escrever na linha de comandos

isabelle FOL

Uma prova em *Isabelle* é feita estabelecendo objectivos (ou subobjectivos) através da aplicação de tácticas. Estas tácticas dão por sua vez origem a novos subobjectivos.

1. {A&B->C;A}$\vdash_{FOL}$B -> C.

Ao iniciar uma prova é necessário indicar o que se quer provar, ou seja, qual o objectivo. Para tal, e neste caso, escreve-se na linha de comandos

Goal "[|(A&B)->C;A|]==>(B->C)";

Obtém-se assim:

```
> Goal "[|(A&B)->C;A|]==>(B->C)";
Level 0 (1 subgoal)
[| A & B -> C; A |] ==> B -> C
1. [| A & B -> C; A |] ==> B -> C
val it = [] : Thm.thm list
```

Neste exemplo, tem-se um subobjectivo que é a meta-fórmula

1. [| A & B -> C; A|] ==> B -> C,

neste caso numerado por 1. Este significa que é preciso estabelecer B -> C a partir da hipóteses A&B->C e A. A conclusão B->C decorre da aplicação de regra de introdução da implicação, designada por impI e que é o meta-axioma (?P==>?Q)==>?P->?Q. Durante a aplicação desta regra, em primeiro lugar é feita a identificação das fórmulas adequadas para ?P e ?Q. Para tal o sistema identifica ?P->?Q como sendo B->C, verificando assim, que ?P = B e ?Q = C. Depois, como (?P ==> ?Q) ==> ?P -> Q? significa que para obter ?P -> ?Q é necessário a partir de ?P deduzir ?Q, o sistema apresenta um novo subobjectivo no qual ?P e ?Q já aparecem substituidos pelas fórmulas convenientes.

1. [| A&B -> C; A; B |] ==> C

O processo descrito, de identificação das fórmulas convenientes e a sua substituição, é designado por *unificação*. Este associado à apresentação de novos subobjectivos designa-se por *resolução* e é obtido em *Isabelle* através da *táctica de resolução resolve_tac [regras] i*, que quando aplicada apenas a uma regra pode ser abreviada por *rtac regra i*. A aplicação *by (rtac regra i)* é abreviada por *br regra 1*. Neste exemplo,

by (rtac impI 1);

ou abreviadamente

br impI 1;

O 1 introduzido após a táctica significa que se está a aplicar aquela táctica de resolução ao subobjectivo numerado por 1. Isto é importante quando se tem mais do que um subobjectivo, como se verá mais à frente.

```
> br impI 1;
Level 1 (1 subgoal)
[| A & B -> C; A |] ==> B -> C
1. [| A & B -> C; A; B |] ==> C
```

```
val it = () : unit
```

O próximo passo da prova é estabelecer C a partir das hipóteses A&B -> C, A e B. A conclusão C advém da aplicação da regra mp ([| ?P -> ?Q; ?P |] ==> ?Q). Esta significa que para se concluir ?Q é necessário estabelecer ?P-->?Q e ?P, ou seja, tem-se dois novos subobjectivos.

```
> br mp 1;
Level 2 (2 subgoals)
[| A & B -> C; A |] ==> B -> C
1. [| A & B -> C; A; B |] ==> ?P1 -> C
2. [| A & B -> C; A; B |] ==> ?P1
val it = () : unit
```

Observando o primeiro subobjectivo verifica-se que surgiu ?P1, ou seja, uma fórmula que ainda não foi substituída. Neste caso o que se pretende é ?P1 = A & B e portanto ?P1 -> C = A & B -> C. Para tal basta utilizar um outro tipo de táctica, uma táctica que compara sintacticamente a conclusão do subobjectivo com as respectivas hipóteses do lado esquerdo. Esta é designada por *táctica de unificação com hipóteses* (*assume_tac i*). Na linha de comandos digita-se

```
by (assume_tac 1);
```

ou abreviadamente

```
by (atac 1);
```

ou ainda

```
ba 1;
```

e obtém-se:

```
Level 3 (1 subgoal)
[| A & B -> C; A |] ==> B -> C
1. [| A & B -> C; A; B |] ==> A & B
val it = () : unit
```

O novo subobjectivo apresentado diz que é necessário, a partir das hipótese A & B -> C, A e B, obter A & B. Para tal utiliza-se a regra conjI ([| P; Q |] ==> P & Q).

```
> br conjI 1;
Level 4 (2 subgoals)
[| A & B -> C; A |] ==> B -> C
1. [| A & B -> C; A; B |] ==> A
2. [| A & B -> C; A; B |] ==> B
val it = () : unit
```

Obtém-se assim dois novos subobjectivos que são resolvidos utilizando a táctica de unificação com hipóteses.

```
> ba 1;
Level 5 (1 subgoal)
[| A & B -> C; A |] ==> B -> C
1. [| A & B -> C; A; B |] ==> B
val it = () : unit
> ba 1;
Level 6
[| A & B -> C; A |] ==> B -> C
No subgoals!
val it = () : unit
```

Para se poder efectuar provas que envolvem abreviaturas é necessário reescrever as fórmulas de acordo com as definições correspondentes. Para tal, pode-se utilizar o comando

Goalw [definições] "metafórmula";

Este reescreve as abreviaturas de acordo com a definições, em todas as fórmulas que estão presentes na metafórmula. Ainda assim, no decorrer da prova podem surgir novamente fórmulas que utilizam abreviaturas. Neste caso, será necessário reescrever a fórmula, pois como o objectivo está escrito de forma abrevida e as hipóteses não, ou seja, as fórmulas têm estrutura sintáctica diferente, então a unificação falha. Para resolver este problema basta digitar a táctica

by (rewrite_goals_tac [definições]);

O exemplo seguinte ilustra o uso do comando Goalw e da táctica anterior para a abreviatura da negação (~P == P -> False).

2. $\vdash_{FOL}$ ~~p&~~(p->q)->~~q

```
> Goalw [not_def] "~~p&~~(p->q)->~~q";
Level 0 (1 subgoal)
~~p & ~~(p -> q) -> ~~q
1. ((p -> False) -> False) & (((p -> q) -> False) -> False) ->
(q -> False) -> False
val it = [] : Thm.thm list
> br impI 1;
Level 1 (1 subgoal)
~~p & ~~(p -> q) -> ~~q
1. ((p -> False) -> False) & (((p -> q) -> False) -> False)
==> (q -> False) -> False
val it = () : unit
> br impI 1;
Level 2 (1 subgoal)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> False
val it = () : unit
```

```
> br mp 1;
Level 3 (2 subgoals)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P2 -> False
2. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P2
val it = () : unit
> ba 1;
Level 4 (1 subgoal)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> q
val it = () : unit
> br mp 1;
Level 5 (2 subgoals)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P3 -> q
2. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P3
val it = () : unit
> br notnotD 2;
Level 6 (2 subgoals)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P3 -> q
2. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ~~?P3
val it = () : unit
> br conjunct1 2;
Level 7 (2 subgoals)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P3 -> q
2. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ~~?P3 & ?Q5
val it = () : unit
```

Com se pretende que no subobjectivo 2, ~~?P3 & ?Q5 seja ((p -> False) -> False) & (((p -> q) -> False) -> False), então tem-se de reescrever o subobjectivo de modo a que a unificação não falhe.

```
> by (rewrite_goals_tac [not_def]);
Level 8 (2 subgoals)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P3 -> q
2. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ((?P3 -> False) -> False) & ?Q5
val it = () : unit
```

Agora é só utilizar a táctica de unificação com hipóteses para se obter o desejado.

```
> ba 2;
Level 9 (1 subgoal)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> p -> q
val it = () : unit
> br notnotD 1;
Level 10 (1 subgoal)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ~~(p -> q)
val it = () : unit
> br conjunct2 1;
Level 11 (1 subgoal)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P7 & ~~(p -> q)
val it = () : unit
> by (rewrite_goals_tac [not_def]);
Level 12 (1 subgoal)
~~p & ~~(p -> q) -> ~~q
1. [| ((p -> False) -> False) & (((p -> q) -> False) -> False);
q -> False |]
==> ?P7 & (((p -> q) -> False) -> False)
val it = () : unit
> ba 1;
```

```
Level 13
~~p & ~~(p -> q) -> ~~q
No subgoals!
val it = () : unit
```

3. $\vdash_{FOL} \exists_y \forall_x Q(x,y) \rightarrow \forall_x \exists_y Q(x,y)$.

```
> Goal "(EX y. ALL x. Q(x,y))->(ALL x. EX y. Q(x,y))";
Level 0 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. (EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
val it = [] : Thm.thm list
> br impI 1;
Level 1 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. EX y. ALL x. Q(x, y) ==> ALL x. EX y. Q(x, y)
val it = () : unit
> br allI 1;
Level 2 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. !!x. EX y. ALL x. Q(x, y) ==> EX y. Q(x, y)
val it = () : unit
```

Neste momento da prova surge a necessidade de utilizar a regra *exE* que é a metafórmula [| EX x. ?P(x); !!x. ?P(x) ==> ?R |] ==> ?R. Esta significa que para obter ?R é necessário ter como hipótese EX x. ?P(x) e que a partir de !!x. ?P(x) se tem de deduzir ?R.

Se se aplicar *resolve_tac [exE] 1* obtém-se:

```
Level 3 (2 subgoals)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. !!x. EX y. ALL x. Q(x, y) ==> EX xa. ?P2(x, xa)
2. !!x xa. [| EX y. ALL x. Q(x, y); ?P2(x, xa) |] ==> EX y. Q(x, y)
val it = () : unit
```

Neste caso, será necessário usar a táctica de unificação com as hipóteses aplicada ao primeiro subobjectivo, pois o que se pretende é que EX xa. ?P2(x, xa) seja EX y. ALL x. Q(x, y) e assim, ?P2(x, xa) seja ALL x. Q(x, y).

```
> ba 1;
Level 4 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. !!x xa. [| EX y. ALL x. Q(x, y); ALL x. Q(x, xa) |] ==> EX y. Q(x, y)
val it = () : unit
```

No entanto, além da táctica de resolução resolve_tac [regras] i existe também a *táctica de resolução com eliminação* que é equivalente à resolução seguida de unificação da primeira premissa, por hipótese. Esta regra usa-se para regras de

eliminação, ou seja, regras esquerdas e é a táctica *eresolve_tac [regras] i.* A táctica *eresolve_tac [regra] i*, pode ser abreviada por *etac regra i.* A aplicação *by (etac regra i)* é abreviada por *be regra i.*

No entanto, por vezes esta táctica não poderá ser utilizada, isto acontece quando a unificação da primeira permissa, por hipótese não é imediata.

De seguida apresenta-se a prova completa da propriedade 3 recorrendo à táctica *etac.*

```
> Goal "(EX y. ALL x. Q(x,y))->(ALL x. EX y. Q(x,y))";
Level 0 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. (EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
val it = [] : Thm.thm list
> br impI 1;
Level 1 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. EX y. ALL x. Q(x, y) ==> ALL x. EX y. Q(x, y)
val it = () : unit
> br allI 1;
Level 2 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. !!x. EX y. ALL x. Q(x, y) ==> EX y. Q(x, y)
val it = () : unit
> be exE 1;
Level 3 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. !!x y. ALL x. Q(x, y) ==> EX y. Q(x, y)
val it = () : unit
> br exI 1;
Level 4 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. !!x y. ALL x. Q(x, y) ==> Q(x, ?y3(x, y))
val it = () : unit
> br spec 1;
Level 5 (1 subgoal)
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
1. !!x y. ALL x. Q(x, y) ==> ALL xa. Q(xa, ?y6(x, xa, y))
val it = () : unit
> ba 1;
Level 6
(EX y. ALL x. Q(x, y)) -> (ALL x. EX y. Q(x, y))
No subgoals!
val it = () : unit
```

2.3 ZF

Durante esta secção serão apresentadas as representações básicas da teoria ZF que permitem introduzir a teoria de conjuntos descrita no capítulo anterior. Depois passar-se-á à apresentação das provas em *Isabelle* das propriedades da união, da intersecção, da complementaridade, dos conjuntos das partes, etc. seguindo a ordem pela qual estes conceitos foram referidos no capítulo anterior.

Como já foi referido anteriormente, a teoria ZF implementa a teoria de conjuntos de Zermelo-Fraenkel como uma extensão da teoria FOL, portanto todas as regras disponíveis na teoria FOL estão também disponíveis na teoria ZF.

Em *Isabelle* ZF o conjunto vazio ($\varnothing$) é representado por 0 e o símbolo de pertençe por : . Assim, por exemplo se se quiser representar $x \in A$, digita-se x : A. No caso de se querer $x \notin A$, escreve-se x~:A.

2.3.1 Axioma da extensionalidade e subconjuntos

No capítulo anterior, o primeiro axioma apresentado é o *axioma da extensionalidade.* Relembrando

$$(A = B) \Leftrightarrow (\forall_x \ (x \in A \Leftrightarrow x \in B)) .$$

Em seguida, definiu-se o conceito de *subconjunto.* Em ZF esta definição designa-se por *subset_def* e é representada por:

$$\text{A <= B == ALL x:A. x:B} \ ,$$

onde <= representa o símbolo $\subseteq$.

Verificou-se após esta definição que o axioma da extensionalidade podia ser reformulado dizendo que:

$$(A = B) \Leftrightarrow A \subseteq B \wedge B \subseteq A.$$

Em ZF este é designado por *extension* e codificado do seguinte modo:

$$\text{A = B <-> A <= B \& B <= A}$$

Como consequência deste axioma e do conceito de subconjunto, em ZF foram derivadas as seguintes regras para subconjuntos, para a igualdade de conjuntos e para o conjunto vazio.

Designação	Regra
subsetI	(!!x. x : A ==> x : B) ==> A <= B
subsetD	[| A <= B; c : A |] ==> c : B
subsetCE	[| A <= B; c ~: A ==> P; c : B ==> P |] ==> P
subset_refl	A <= A
subset_trans	[| A <= B; B <= C |] ==> A <= C
equalityI	[| A <= B; B <= A |] ==> A = B
equalityD1	A = B ==> A <= B
equalityD2	A = B ==> B <= A
equalityE	[| A = B; [| A <= B; B <= A |] ==> P |] ==> P
emptyE	a : 0 ==> R
empty-subsetI	0 <= A
equals0I	(!!y. y : A ==> False) ==> A = 0
equals0D	A = 0 ==> a ~: A

2.3.2 Uniões e intersecções

União

O *axioma da união* foi o segundo axioma apresentado no capítulo anterior. Em ZF este é designado por *Union_iff* e é codificado do seguinte modo:

A : Union(C) <-> (EX B:C. A:B)

onde Union(C) é a representação em ZF da união de uma classe de conjuntos ($\cup C$).

A união de uma classe que contenha apenas dois conjuntos A e B é designada por Un e a sua definição é a seguinte:

Un_def A Un B == Union(Upair(A,B))

onde Upair é a designação em ZF de par não ordenado. Assim, Upair(A,B) representa o conjunto $\{A, B\}$.

Se se desejar denotar o conjunto singular $\{A\}$ utiliza-se Upair(A,A).

Em ZF tem-se as seguintes regras para a união generalizada (Union) e para a união de dois conjuntos (Un).

Designação	**Regra**
UnionI	[| B : C; A : B |] ==> A : Union(C)
UnionE	[| A : Union(C); !!B. [| A : B; B : C |] ==> R |] ==> R
Union_upper	B : A ==> B <= Union(A)
Union_least	(!!x. x : A ==> x <= C) ==> Union(A) <= C
UnI1	c : A ==> c : A Un B
UnI2	c : B ==> c : A Un B
UnCI	(c ~: B ==> c : A) ==> c : A Un B
UnE	[| c : A Un B; c : A ==> P; c : B ==> P |] ==> P
Un_upper1	A <= A Un B"
Un_upper2	B <= A Un B
Un_least	[| A <= C; B <= C |] ==> A Un B <= C

De seguida apresenta-se provas de algumas das propriedades da união de conjuntos apresentadas na secção 1.5.

2. $\vdash_{ZF} \cup\{A\} = A$

```
> Goal "Union({A})=A";
Level 0 (1 subgoal)
Union({A}) = A
1. Union({A}) = A
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
Union({A}) = A
1. Union({A}) <= A
2. A <= Union({A})
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
Union({A}) = A
1. !!x. x : Union({A}) ==> x : A
2. A <= Union({A})
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
Union({A}) = A
1. !!x. x : Union({A}) ==> x : A
2. !!x. x : A ==> x : Union({A})
val it = () : unit
> be UnionE 1;
```

```
Level 4 (2 subgoals)
Union({A}) = A
1. !!x B. [| x : B; B : {A} |] ==> x : A
2. !!x. x : A ==> x : Union({A})
val it = () : unit
> be singletonE 1;
Level 5 (2 subgoals)
Union({A}) = A
1. !!x B. [| x : B; B = A |] ==> x : A
2. !!x. x : A ==> x : Union({A})
val it = () : unit
> be equalityE 1;
Level 6 (2 subgoals)
Union({A}) = A
1. !!x B. [| x : B; B <= A; A <= B |] ==> x : A
2. !!x. x : A ==> x : Union({A})
val it = () : unit
> be subsetD 1;
Level 7 (2 subgoals)
Union({A}) = A
1. !!x B. [| x : B; A <= B |] ==> x : B
2. !!x. x : A ==> x : Union({A})
val it = () : unit
> ba 1;
Level 8 (1 subgoal)
Union({A}) = A
1. !!x. x : A ==> x : Union({A})
val it = () : unit
> br UnionI 1;
Level 9 (2 subgoals)
Union({A}) = A
1. !!x. x : A ==> ?B7(x) : {A}
2. !!x. x : A ==> x : ?B7(x)
val it = () : unit
> ba 2;
Level 10 (1 subgoal)
Union({A}) = A
1. !!x. x : A ==> A : {A}
val it = () : unit
> br singletonI 1;
Level 11
Union({A}) = A
No subgoals!
val it = () : unit
```

4. $\vdash_{ZF} A \cup B = B \cup A$

```
> Goal "A Un B = B Un A";
Level 0 (1 subgoal)
A Un B = B Un A
1. A Un B = B Un A
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Un B = B Un A
1. A Un B <= B Un A
2. B Un A <= A Un B
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Un B = B Un A
1. !!x. x : A Un B ==> x : B Un A
2. B Un A <= A Un B
val it = () : unit
> be UnE 1;
Level 3 (3 subgoals)
A Un B = B Un A
1. !!x. x : A ==> x : B Un A
2. !!x. x : B ==> x : B Un A
3. B Un A <= A Un B
val it = () : unit
> br UnI2 1;
Level 4 (3 subgoals)
A Un B = B Un A
1. !!x. x : A ==> x : A
2. !!x. x : B ==> x : B Un A
3. B Un A <= A Un B
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
A Un B = B Un A
1. !!x. x : B ==> x : B Un A
2. B Un A <= A Un B
val it = () : unit
> br UnI1 1;
Level 6 (2 subgoals)
A Un B = B Un A
1. !!x. x : B ==> x : B
2. B Un A <= A Un B
val it = () : unit
> ba 1;
Level 7 (1 subgoal)
```

```
A Un B = B Un A
1. B Un A <= A Un B
val it = () : unit
> br subsetI 1;
Level 8 (1 subgoal)
A Un B = B Un A
1. !!x. x : B Un A ==> x : A Un B
val it = () : unit
> be UnE 1;
Level 9 (2 subgoals)
A Un B = B Un A
1. !!x. x : B ==> x : A Un B
2. !!x. x : A ==> x : A Un B
val it = () : unit
> br UnI2 1;
Level 10 (2 subgoals)
A Un B = B Un A
1. !!x. x : B ==> x : B
2. !!x. x : A ==> x : A Un B
val it = () : unit
> ba 1;
Level 11 (1 subgoal)
A Un B = B Un A
1. !!x. x : A ==> x : A Un B
val it = () : unit
> br UnI1 1;
Level 12 (1 subgoal)
A Un B = B Un A
1. !!x. x : A ==> x : A
val it = () : unit
> ba 1;
Level 13
A Un B = B Un A
No subgoals!
val it = () : unit
```

As provas das restantes propriedades da união são semelhantes às anteriores e podem ser vistas na subsecção A.2.1 do apêndice.

Intersecção

A definição da intersecção de uma classe A em ZF é designada por *Inter_def* e é codificada por

```
Inter(A) == {x: Union(A) . ALL y:A. x : y}
```

A intersecção de dois conjuntos A e B é denotada por A Int B e é definida por

```
Int_def   A Int B == Inter(Upair(A,B))
```

Em ZF tem-se as seguintes regras para a intersecção generalizada (Inter) e para a intersecção de dois conjuntos (Int).

Designação	**Regra**
InterI	[| !!x. x : C ==> A : x; C ~= 0 |] ==> A : Inter(C)
InterD	[| A : Inter(C); B : C |] ==> A : B
InterE	[| A : Inter(C); B ~: C ==> R; A : B ==> R |] ==> R
Inter_lower	B : A ==> Inter(A) <= B
Inter_greatest	[| A ~= 0; !!x. x : A ==> C <= x |] ==> C <= Inter(A)
IntI	[| c : A; c : B |] ==> c : A Int B
IntD1	c : A Int B ==> c : A
IntD2	c : A Int B ==> c : B
IntE	[| c : A Int B; [| c : A; c : B |] ==> P |] ==> P
Int_lower1	A Int B <= A
Int_lower2	A Int B <= B
Int_greatest	[| C <= A; C <= B |] ==> C <= A Int B

De seguida apresenta-se a prova de algumas das propriedades da intersecção de conjuntos apresentadas na secção 1.5.

1. $\vdash A \cap \varnothing = \varnothing$

```
> Goal "A Int 0 = 0";
Level 0 (1 subgoal)
A Int 0 = 0
1. A Int 0 = 0
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Int 0 = 0
1. A Int 0 <= 0
2. 0 <= A Int 0
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Int 0 = 0
1. !!x. x : A Int 0 ==> x : 0
2. 0 <= A Int 0
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
A Int 0 = 0
```

```
1. !!x. x : A Int 0 ==> x : 0
2. !!x. x : 0 ==> x : A Int 0
val it = () : unit
> br emptyE 2;
Level 4 (2 subgoals)
A Int 0 = 0
1. !!x. x : A Int 0 ==> x : 0
2. !!x. x : 0 ==> ?a3(x) : 0
val it = () : unit
> ba 2;
Level 5 (1 subgoal)
A Int 0 = 0
1. !!x. x : A Int 0 ==> x : 0
val it = () : unit
> be IntD2 1;
Level 6
A Int 0 = 0
No subgoals!
val it = () : unit
```

2. $\vdash_{ZF} A \cap B = B \cap A$

```
> Goal "A Int B = B Int A";
Level 0 (1 subgoal)
A Int B = B Int A
1. A Int B = B Int A
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Int B = B Int A
1. A Int B <= B Int A
2. B Int A <= A Int B
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Int B = B Int A
1. !!x. x : A Int B ==> x : B Int A
2. B Int A <= A Int B
val it = () : unit
> br IntI 1;
Level 3 (3 subgoals)
A Int B = B Int A
1. !!x. x : A Int B ==> x : B
2. !!x. x : A Int B ==> x : A
3. B Int A <= A Int B
val it = () : unit
> br IntD2 1;
```

```
Level 4 (3 subgoals)
A Int B = B Int A
1. !!x. x : A Int B ==> x : ?A3(x) Int B
2. !!x. x : A Int B ==> x : A
3. B Int A <= A Int B
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
A Int B = B Int A
1. !!x. x : A Int B ==> x : A
2. B Int A <= A Int B
val it = () : unit
> br IntD1 1;
Level 6 (2 subgoals)
A Int B = B Int A
1. !!x. x : A Int B ==> x : A Int ?B4(x)
2. B Int A <= A Int B
val it = () : unit
> ba 1;
Level 7 (1 subgoal)
A Int B = B Int A
1. B Int A <= A Int B
val it = () : unit
> br subsetI 1;
Level 8 (1 subgoal)
A Int B = B Int A
1. !!x. x : B Int A ==> x : A Int B
val it = () : unit
> br IntI 1;
Level 9 (2 subgoals)
A Int B = B Int A
1. !!x. x : B Int A ==> x : A
2. !!x. x : B Int A ==> x : B
val it = () : unit
> br IntD2 1;
Level 10 (2 subgoals)
A Int B = B Int A
1. !!x. x : B Int A ==> x : ?A7(x) Int A
2. !!x. x : B Int A ==> x : B
val it = () : unit
> ba 1;
Level 11 (1 subgoal)
A Int B = B Int A
1. !!x. x : B Int A ==> x : B
val it = () : unit
> br IntD1 1;
Level 12 (1 subgoal)
```

```
A Int B = B Int A
1. !!x. x : B Int A ==> x : B Int ?B8(x)
val it = () : unit
> ba 1;
Level 13
A Int B = B Int A
No subgoals!
val it = () : unit
```

As provas das restantes propriedades da intersecção são semelhante às anteriores e podem ser vistas na subsecção A.2.2 do apêndice. As provas das propriedades distributivas referidas na secção 1.5 podem ser consultadas na subsecção A.2.3 do apêndice.

2.3.3 Complementares

A diferença entre dois conjuntos A e B é denotada em ZF por A - B e é definida por

$$\text{Diff\_def} \quad \text{A - B} == \{\text{x:A . x}\tilde{}\text{:B}\}$$

O complementar de um conjunto é neste caso representado utilizando a diferença. Assim, se $A \subseteq E$, então $A' = E - A$.

Em ZF tem-se as seguintes regras para a diferença entre dois conjuntos.

Designação	**Regra**
DiffI	[| c : A; c ~: B |] ==> c : A - B
DiffD1	c : A - B ==> c : A
DiffD2	c : A - B ==> c ~: B
DiffE	[| c : A - B; [| c : A; c ~: B |] ==> P |] ==> P

Em seguida, apresenta-se a prova em *Isabelle* ZF de algumas das propriedades referidas na secção 1.6.

1. $\vdash_{ZF} (A')' = A$

```
> Goal "A<=E ==> (E-(E-A)) = A";
Level 0 (1 subgoal)
A <= E ==> E - (E - A) = A
1. A <= E ==> E - (E - A) = A
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A <= E ==> E - (E - A) = A
1. A <= E ==> E - (E - A) <= A
2. A <= E ==> A <= E - (E - A)
```

```
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : E - (E - A) |] ==> x : A
2. A <= E ==> A <= E - (E - A)
val it = () : unit
> be DiffE 1;
Level 3 (2 subgoals)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : E; x ~: E - A |] ==> x : A
2. A <= E ==> A <= E - (E - A)
val it = () : unit
> be swap 1;
Level 4 (2 subgoals)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : E; x ~: A |] ==> x : E - A
2. A <= E ==> A <= E - (E - A)
val it = () : unit
> br DiffI 1;
Level 5 (3 subgoals)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : E; x ~: A |] ==> x : E
2. !!x. [| A <= E; x : E; x ~: A |] ==> x ~: A
3. A <= E ==> A <= E - (E - A)
val it = () : unit
> ba 1;
Level 6 (2 subgoals)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : E; x ~: A |] ==> x ~: A
2. A <= E ==> A <= E - (E - A)
val it = () : unit
> ba 1;
Level 7 (1 subgoal)
A <= E ==> E - (E - A) = A
1. A <= E ==> A <= E - (E - A)
val it = () : unit
> br subsetI 1;
Level 8 (1 subgoal)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : A |] ==> x : E - (E - A)
val it = () : unit
> br DiffI 1;
Level 9 (2 subgoals)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : A |] ==> x : E
2. !!x. [| A <= E; x : A |] ==> x ~: E - A
```

```
val it = () : unit
> be subsetD 1;
Level 10 (2 subgoals)
A <= E ==> E - (E - A) = A
1. !!x. x : A ==> x : A
2. !!x. [| A <= E; x : A |] ==> x ~: E - A
val it = () : unit
> ba 1;
Level 11 (1 subgoal)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : A |] ==> x ~: E - A
val it = () : unit
> br notI 1;
Level 12 (1 subgoal)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : A; x : E - A |] ==> False
val it = () : unit
> be DiffE 1;
Level 13 (1 subgoal)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : A; x : E; x ~: A |] ==> False
val it = () : unit
> be notE 1;
Level 14 (1 subgoal)
A <= E ==> E - (E - A) = A
1. !!x. [| A <= E; x : A; x : E |] ==> x : A
val it = () : unit
> ba 1;
Level 15
A <= E ==> E - (E - A) = A
No subgoals!
val it = () : unit
```

Por vezes, torna-se necessário modificar a ordem pela qual as hipóteses são apresentadas. Para tal usa-se a táctica *rotate_tac n i* que roda as hipóteses do subobjectivo *i*, *n* posições: da direita para a esquerda, se *n* for positivo e da esquerda para a direita, se *n* for negativo. Em substituição, pode-se usar o comando *back()* as vezes que forem necessárias para as hipóteses fiquem pela ordem desejada. Na prova da propriedade seguinte ilustra-se a aplicação de *rotate_tac*.

6. $\vdash_{ZF} A \subseteq B \Leftrightarrow B' \subseteq A'$

```
> Goal "[|A<=E;B<=E|]==>(A<=B)<->(E-B)<=E-A ";
Level 0 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. [| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
val it = [] : Thm.thm list
```

```
> br iffI 1;
Level 1 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. [| A <= E; B <= E; A <= B |] ==> E - B <= E - A
2. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; A <= B; x : E - B |] ==> x : E - A
2. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> br DiffI 1;
Level 3 (3 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; A <= B; x : E - B |] ==> x : E
2. !!x. [| A <= E; B <= E; A <= B; x : E - B |] ==> x ~: A
3. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> be DiffE 1;
Level 4 (3 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; A <= B; x : E; x ~: B |] ==> x : E
2. !!x. [| A <= E; B <= E; A <= B; x : E - B |] ==> x ~: A
3. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; A <= B; x : E - B |] ==> x ~: A
2. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> br notI 1;
Level 6 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; A <= B; x : E - B; x : A |] ==> False
2. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> be DiffE 1;
Level 7 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; A <= B; x : A; x : E; x ~: B |] ==> False
2. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> be notE 1;
Level 8 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
```

```
1. !!x. [| A <= E; B <= E; A <= B; x : A; x : E |] ==> x : B
2. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> be subsetD 1;
Level 9 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; x : A; x : E |] ==> x : A
2. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> ba 1;
Level 10 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. [| A <= E; B <= E; E - B <= E - A |] ==> A <= B
val it = () : unit
> br subsetI 1;
Level 11 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; E - B <= E - A; x : A |] ==> x : B
val it = () : unit
> br classical 1;
Level 12 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; E - B <= E - A; x : A; x ~: B |] ==> x : B
val it = () : unit
> br FalseE 1;
Level 13 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; E - B <= E - A; x : A; x ~: B |] ==> False
val it = () : unit
> br notE 1;
Level 14 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; E - B <= E - A; x : A; x ~: B |] ==> ~?P11(x)
2. !!x. [| A <= E; B <= E; E - B <= E - A; x : A; x ~: B |] ==> ?P11(x)
val it = () : unit
```

Como se pretende que no subobjectivo 2 ?P11(x) seja x : A, então tem-se que colocar esta hipótese em primeiro lugar, ou seja, tem-se de fazê-la rodar da direita para a esquerda 3 posições. Isto porque quando se aplica a táctica de unificação com hipóteses esta compara sintacticamente a conclusão do subobjectivo com as hipóteses. Neste caso qualquer uma das hipóteses é sintacticamente igual a ?P11(x) e logo qualquer uma delas unificaria com a conclusão.

```
> by (rotate_tac 3 2);
Level 15 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; E - B <= E - A; x : A; x ~: B |] ==> ~?P11(x)
```

```
2. !!x. [| x : A; x ~: B; A <= E; B <= E; E - B <= E - A |] ==> ?P11(x)
val it = () : unit
```

Agora se se aplicar a táctica de unificação com hipóteses obtém-se o desejado.

```
> ba 2;
Level 16 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; E - B <= E - A; x : A; x ~: B |] ==> x ~: A
val it = () : unit
> br DiffD2 1;
Level 17 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; E - B <= E - A; x : A; x ~: B |]
==> x : ?A12(x) - A
val it = () : unit
> be subsetD 1;
Level 18 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; x : A; x ~: B |] ==> x : E - B
val it = () : unit
> br DiffI 1;
Level 19 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; x : A; x ~: B |] ==> x : E
2. !!x. [| A <= E; B <= E; x : A; x ~: B |] ==> x ~: B
val it = () : unit
> be subsetD 1;
Level 20 (2 subgoals)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| B <= E; x : A; x ~: B |] ==> x : A
2. !!x. [| A <= E; B <= E; x : A; x ~: B |] ==> x ~: B
val it = () : unit
> ba 1;
Level 21 (1 subgoal)
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
1. !!x. [| A <= E; B <= E; x : A; x ~: B |] ==> x ~: B
val it = () : unit
> ba 1;
Level 22
[| A <= E; B <= E |] ==> A <= B <-> E - B <= E - A
No subgoals!
val it = () : unit
```

As provas da restantes propriedades são semelhante às anteriores e podem ser vistas na subsecção A.2.4 do apêndice.

As próximas provas são as provas das propriedades da soma de conjuntos apresentadas na secção 1.6. Relembre-se que a soma de dois conjuntos A e B era definida do seguinte modo:

$$A + B = (A - B) \cup (B - A)\,.$$

1- $\vdash_{ZF} A + B = B + A$

```
> Goal "((A-B)Un(B-A))=((B-A)Un(A-B))";
Level 0 (1 subgoal)
A - B Un (B - A) = B - A Un (A - B)
1. A - B Un (B - A) = B - A Un (A - B)
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A - B Un (B - A) = B - A Un (A - B)
1. A - B Un (B - A) <= B - A Un (A - B)
2. B - A Un (A - B) <= A - B Un (B - A)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : A - B Un (B - A) ==> x : B - A Un (A - B)
2. B - A Un (A - B) <= A - B Un (B - A)
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : A - B Un (B - A) ==> x : B - A Un (A - B)
2. !!x. x : B - A Un (A - B) ==> x : A - B Un (B - A)
val it = () : unit
> be UnE 1;
Level 4 (3 subgoals)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : A - B ==> x : B - A Un (A - B)
2. !!x. x : B - A ==> x : B - A Un (A - B)
3. !!x. x : B - A Un (A - B) ==> x : A - B Un (B - A)
val it = () : unit
> br UnI2 1;
Level 5 (3 subgoals)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : A - B ==> x : A - B
2. !!x. x : B - A ==> x : B - A Un (A - B)
3. !!x. x : B - A Un (A - B) ==> x : A - B Un (B - A)
val it = () : unit
> ba 1;
Level 6 (2 subgoals)
A - B Un (B - A) = B - A Un (A - B)
```

```
1. !!x. x : B - A ==> x : B - A Un (A - B)
2. !!x. x : B - A Un (A - B) ==> x : A - B Un (B - A)
val it = () : unit
> br UnI1 1;
Level 7 (2 subgoals)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : B - A ==> x : B - A
2. !!x. x : B - A Un (A - B) ==> x : A - B Un (B - A)
val it = () : unit
> ba 1;
Level 8 (1 subgoal)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : B - A Un (A - B) ==> x : A - B Un (B - A)
val it = () : unit
> be UnE 1;
Level 9 (2 subgoals)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : B - A ==> x : A - B Un (B - A)
2. !!x. x : A - B ==> x : A - B Un (B - A)
val it = () : unit
> br UnI2 1;
Level 10 (2 subgoals)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : B - A ==> x : B - A
2. !!x. x : A - B ==> x : A - B Un (B - A)
val it = () : unit
> ba 1;
Level 11 (1 subgoal)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : A - B ==> x : A - B Un (B - A)
val it = () : unit
> br UnI1 1;
Level 12 (1 subgoal)
A - B Un (B - A) = B - A Un (A - B)
1. !!x. x : A - B ==> x : A - B
val it = () : unit
> ba 1;
Level 13
A - B Un (B - A) = B - A Un (A - B)
No subgoals!
val it = () : unit
```

2- $\vdash_{ZF} A + (B + C) = (A + B) + C$

Para provar esta propriedade torna-se mais fácil, em primeiro lugar, provar os dois lemas seguintes.

```
Lema 1:
> val [m1,m2,m3]= Goal "[|x ~:A - B; x~:A==>R; x:B==>R|]==>R";
Level 0 (1 subgoal)
R
1. R
val m1 = "x ~: A - B" [.] : Thm.thm
val m2 = "x ~: A ==> R" [.] : Thm.thm
val m3 = "x : B ==> R" [.] : Thm.thm
> br classical 1;
Level 1 (1 subgoal)
R
1. ~R ==> R
val it = () : unit
> br m3 1;
Level 2 (1 subgoal)
R
1. ~R ==> x : B
val it = () : unit
> br classical 1;
Level 3 (1 subgoal)
R
1. [| ~R; x ~: B |] ==> x : B
val it = () : unit
> br FalseE 1;
Level 4 (1 subgoal)
R
1. [| ~R; x ~: B |] ==> False
val it = () : unit
> br notE 1;
Level 5 (2 subgoals)
R
1. [| ~R; x ~: B |] ==> ~?P3
2. [| ~R; x ~: B |] ==> ?P3
val it = () : unit
> ba 1;
Level 6 (1 subgoal)
R
1. [| ~R; x ~: B |] ==> R
val it = () : unit
> br m2 1;
Level 7 (1 subgoal)
R
```

```
1. [| ~R; x ~: B |] ==> x ~: A
val it = () : unit
> br notI 1;
Level 8 (1 subgoal)
R
1. [| ~R; x ~: B; x : A |] ==> False
val it = () : unit
> br notE 1;
Level 9 (2 subgoals)
R
1. [| ~R; x ~: B; x : A |] ==> ~?P5
2. [| ~R; x ~: B; x : A |] ==> ?P5
val it = () : unit
> br mt 1;
Level 10 (1 subgoal)
R
1. [| ~R; x ~: B; x : A |] ==> x : A - B
val it = () : unit
> br DiffI 1;
Level 11 (2 subgoals)
R
1. [| ~R; x ~: B; x : A |] ==> x : A
2. [| ~R; x ~: B; x : A |] ==> x ~: B
val it = () : unit
> ba 1;
Level 12 (1 subgoal)
R
1. [| ~R; x ~: B; x : A |] ==> x ~: B
val it = () : unit
> ba 1;
Level 13
R
No subgoals!
val it = () : unit
```

Para que este lema fique disponível para a prova da propriedade anterior ou outra, basta digitar o comando *qed* com o nome desejado para o lema, entre aspas. Neste caso,

```
> qed "nDiffE";
val nDiffE = "[| ?x ~: ?A - ?B; ?x ~: ?A ==> ?R; ?x : ?B ==> ?R |] ==>
?R"
: Thm.thm
val it = () : unit
```

Lema 2:

```
> val [m1,m2] = Goal "[|x ~:A Un B; [| x~:A ; x~:B |] ==> R|] ==>R";
Level 0 (1 subgoal)
R
1. R
val m1 = "x ~: A Un B" [.] : Thm.thm
val m2 = "[| x ~: A; x ~: B |] ==> R" [.] : Thm.thm
> br m2 1;
Level 1 (2 subgoals)
R
1. x ~: A
2. x ~: B
val it = () : unit
> br notI 1;
Level 2 (2 subgoals)
R
1. x : A ==> False
2. x ~: B
val it = () : unit
> br notI 2;
Level 3 (2 subgoals)
R
1. x : A ==> False
2. x : B ==> False
val it = () : unit
> br notE 1;
Level 4 (3 subgoals)
R
1. x : A ==> ~?P2
2. x : A ==> ?P2
3. x : B ==> False
val it = () : unit
> br m1 1;
Level 5 (2 subgoals)
R
1. x : A ==> x : A Un B
2. x : B ==> False
val it = () : unit
> br notE 2;
Level 6 (3 subgoals)
R
1. x : A ==> x : A Un B
2. x : B ==> ~?P3
3. x : B ==> ?P3
val it = () : unit
> br m1 2;
```

```
Level 7 (2 subgoals)
R
1. x : A ==> x : A Un B
2. x : B ==> x : A Un B
val it = () : unit
> br UnI1 1;
Level 8 (2 subgoals)
R
1. x : A ==> x : A
2. x : B ==> x : A Un B
val it = () : unit
> br UnI2 2;
Level 9 (2 subgoals)
R
1. x : A ==> x : A
2. x : B ==> x : B
val it = () : unit
> ba 1;
Level 10 (1 subgoal)
R
1. x : B ==> x : B
val it = () : unit
> ba 1;
Level 11
R
No subgoals!
val it = () : unit
> qed "nUnE";
val nUnE = "[| ?x ~: ?A Un ?B; [| ?x ~: ?A; ?x ~: ?B |] ==> ?R |] ==> ?R"
: Thm.thm
val it = () : unit
```

Contudo, apesar de se poder utilizar os lemas anteriores para provar a propriedade referida, esta é maçadora pois tem muitos níveis. Para superar este problema pode-se construir uma táctica que permita fazer demonstração automática em vez de demonstração assistida. Esta táctica será constituida por tacticais, pelas tácticas anteriormente referidas e pelos lemas nDiffE e nUnE.

Um *tactical* é um operador que permite combinar tácticas. Os tacticais utilizados neste caso são THEN, ORELSE , REPEAT e REPEAT_FIRST. Estes usam-se na forma *tac1* THEN *tac2*, *tac1* ORELSE *tac2*, REPEAT *tac* e REPEAT_FIRST *tacf*.

tac1 THEN *tac2* é a composição sequencial de duas tácticas. Em primeiro lugar, esta composição aplica a táctica *tac1* ao subobjectivo desejado, obtendo a sequência dos próximos subobjectivos. Depois aplica *tac2* e concretiza o resultado.

tac1 ORELSE *tac2* é a composição alternativa de duas tácticas. Esta composição aplica a táctica *tac1* ao subobjectivo desejado e apresenta o resultado, se

esta não falhar. No entanto, se *tac1* falhar, então *tac2* é aplicada. Esta é uma escolha determinística: se *tac1* não falhar, então *tac2* não é aplicada.

REPEAT *tac* aplica repetidamente a táctica *tac* ao subobjectivo desejado, até a táctica falhar.

REPEAT_FIRST *tacf* aplica a função táctica *tacf*, uma ou mais vezes, a cada subobjectivo, por ordem decrescente.

Para que a nova táctica fique disponível para provar a propriedade 2 e outras, terá de ser guardada num ficheiro ML, em conjunto com as tácticas que permitem a prova dos dois lemas anteriores. Neste caso, o ficheiro foi designado ZF.ML e o seu conteúdo é o seguinte.

```
val [m1,m2,m3]= Goal "[|x ~:A - B; x~:A==>R; x:B==>R|]==>R";
br classical 1;
br m3 1;
br classical 1;
br FalseE 1;
br notE 1;
ba 1;
br m2 1;
br notI 1;
br notE 1;
br m1 1;
br DiffI 1;
ba 1;
ba 1;
qed "nDiffE";
val [m1,m2] = Goal "[|x ~:A Un B; [| x~:A ; x~:B |] ==> R|] ==>R";
br m2 1;
br notI 1;
br notI 2;
br notE 1;
br m1 1;
br notE 2;
br m1 2;
br UnI1 1;
br UnI2 2;
ba 1;
ba 1;
qed "nUnE";
```

```
val ftact1 = fn i => ((resolve_tac [notI,iffI,subsetI,equalityI,DiffI,empty_subsetI,IntI]
i) ORELSE (eresolve_tac [equalityE,emptyE,UnionE,UnE,DiffE,IntE,subsetD,subset_trans,
subset_refl,nUnE,nDiffE] i));
val ftact2 = fn i => (((rtac UnI1 i) THEN (atac i) ) ORELSE ((rtac UnI2
i) THEN (atac i)) ORELSE (REPEAT (rtac UnI1 i) THEN (atac i)) ORELSE
(REPEAT (rtac UnI2 i) THEN (atac i)) ORELSE (((rtac UnI1 i) THEN (rtac UnI2
i))THEN (atac i)) ORELSE (((rtac UnI2 i) THEN (rtac UnI1 i))THEN (atac i))
ORELSE (((rtac UnI1 i) THEN (rtac DiffI i))THEN (atac i)) ORELSE (((rtac UnI2
i) THEN (rtac DiffI i)) THEN (atac i)) ORELSE ((rtac UnI2 i) THEN (rtac DiffI
i) THEN (rtac UnI2 i) THEN (rtac DiffI i) THEN (atac i)) ORELSE ((rtac UnI2
i) THEN (rtac DiffI i) THEN (rtac UnI1 i) THEN (rtac DiffI i) THEN (atac i))
ORELSE ((rtac UnI1 i) THEN (rtac DiffI i) THEN (rtac UnI1 i) THEN (rtac DiffI
i) THEN (atac i)) ORELSE ((rtac UnI1 i) THEN (rtac DiffI i) THEN (rtac UnI2 i)
THEN (rtac DiffI i) THEN (atac i)) );
val fax1 = fn i => (((atac i) ORELSE (etac FalseE i ) ORELSE ((etac notE i )
THEN (atac i )) ORELSE (etac emptyE i)));
val ftact3 = fn i => ((fax1 i) ORELSE (ftact1 i ) ORELSE (ftact2 i ) );
val tactfinal = (REPEAT_FIRST (ftact3));
```

A táctica foi denominada *tactfinal* e é composta por tácticas e tacticais. A ideia principal desta táctica é aplicar em primeiro lugar as tácticas mais simples, deixando para o final as tácticas cuja a aplicação é mais complicada.

O primeiro passo é verificar se é possível aplicar a táctica de unificação com as hipóteses, ou seja, verificar se existe algum subobjectivo que seja possível resolver por hipótese. Se isto não acontecer a táctica verifica se é possível aplicar as tácticas cuja unificação com hipóteses é imediata, ou seja, não dependem da aplicação de outras regras (FalseE, notE, emptyE). Isto é conseguido com a táctica *fax1*.

De seguida a táctica verifica se é possível aplicar alguma das regras de introdução. Quando já não for possível aplicar nenhuma destas tácticas aplica as tácticas de eliminação. A táctica *ftact1* é responsável por este processo.

As regras de introdução da união (UnI1 e UnI2) têm de ser aplicadas de forma diferente, uma vez que a aplicação destas depende daquilo que se tem por hipótese e daquilo que se quer provar. Então, em primeiro lugar, é necessário verificar se é possível aplicar alguma destas regras, sendo aplicada de seguida a táctica atac. Contudo, se a unificação com as hipóteses não for possível, então talvez seja necessário aplicar as regras de introdução da união mais do que uma vez (REPEAT (rtac UnI1 i) ou REPEAT (rtac UnI2 i)), seguidas de atac. No entanto também este processo poderá falhar. Assim, é necessário combinar as duas regras sequencialmente, aplicando de seguida atac. Poderá ainda ser necessário combinar estas regras com DiffI. Todo este processo é conseguido através da aplicação da táctica ftact2.

As três tácticas descritas anteriormente são aplicadas utilizando a táctica ftact3.

Finalmente, juntou-se todo este processo numa só táctica, que aplica este processo a todos os subojectivos i. Esta é a táctica designada por tacfinal.

Para que o ficheiro ZF.ML seja executado usa-se o comando *use "ZF.ML"*. Após a execução deste ficheiro a táctica *tactfinal* fica disponível. Assim, a prova da propriedade 2 resume-se ao seguinte:

```
> Goal "((A-((B-C)Un(C-B)))Un(((B-C)Un(C-B))-A))=((((A-B)Un(B-A))-C)Un(C-
((A-B)Un(B-A))))";
Level 0 (1 subgoal)
A - (B - C Un (C - B)) Un (B - C Un (C - B) - A) =
A - B Un (B - A) - C Un (C - (A - B Un (B - A)))
1. A - (B - C Un (C - B)) Un (B - C Un (C - B) - A) =
A - B Un (B - A) - C Un (C - (A - B Un (B - A)))
val it = [] : Thm.thm list
> by tactfinal;
Level 1
A - (B - C Un (C - B)) Un (B - C Un (C - B) - A) =
A - B Un (B - A) - C Un (C - (A - B Un (B - A)))
No subgoals!
val it = () : unit
```

3. $\vdash_{ZF} A + \varnothing = A$

```
> Goal "((A - 0)Un(0 - A))=A";
Level 0 (1 subgoal)
A - 0 Un (0 - A) = A
1. A - 0 Un (0 - A) = A
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A - 0 Un (0 - A) = A
1. A - 0 Un (0 - A) <= A
2. A <= A - 0 Un (0 - A)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A - 0 Un (0 - A) = A
1. !!x. x : A - 0 Un (0 - A) ==> x : A
2. A <= A - 0 Un (0 - A)
val it = () : unit
> be UnE 1;
Level 3 (3 subgoals)
A - 0 Un (0 - A) = A
1. !!x. x : A - 0 ==> x : A
2. !!x. x : 0 - A ==> x : A
3. A <= A - 0 Un (0 - A)
val it = () : unit
> be DiffD1 1;
Level 4 (2 subgoals)
A - 0 Un (0 - A) = A
1. !!x. x : 0 - A ==> x : A
2. A <= A - 0 Un (0 - A)
val it = () : unit
```

```
> br DiffD1 1;
Level 5 (2 subgoals)
A - 0 Un (0 - A) = A
1. !!x. x : 0 - A ==> x : A - ?B4(x)
2. A <= A - 0 Un (0 - A)
val it = () : unit
> undo();
Level 4 (2 subgoals)
A - 0 Un (0 - A) = A
1. !!x. x : 0 - A ==> x : A
2. A <= A - 0 Un (0 - A)
val it = () : unit
> be DiffE 1;
Level 5 (2 subgoals)
A - 0 Un (0 - A) = A
1. !!x. [| x : 0; x ~: A |] ==> x : A
2. A <= A - 0 Un (0 - A)
val it = () : unit
> be emptyE 1;
Level 6 (1 subgoal)
A - 0 Un (0 - A) = A
1. A <= A - 0 Un (0 - A)
val it = () : unit
> br subsetI 1;
Level 7 (1 subgoal)
A - 0 Un (0 - A) = A
1. !!x. x : A ==> x : A - 0 Un (0 - A)
val it = () : unit
> br UnI1 1;
Level 8 (1 subgoal)
A - 0 Un (0 - A) = A
1. !!x. x : A ==> x : A - 0
val it = () : unit
> br DiffI 1;
Level 9 (2 subgoals)
A - 0 Un (0 - A) = A
1. !!x. x : A ==> x : A
2. !!x. x : A ==> x ~: 0
val it = () : unit
> ba 1;
Level 10 (1 subgoal)
A - 0 Un (0 - A) = A
1. !!x. x : A ==> x ~: 0
val it = () : unit
> br notI 1;
Level 11 (1 subgoal)
A - 0 Un (0 - A) = A
```

```
1. !!x. || x : A; x : 0 |] ==> False
val it = () : unit
> be emptyE 1;
Level 12
A - 0 Un (0 - A) = A
No subgoals!
val it = () : unit
```

Ou utilizando a táctica anterior.

```
> Goal "((A - 0)Un(0 - A))=A";
Level 0 (1 subgoal)
A - 0 Un (0 - A) = A
1. A - 0 Un (0 - A) = A
val it = [] : Thm.thm list
> by tactfinal;
Level 1
A - 0 Un (0 - A) = A
No subgoals!
val it = () : unit
```

4. $\vdash_{ZF} A + A = \varnothing$

```
> Goal "A - A = 0";
Level 0 (1 subgoal)
A - A = 0
1. A - A = 0
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A - A = 0
1. A - A <= 0
2. 0 <= A - A
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A - A = 0
1. !!x. x : A - A ==> x : 0
2. 0 <= A - A
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
A - A = 0
1. !!x. x : A - A ==> x : 0
2. !!x. x : 0 ==> x : A - A
val it = () : unit
> be DiffE 1;
Level 4 (2 subgoals)
```

```
A - A = 0
1. !!x. [| x : A; x ~: A |] ==> x : 0
2. !!x. x : 0 ==> x : A - A
val it = () : unit
> be notE 1;
Level 5 (2 subgoals)
A - A = 0
1. !!x. x : A ==> x : A
2. !!x. x : 0 ==> x : A - A
val it = () : unit
> ba 1;
Level 6 (1 subgoal)
A - A = 0
1. !!x. x : 0 ==> x : A - A
val it = () : unit
> be emptyE 1;
Level 7
A - A = 0
No subgoals!
val it = () : unit
```

Ou

```
> Goal "A - A = 0";
Level 0 (1 subgoal)
A - A = 0
1. A - A = 0
val it = [] : Thm.thm list
> by tactfinal;
Level 1
A - A = 0
No subgoals!
val it = () : unit
```

2.3.4 Conjunto das partes

O conjunto das partes de um conjunto A é denotado em ZF por Pow(A).

O axioma da potência em ZF é designado por Pow_iff e é codificado do seguinte modo:

```
Pow_iff   A : Pow(B) <-> A <= B
```

As regras para o conjunto das partes em ZF são as seguintes.

Designação	Regra
PowI	A <= B ==> A : Pow(B)
PowD	A : Pow(B) ==> A <= B

A prova em ZF apresentada a seguir é a prova de uma das propriedades do conjunto das partes referidas na secção 1.7. As provas das restantes podem ser vistas na subsecção A.2.5 do apêndice.

1. $E \subseteq F \vdash_{ZF} \mathcal{P}(E) \subseteq \mathcal{P}(F)$

```
> Goal "(E<=F)==>(Pow(E)<=Pow(F))";
Level 0 (1 subgoal)
E <= F ==> Pow(E) <= Pow(F)
1. E <= F ==> Pow(E) <= Pow(F)
val it = [] : Thm.thm list
> br subsetI 1;
Level 1 (1 subgoal)
E <= F ==> Pow(E) <= Pow(F)
1. !!x. [| E <= F; x : Pow(E) |] ==> x : Pow(F)
val it = () : unit
> br PowI 1;
Level 2 (1 subgoal)
E <= F ==> Pow(E) <= Pow(F)
1. !!x. [| E <= F; x : Pow(E) |] ==> x <= F
val it = () : unit
> br subset_trans 1;
Level 3 (2 subgoals)
E <= F ==> Pow(E) <= Pow(F)
1. !!x. [| E <= F; x : Pow(E) |] ==> x <= ?B2(x)
2. !!x. [| E <= F; x : Pow(E) |] ==> ?B2(x) <= F
val it = () : unit
> ba 2;
Level 4 (1 subgoal)
E <= F ==> Pow(E) <= Pow(F)
1. !!x. [| E <= F; x : Pow(E) |] ==> x <= E
val it = () : unit
> br PowD 1;
Level 5 (1 subgoal)
E <= F ==> Pow(E) <= Pow(F)
1. !!x. [| E <= F; x : Pow(E) |] ==> x : Pow(E)
val it = () : unit
> ba 1;
Level 6
E <= F ==> Pow(E) <= Pow(F)
No subgoals!
val it = () : unit
```

2.3.5 Pares ordenados

Um par ordenado com primeira coordenada a e segunda coordenada b representa-se em *Isabelle* ZF por <a,b>. A sua definição é designada por *pair_def* e é a seguinte:

$$\text{<a,b>} == \{\{a,a\}, \{a,b\}\}.$$

O produto cartesiano de A e B é denotado por *Sigma(A,B)* e a sua definição por *Sigma_def*. Esta definição é representada por

$$\text{Sigma(A,B)} == \text{UN x:A. UN y:B(x). } \{\text{<x,y>}\}$$

As regras para pares ordenados e produto cartesiano são as seguintes:

Designação	Regra
Pair_inject1	<a, b> = <c, d> ==> a = c
Pair_inject2	<a, b> = <c, d> ==> b = d
Pair_inject	[| <a, b> = <c, d>; [| a = c; b = d |] ==> R |] ==> R
Pair_neq_0	<a, b> = 0 ==> R
SigmaI	[| a : A; b : B(a) |] ==> <a, b> : Sigma(A, B)
SigmaE	[| c : Sigma(A, B);!!x y. [| x : A; y : B(x); c = <x, y> |] ==> P |] ==> P
SigmaE2	[| <a, b> : Sigma(A, B); [| a : A; b : B(a) |] ==> P |] ==> P

De seguida apresenta-se a prova de algumas das propriedades apresentadas na secção 1.8.

Sejam A, B, X e Y conjuntos.

1. $\vdash_{ZF} (A \cup B) \times X = (A \times X) \cup (B \times X)$

Para se efectuar a prova desta propriedade é necessário em primeiro lugar provar o lema [|x=<a,b>;<a,b>:A|]==>x:A, que no final designar-se-á por *lema1*.

```
> Goal "[|x=<a,b>;<a,b>:A|]==>x:A";
Level 0 (1 subgoal)
[| x = <a, b>; <a, b> : A |] ==> x : A
1. [| x = <a, b>; <a, b> : A |] ==> x : A
val it = [] : Thm.thm list
> br ssubst 1;
Level 1 (2 subgoals)
[| x = <a, b>; <a, b> : A |] ==> x : A
1. [| x = <a, b>; <a, b> : A |] ==> A = ?a
2. [| x = <a, b>; <a, b> : A |] ==> x : ?a
val it = () : unit
> back();
Level 1 (2 subgoals)
```

```
[| x = <a, b>; <a, b> : A |] ==> x : A
1. [| x = <a, b>; <a, b> : A |] ==> x = ?a
2. [| x = <a, b>; <a, b> : A |] ==> ?a : A
val it = () : unit
> ba 1;
Level 2 (1 subgoal)
[| x = <a, b>; <a, b> : A |] ==> x : A
1. [| x = <a, b>; <a, b> : A |] ==> <a, b> : A
val it = () : unit
> ba 1;
Level 3
[| x = <a, b>; <a, b> : A |] ==> x : A
No subgoals!
> qed "lema1";
val lema1 = "[| ?x = <?a, ?b>; <?a, ?b> : ?A |] ==> ?x : ?A" : Thm.thm
val it = () : unit
```

Agora está-se em condições de provar a propriedade desejada.

```
> Goal "(A Un B) * X = (A * X) Un (B * X)";
Level 0 (1 subgoal)
(A Un B) * X = A * X Un B * X
1. (A Un B) * X = A * X Un B * X
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. (A Un B) * X <= A * X Un B * X
2. A * X Un B * X <= (A Un B) * X
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x. x : (A Un B) * X ==> x : A * X Un B * X
2. A * X Un B * X <= (A Un B) * X
val it = () : unit
> be SigmaE 1;
Level 3 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : A Un B; y : X; x = <xa, y> |] ==> x : A * X Un B * X
2. A * X Un B * X <= (A Un B) * X
val it = () : unit
> be UnE 1;
Level 4 (3 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; x = <xa, y>; xa : A |] ==> x : A * X Un B * X
2. !!x xa y. [| y : X; x = <xa, y>; xa : B |] ==> x : A * X Un B * X
```

```
3. A * X Un B * X <= (A Un B) * X
val it = () : unit
> br UnI1 1;
Level 5 (3 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; x = <xa, y>; xa : A |] ==> x : A * X
2. !!x xa y. [| y : X; x = <xa, y>; xa : B |] ==> x : A * X Un B * X
3. A * X Un B * X <= (A Un B) * X
val it = () : unit
> br UnI2 2;
Level 6 (3 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; x = <xa, y>; xa : A |] ==> x : A * X
2. !!x xa y. [| y : X; x = <xa, y>; xa : B |] ==> x : B * X
3. A * X Un B * X <= (A Un B) * X
val it = () : unit
> be lema1 1;
Level 7 (3 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; xa : A |] ==> <xa, y> : A * X
2. !!x xa y. [| y : X; x = <xa, y>; xa : B |] ==> x : B * X
3. A * X Un B * X <= (A Un B) * X
val it = () : unit
> be lema1 2;
Level 8 (3 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; xa : A |] ==> <xa, y> : A * X
2. !!x xa y. [| y : X; xa : B |] ==> <xa, y> : B * X
3. A * X Un B * X <= (A Un B) * X
val it = () : unit
> br SigmaI 1;
Level 9 (4 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; xa : A |] ==> xa : A
2. !!x xa y. [| y : X; xa : A |] ==> y : X
3. !!x xa y. [| y : X; xa : B |] ==> <xa, y> : B * X
4. A * X Un B * X <= (A Un B) * X
val it = () : unit
> ba 1;
Level 10 (3 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; xa : A |] ==> y : X
2. !!x xa y. [| y : X; xa : B |] ==> <xa, y> : B * X
3. A * X Un B * X <= (A Un B) * X
val it = () : unit
> ba 1;
Level 11 (2 subgoals)
```

```
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; xa : B |] ==> <xa, y> : B * X
2. A * X Un B * X <= (A Un B) * X
val it = () : unit
> br SigmaI 1;
Level 12 (3 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; xa : B |] ==> xa : B
2. !!x xa y. [| y : X; xa : B |] ==> y : X
3. A * X Un B * X <= (A Un B) * X
val it = () : unit
> ba 1;
Level 13 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| y : X; xa : B |] ==> y : X
2. A * X Un B * X <= (A Un B) * X
val it = () : unit
> ba 1;
Level 14 (1 subgoal)
(A Un B) * X = A * X Un B * X
1. A * X Un B * X <= (A Un B) * X
> br subsetI 1;
Level 15 (1 subgoal)
(A Un B) * X = A * X Un B * X
1. !!x. x : A * X Un B * X ==> x : (A Un B) * X
val it = () : unit
> be UnE 1;
Level 16 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x. x : A * X ==> x : (A Un B) * X
2. !!x. x : B * X ==> x : (A Un B) * X
val it = () : unit
> be SigmaE 1;
Level 17 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : A; y : X; x = <xa, y> |] ==> x : (A Un B) * X
2. !!x. x : B * X ==> x : (A Un B) * X
val it = () : unit
> be lemma1 1;
Level 18 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : A; y : X |] ==> <xa, y> : (A Un B) * X
2. !!x. x : B * X ==> x : (A Un B) * X
val it = () : unit
> br SigmaI 1;
Level 19 (3 subgoals)
(A Un B) * X = A * X Un B * X
```

```
1. !!x xa y. [| xa : A; y : X |] ==> xa : A Un B
2. !!x xa y. [| xa : A; y : X |] ==> y : X
3. !!x. x : B * X ==> x : (A Un B) * X
val it = () : unit
> ba 2;
Level 20 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : A; y : X |] ==> xa : A Un B
2. !!x. x : B * X ==> x : (A Un B) * X
val it = () : unit
> br UnI1 1;
Level 21 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : A; y : X |] ==> xa : A
2. !!x. x : B * X ==> x : (A Un B) * X
val it = () : unit
> ba 1;
Level 22 (1 subgoal)
(A Un B) * X = A * X Un B * X
1. !!x. x : B * X ==> x : (A Un B) * X
val it = () : unit
> be SigmaE 1;
Level 23 (1 subgoal)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : B; y : X; x = <xa, y> |] ==> x : (A Un B) * X
val it = () : unit
> be lema1 1;
Level 24 (1 subgoal)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : B; y : X |] ==> <xa, y> : (A Un B) * X
val it = () : unit
> br SigmaI 1;
Level 25 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : B; y : X |] ==> xa : A Un B
2. !!x xa y. [| xa : B; y : X |] ==> y : X
val it = () : unit
> br UnI2 1;
Level 26 (2 subgoals)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : B; y : X |] ==> xa : B
2. !!x xa y. [| xa : B; y : X |] ==> y : X
val it = () : unit
> ba 1;
Level 27 (1 subgoal)
(A Un B) * X = A * X Un B * X
1. !!x xa y. [| xa : B; y : X |] ==> y : X
```

```
val it = () : unit
> ba 1;
Level 28
(A Un B) * X = A * X Un B * X
No subgoals!
val it = () : unit
```

Esta propriedade pode ser provada automaticamente, se se usar a táctica *Blast_tac*, como se pode ver de seguida. A táctica Blast_tac vem com a distribuição de ZF e é baseada em tableaux. Não será descrita aqui.

```
> Goal "(A Un B) * X = (A * X) Un(B * X)";
Level 0 (1 subgoal)
(A Un B) * X = A * X Un B * X
1. (A Un B) * X = A * X Un B * X
val it = [] : Thm.thm list
> by (Blast_tac 1);
### Search depth = 0
### Search depth = 1
Level 1
(A Un B) * X = A * X Un B * X
No subgoals!
val it = () : unit
```

Na prova da propriedade seguinte é necessário utilizar o lema1 estabelecido anteriormente. Além disso, usa-se uma nova táctica de resolução que permite acrescentar uma premissa, ou várias, que depois têm de ser provadas. Esta é designada por *subgoal_tac* () no caso de se acrescentar apenas uma premissa e *subgoals_tac* quando se acrescenta mais do que uma premissa. Na prova seguinte é necessário acrescentar a permissa $<xa,y>=<xb,ya>$ ao subobjectivo 1 no "level" 21, para tal escreve-se *by (subgoal_tac "<xa,y>=<xb,ya>" 1);*.

2. $\vdash_{ZF} (A \cap B) \times (X \cap Y) = (A \times X) \cap (B \times Y)$

```
> Goal "(A Int B) * (X Int Y) = (A * X) Int (B * Y)";
Level 0 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. (A Int B) * (X Int Y) = A * X Int B * Y
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. (A Int B) * (X Int Y) <= A * X Int B * Y
2. A * X Int B * Y <= (A Int B) * (X Int Y)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
```

```
1. !!x. x : (A Int B) * (X Int Y) ==> x : A * X Int B * Y
2. A * X Int B * Y <= (A Int B) * (X Int Y)
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x. x : (A Int B) * (X Int Y) ==> x : A * X Int B * Y
2. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> br IntI 1;
Level 4 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x. x : (A Int B) * (X Int Y) ==> x : A * X
2. !!x. x : (A Int B) * (X Int Y) ==> x : B * Y
3. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be SigmaE 1;
Level 5 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A Int B; y : X Int Y; x = <xa, y> |] ==> x : A * X
2. !!x. x : (A Int B) * (X Int Y) ==> x : B * Y
3. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be lema1 1;
Level 6 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A Int B; y : X Int Y |] ==> <xa, y> : A * X
2. !!x. x : (A Int B) * (X Int Y) ==> x : B * Y
3. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be IntE 1;
Level 7 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| y : X Int Y; xa : A; xa : B |] ==> <xa, y> : A * X
2. !!x. x : (A Int B) * (X Int Y) ==> x : B * Y
3. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be IntE 1;
Level 8 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A; xa : B; y : X; y : Y |] ==> <xa, y> : A * X
2. !!x. x : (A Int B) * (X Int Y) ==> x : B * Y
3. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> br SigmaI 1;
Level 9 (4 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
```

```
1. !!x xa y. [| xa : A; xa : B; y : X; y : Y |] ==> xa : A
2. !!x xa y. [| xa : A; xa : B; y : X; y : Y |] ==> y : X
3. !!x. x : (A Int B) * (X Int Y) ==> x : B * Y
4. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> ba 1;
Level 10 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A; xa : B; y : X; y : Y |] ==> y : X
2. !!x. x : (A Int B) * (X Int Y) ==> x : B * Y
3. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> ba 1;
Level 11 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x. x : (A Int B) * (X Int Y) ==> x : B * Y
2. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be SigmaE 1;
Level 12 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A Int B; y : X Int Y; x = <xa, y> |] ==> x : B * Y
2. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be lemma1 1;
Level 13 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A Int B; y : X Int Y |] ==> <xa, y> : B * Y
2. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be IntE 1;
Level 14 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| y : X Int Y; xa : A; xa : B |] ==> <xa, y> : B * Y
2. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be IntE 1;
Level 15 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A; xa : B; y : X; y : Y |] ==> <xa, y> : B * Y
2. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> br SigmaI 1;
Level 16 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A; xa : B; y : X; y : Y |] ==> xa : B
2. !!x xa y. [| xa : A; xa : B; y : X; y : Y |] ==> y : Y
```

```
3. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> ba 1;
Level 17 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y. [| xa : A; xa : B; y : X; y : Y |] ==> y : Y
2. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> ba 1;
Level 18 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x. x : A * X Int B * Y ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be IntE 1;
Level 19 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x. [| x : A * X; x : B * Y |] ==> x : (A Int B) * (X Int Y)
val it = () : unit
> be SigmaE 1;
Level 20 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y.
[| x : B * Y; xa : A; y : X; x = <xa, y> |]
==> x : (A Int B) * (X Int Y)
val it = () : unit
> be SigmaE 1;
Level 21 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> x : (A Int B) * (X Int Y)
val it = () : unit
> by (subgoal_tac "<xa,y>=<xb,ya>" 1);
Level 22 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya>;
<xa, y> = <xb, ya> |]
==> x : (A Int B) * (X Int Y)
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> be lema1 1;
Level 23 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
```

```
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> <xa, y> : (A Int B) * (X Int Y)
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> br SigmaI 1;
Level 24 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> xa : A Int B
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
3. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> br IntI 1;
Level 25 (4 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> xa : A
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> xa : B
3. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
4. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> ba 1;
Level 26 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> xa : B
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
3. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
```

```
val it = () : unit
> be ssubst 1;
Level 27 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; <xa, y> = <xb, ya> |] ==> xa : B
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
3. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> br Pair_inject 1;
Level 28 (4 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; <xa, y> = <xb, ya> |]
==> <?a22(x, xa, y, xb, ya), ?b22(x, xa, y, xb, ya)> =
<?c22(x, xa, y, xb, ya), ?d22(x, xa, y, xb, ya)>
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; <xa, y> = <xb, ya>;
?a22(x, xa, y, xb, ya) = ?c22(x, xa, y, xb, ya);
?b22(x, xa, y, xb, ya) = ?d22(x, xa, y, xb, ya) |]
==> xa : B
3. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
4. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> ba 1;
Level 29 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; <xa, y> = <xb, ya>; xa = xb;
y = ya |]
==> xa : B
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
3. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> be ssubst 1;
```

```
Level 30 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; xa = xb; y = ya |] ==> xa : B
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
3. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> back();
Level 30 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; <xa, y> = <xb, ya>; y = ya |]
==> xb : B
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
3. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> ba 1;
Level 31 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; <xa, y> = <xb, ya> |]
==> y : X Int Y
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> be Pair_inject 1;
Level 32 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; xa = xb; y = ya |]
==> y : X Int Y
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> br IntI 1;
Level 33 (3 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
```

```
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; xa = xb; y = ya |]
==> y : X
2. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; xa = xb; y = ya |]
==> y : Y
3. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> ba 1;
Level 34 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; xa = xb; y = ya |]
==> y : Y
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> be ssubst 1;
Level 35 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; xa = xb; y = ya |] ==> y : Y
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> back();
Level 35 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; y = ya |] ==> y : Y
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> back();
Level 35 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; xb : B; ya : Y; x = <xb, ya>; xa = xb |] ==> ya : Y
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
```

```
> ba 1;
Level 36 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = <xb, ya>
val it = () : unit
> br ssubst 1;
Level 37 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xb, ya> = ?a27(x, xa, y, xb, ya)
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = ?a27(x, xa, y, xb, ya)
val it = () : unit
> br sym 1;
Level 38 (2 subgoals)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> ?a27(x, xa, y, xb, ya) = <xb, ya>
2. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = ?a27(x, xa, y, xb, ya)
val it = () : unit
> ba 1;
Level 39 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> <xa, y> = x
val it = () : unit
> br sym 1;
Level 40 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. !!x xa y xb ya.
[| xa : A; y : X; x = <xa, y>; xb : B; ya : Y; x = <xb, ya> |]
==> x = <xa, y>
val it = () : unit
> ba 1;
Level 41
(A Int B) * (X Int Y) = A * X Int B * Y
No subgoals!
val it = () : unit
```

Ou utilizando a táctica referida anteriormente:

```
> Goal "(A Int B) * (X Int Y) = (A * X) Int (B * Y)";
Level 0 (1 subgoal)
(A Int B) * (X Int Y) = A * X Int B * Y
1. (A Int B) * (X Int Y) = A * X Int B * Y
val it = [] : Thm.thm list
> by (Blast_tac 1);
### Search depth = 0
### Search depth = 1
Level 1
(A Int B) * (X Int Y) = A * X Int B * Y
No subgoals!
val it = () : unit
```

As provas das restantes propriedades apresentadas na secção 1.8 são semelhante às anteriores e podem ser vistas na subsecção A.2.6 do apêndice.

2.3.6 Números naturais e aritmética

A teoria *Nat* define os números naturais e a indução matemática. O conjunto dos números naturais (w) é designado por *nat* e a teoria *Arith* desenvolve a aritmética dos números naturais.

Relembrando $suc(x) = x \cup \{x\}$. Em *Isabelle* a definição da função sucessor (*succ_def*) é feita à custa da função *cons* cuja definição é

cons_def cons (a,A) == Upair (a,a) Un A.

Assim,

succ_def succ (i) == cons (i,i).

As regras para a função sucessor são as seguintes.

Designação	**Regra**
succI1	i : succ(i)
succI2	i : j ==> i : succ(j)
succCI	(i ~: j ==> i = j) ==> i : succ(j)
succE	[| i : succ(j); i = j ==> P; i : j ==> P |] ==> P
succ_neq_0	succ(n) = 0 ==> R
succ_inject	succ(m) = succ(n) ==> m = n
succ_inject_iff	succ(m) = succ(n) <-> m = n

Repare-se que as regras succ_neq_0 e succ_inject são respectivamente os axiomas IV e V de Peano, apresentados na secção 1.14.

De seguida, apresenta-se a prova do lema a=b ==> succ(a)=succ(b) que se denominará *subsucc* e que será utilizado, mais à frente, na prova de outros lemas e propriedades.

```
> Goal "a=b ==> succ(a)=succ(b)";
Level 0 (1 subgoal)
a = b ==> succ(a) = succ(b)
1. a = b ==> succ(a) = succ(b)
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
a = b ==> succ(a) = succ(b)
1. a = b ==> succ(a) <= succ(b)
2. a = b ==> succ(b) <= succ(a)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(a) |] ==> x : succ(b)
2. a = b ==> succ(b) <= succ(a)
val it = () : unit
> br subst 1;
Level 3 (3 subgoals)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(a) |] ==> ?a2(x) = succ(b)
2. !!x. [| a = b; x : succ(a) |] ==> x : ?a2(x)
3. a = b ==> succ(b) <= succ(a)
val it = () : unit
> back();
Level 3 (3 subgoals)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(a) |] ==> ?a2(x) = b
2. !!x. [| a = b; x : succ(a) |] ==> x : succ(?a2(x))
3. a = b ==> succ(b) <= succ(a)
val it = () : unit
> ba 1;
Level 4 (2 subgoals)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(a) |] ==> x : succ(a)
2. a = b ==> succ(b) <= succ(a)
val it = () : unit
> ba 1;
Level 5 (1 subgoal)
a = b ==> succ(a) = succ(b)
1. a = b ==> succ(b) <= succ(a)
val it = () : unit
> br subsetI 1;
Level 6 (1 subgoal)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(b) |] ==> x : succ(a)
val it = () : unit
```

```
> br subst 1;
Level 7 (2 subgoals)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(b) |] ==> ?a4(x) = succ(a)
2. !!x. [| a = b; x : succ(b) |] ==> x : ?a4(x)
val it = () : unit
> back();
Level 7 (2 subgoals)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(b) |] ==> ?a4(x) = a
2. !!x. [| a = b; x : succ(b) |] ==> x : succ(?a4(x))
val it = () : unit
> br sym 1;
Level 8 (2 subgoals)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(b) |] ==> a = ?a4(x)
2. !!x. [| a = b; x : succ(b) |] ==> x : succ(?a4(x))
val it = () : unit
> ba 1;
Level 9 (1 subgoal)
a = b ==> succ(a) = succ(b)
1. !!x. [| a = b; x : succ(b) |] ==> x : succ(b)
val it = () : unit
> ba 1;
Level 10
a = b ==> succ(a) = succ(b)
No subgoals!
val it = () : unit
> qed "subsucc";
val subsucc = "?a = ?b ==> succ(?a) = succ(?b)" : Thm.thm
val it = () : unit
```

Uma operação no conjunto dos naturais só está bem definida se os seus argumentos forem números naturais. Assim, para minimizar a necessidade de provar que um argumento é natural, construiu-se a função *natify*. Esta é definida de tal modo que

$$\textit{natify}(n) = n \text{ se } n \text{ é um número natural;}$$
$$\textit{natify}(\textit{succ}(x)) = \textit{succ}(\textit{natify}\ (x)) \text{ para todo o } x;$$
$$\text{e}$$
$$\textit{natify}(x) = 0 \text{ para os restantes casos.}$$

O benefício desta função é que a adição e multiplicação têm sempre como resultado um número natural, independentemente dos seus argumentos.

As leis algébricas (comutativa, associativa e distributiva) são incondicionais.

As regras apresentadas de seguida são as regras para a função *natify*.

Designação	**Regra**
natify_0	natify(0) = 0
natify_non_succ	ALL z. x ~= succ(z) ==> natify(x) = 0
natify_in_nat	natify(x) : nat
natify_ident	n : nat ==> natify(n) = n
natify_eqE	[| natify(x) = y; x : nat |] ==> x = y
natify_idem	natify(natify(x)) = natify(x)
add_natify1	natify(m) #+ n = m #+ n"
add_natify2	"m #+ natify(n) = m #+ n
mult_natify1	"natify(m) #* n = m #* n"
mult_natify2	m #* natify(n) = m #* n

As três regras apresentadas de seguida são os axiomas I ,II e III de Peano.

Designação	**Regra**
nat_0I	0 : nat
nat_succI	n : nat ==> succ(n) : nat
nat_induct	[| n : nat; P(0); !!x. [| x : nat; P(x) |] ==> P(succ(x)) |] ==> P(n)

Facilmente, se verifica que a regra *nat_ induct* é o *princípio de indução matemática.*

De seguida, apresenta-se as regras para a adição e multiplicação de números naturais. Algumas dessas regras (associativa e comutativa da adição e da multiplicação) serão provadas posteriormente, pois as suas provas servem para ilustrar o modo de fazer uma prova por indução em *Isabelle* ZF, bem como o modo de aplicar algumas das restantes regras.

Designação	**Regra**
add_type	m #+ n : nat
add_0_natify	0 #+ m = natify(m)
add_succ	succ(m) #+ n = succ(m #+ n)
add_0	"m : nat ==> 0 #+ m = m"
add_0_right_natify	m #+ 0 = natify(m)
add_succ_right	m #+ succ(n) = succ(m #+ n)
add_0_right	m : nat ==> m #+ 0 = m
add_left_cancel_natify	k #+ m = k #+ n ==> natify(m) = natify(n)
add_left_cancel	[| i = j; i #+ m = j #+ n; m : nat; n : nat |] ==> m = n
add_assoc	m #+ n #+ k = m #+ (n #+ k)
add_commute	m #+ n = n #+ m
add_left_commute	m #+ (n #+ k) = n #+ (m #+ k)

Designação	**Regra**
mult_type	m #* n : nat
mult_0	0 #* m = 0
mult_succ	succ(m) #* n = n #+ m #* n
mult_0_right	m #* 0 = 0"
mult_succ_right	m #* succ(n) = m #+ m #* n"
mult_1_natify	1 #* n = natify(n)
mult_1_right_natify	n #* 1 = natify(n)
mult_1	n : nat ==> 1 #* n = n
mult_1_right	n : nat ==> n #* 1 = n
mult_commute	m #* n = n #* m
add_mult_distrib	(m #+ n) #* k = m #* k #+ n #* k
add_mult_distrib_left	k #* (m #+ n) = k #* m #+ k #* n
mult_assoc	m #* n #* k = m #* (n #* k)
mult_left_commute	m #* (n #* k) = n #* (m #* k)

A propriedade a ser provada de seguida é a propriedade comutativa da adição. Contudo, antes de prová-la é necessário provar uma nova regra, que facilitará a prova da mesma. Esta nova regra é a junção das regras *add_0* e *add_0_right* e designar-se-á *commBase*.

```
> Goal "n:nat ==> 0#+n=n#+0";
Level 0 (1 subgoal)
n : nat ==> 0 #+ n = n #+ 0
1. n : nat ==> 0 #+ n = n #+ 0
val it = [] : Thm.thm list
> by (subgoals_tac ["0 #+ n=n","n=n#+0"]1);
Level 1 (3 subgoals)
n : nat ==> 0 #+ n = n #+ 0
1. [| n : nat; 0 #+ n = n; n = n #+ 0 |] ==> 0 #+ n = n #+ 0
2. [| n : nat; 0 #+ n = n |] ==> n = n #+ 0
3. n : nat ==> 0 #+ n = n
val it = () : unit
> br add_0 3;
Level 2 (3 subgoals)
n : nat ==> 0 #+ n = n #+ 0
1. [| n : nat; 0 #+ n = n; n = n #+ 0 |] ==> 0 #+ n = n #+ 0
2. [| n : nat; 0 #+ n = n |] ==> n = n #+ 0
3. n : nat ==> n : nat
val it = () : unit
> ba 3;
Level 3 (2 subgoals)
n : nat ==> 0 #+ n = n #+ 0
1. [| n : nat; 0 #+ n = n; n = n #+ 0 |] ==> 0 #+ n = n #+ 0
2. [| n : nat; 0 #+ n = n |] ==> n = n #+ 0
val it = () : unit
> br sym 2;
```

```
Level 4 (2 subgoals)
n : nat ==> 0 #+ n = n #+ 0
1. [| n : nat; 0 #+ n = n; n = n #+ 0 |] ==> 0 #+ n = n #+ 0
2. [| n : nat; 0 #+ n = n |] ==> n #+ 0 = n
val it = () : unit
> br add_0_right 2;
Level 5 (2 subgoals)
n : nat ==> 0 #+ n = n #+ 0
1. [| n : nat; 0 #+ n = n; n = n #+ 0 |] ==> 0 #+ n = n #+ 0
2. [| n : nat; 0 #+ n = n |] ==> n : nat
val it = () : unit
> ba 2;
Level 6 (1 subgoal)
n : nat ==> 0 #+ n = n #+ 0
1. [| n : nat; 0 #+ n = n; n = n #+ 0 |] ==> 0 #+ n = n #+ 0
val it = () : unit
> be trans 1;
Level 7 (1 subgoal)
n : nat ==> 0 #+ n = n #+ 0
1. [| n : nat; n = n #+ 0 |] ==> n = n #+ 0
val it = () : unit
> ba 1;
Level 8
n : nat ==> 0 #+ n = n #+ 0
No subgoals!
val it = () : unit
> qed "commBase";
val commBase = "?n : nat ==> 0 #+ ?n = ?n #+ 0" : Thm.thm
val it = () : unit
```

Agora está-se em condições de provar a propriedade comutativa. Chama-se à atenção o leitor para o facto de que esta propriedade é a regra *add_commut,* em *Isabelle* ZF.

```
> Goal "[|n:nat;m:nat|]==>n#+m=m#+n";
Level 0 (1 subgoal)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. [| n : nat; m : nat |] ==> n #+ m = m #+ n
val it = [] : Thm.thm list
> br nat_induct 1;
Level 1 (3 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. [| n : nat; m : nat |] ==> m #+ n : nat
2. [| n : nat; m : nat |] ==> n #+ m = 0
3. !!x. [| n : nat; m : nat; x : nat; n #+ m = x |] ==> n #+ m = succ(x)
val it = () : unit
> back();
```

```
Level 1 (3 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. [| n : nat; m : nat |] ==> n : nat
2. [| n : nat; m : nat |] ==> 0 #+ m = m #+ 0
3. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x |]
==> succ(x) #+ m = m #+ succ(x)
val it = () : unit
> ba 1;
Level 2 (2 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. [| n : nat; m : nat |] ==> 0 #+ m = m #+ 0
2. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x |]
==> succ(x) #+ m = m #+ succ(x)
val it = () : unit
> br commBase 1;
Level 3 (2 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. [| n : nat; m : nat |] ==> m : nat
2. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x |]
==> succ(x) #+ m = m #+ succ(x)
val it = () : unit
> ba 1;
Level 4 (1 subgoal)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x |]
==> succ(x) #+ m = m #+ succ(x)
val it = () : unit
> by (subgoals_tac ["succ(x)#+m = succ(x#+m)", "succ(x#+m) = succ(m#+x)",
"succ(m#+x) = m #+ succ(x)"] 1);
Level 5 (4 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x);
succ(m #+ x) = m #+ succ(x) |]
==> succ(x) #+ m = m #+ succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x) |]
==> succ(m #+ x) = m #+ succ(x)
3. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m) |]
==> succ(x #+ m) = succ(m #+ x)
4. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x |]
==> succ(x) #+ m = succ(x #+ m)
val it = () : unit
> br add_succ 4;
Level 6 (3 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
```

```
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x);
succ(m #+ x) = m #+ succ(x) |]
==> succ(x) #+ m = m #+ succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x) |]
==> succ(m #+ x) = m #+ succ(x)
3. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m) |]
==> succ(x #+ m) = succ(m #+ x)
val it = () : unit
> br subsucc 3;
Level 7 (3 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x);
succ(m #+ x) = m #+ succ(x) |]
==> succ(x) #+ m = m #+ succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x) |]
==> succ(m #+ x) = m #+ succ(x)
3. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m) |]
==> x #+ m = m #+ x
val it = () : unit
> ba 3;
Level 8 (2 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x);
succ(m #+ x) = m #+ succ(x) |]
==> succ(x) #+ m = m #+ succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x) |]
==> succ(m #+ x) = m #+ succ(x)
val it = () : unit
> br sym 2;
Level 9 (2 subgoals)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x);
succ(m #+ x) = m #+ succ(x) |]
==> succ(x) #+ m = m #+ succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x) |]
==> m #+ succ(x) = succ(m #+ x)
val it = () : unit
```

```
> br add_succ_right 2;
Level 10 (1 subgoal)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x) #+ m = succ(x #+ m); succ(x #+ m) = succ(m #+ x);
succ(m #+ x) = m #+ succ(x) |]
==> succ(x) #+ m = m #+ succ(x)
val it = () : unit
> be trans 1;
Level 11 (1 subgoal)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(x #+ m) = succ(m #+ x); succ(m #+ x) = m #+ succ(x) |]
==> succ(x #+ m) = m #+ succ(x)
val it = () : unit
> be trans 1;
Level 12 (1 subgoal)
[| n : nat; m : nat |] ==> n #+ m = m #+ n
1. !!x. [| n : nat; m : nat; x : nat; x #+ m = m #+ x;
succ(m #+ x) = m #+ succ(x) |]
==> succ(m #+ x) = m #+ succ(x)
val it = () : unit
> ba 1;
Level 13
[| n : nat; m : nat |] ==> n #+ m = m #+ n
No subgoals!
val it = () : unit
```

2.4 Simplificação

A simplificação consiste na reescrita de expressões complexas em expressões equivalentes mais simples.

Simp_tac i simplifica o subobjectivo i utilizando o conjunto de simplificação (em inglês "simpset"), em uso. Pode resolver o subobjectivo por completo se este se tornar trivial, depois de utilizada a táctica de resolução do conjunto de simplificação.

Asm_simp_tac i é semelhante a *Simp_tac i*, mas extraí regras de reescrita adicionais das hipóteses locais.

Full_simp_tac i é semelhante a *Simp_tac i*, mas também simplifica as hipoteses locais (sem utilizar as hipóteses locais para se simplificarem umas às outras ou o subobjectivo i).

Asm_full_simp_tac i é semelhante a *Asm_simp_tac i*, mas também simplifica as hipóteses locais. Em particular, as hipóteses locais podem se simplificar umas às outras.

set trace_simp; é o comando que permite ao utilizador visualizar as operações internas do simplificador.

O simplificador é controlado pela informação contida no conjunto de simplificação, conforme documentado no capítulo 10 de [8].
Em *Isabelle* ZF o conjunto de simplificação é designado ZF_ss e em *Isabelle* FOL, FOL_ss.

print_ss aa; é o comando que permite visualizar o contéudo do conjunto de simplificação *aa*.

Exemplo

Este exemplo é a prova da regra *add_0_right* utilizando, neste caso, a simplificação.

> Goal "n:nat ==> n#+0=n";
Level 0 (1 subgoal)
n : nat ==> n #+ 0 = n
1. n : nat ==> n #+ 0 = n
val it = [] : Thm.thm list

O primeiro passo é aplicar a regra de indução.
> br nat_induct 1;
Level 1 (3 subgoals)
n : nat ==> n #+ 0 = n
1. n : nat ==> n : nat
2. n : nat ==> 0 #+ 0 = 0
3. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit

O subobjectivo 1 é resolvido por hipótese.

> ba 1;
Level 2 (2 subgoals)
n : nat ==> n #+ 0 = n
1. n : nat ==> 0 #+ 0 = 0
2. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit

Neste momento tem-se dois subobjectivos. O primeiro é a base de indução e o segundo é o passo de indução.
Para se resolver o primeiro basta aplicar *Simp_tac*.

> by (Simp_tac 1);

```
Level 3 (1 subgoal)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
```

Para se resolver o último subobjectivo é necessário utilizar Asm_simp_tac para que as hipóteses de indução sejam usadas como regras de reescrita.

```
> by (Asm_simp_tac 1);
Level 4
n : nat ==> n #+ 0 = n
No subgoals!
val it = () : unit
```

ou

```
> Goal "n:nat ==> n#+0=n";
Level 0 (1 subgoal)
n : nat ==> n #+ 0 = n
1. n : nat ==> n #+ 0 = n
val it = [] : Thm.thm list
> by (Asm_simp_tac 1);
Level 1
n : nat ==> n #+ 0 = n
No subgoals!
val it = () : unit
```

Como foi referido, anteriormente, o comando set_trace_simp permite ao utilizador visualizar as operações internas do simplificador. Para o exemplo anterior tem-se:

```
> set trace_simp;
val it = true : bool
> Goal "n:nat ==> n#+0=n";
Level 0 (1 subgoal)
n : nat ==> n #+ 0 = n
1. n : nat ==> n #+ 0 = n
val it = [] : Thm.thm list
> br nat_induct 1;
Level 1 (3 subgoals)
n : nat ==> n #+ 0 = n
1. n : nat ==> n : nat
2. n : nat ==> 0 #+ 0 = 0
3. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> ba 1;
Level 2 (2 subgoals)
n : nat ==> n #+ 0 = n
```

```
1. n : nat ==> 0 #+ 0 = 0
2. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> by (Simp_tac 1);
SIMPLIFIER INVOKED ON THE FOLLOWING TERM:
n : nat ==> 0 #+ 0 = 0
Applying instance of rewrite rule "Arith.add_0_right_natify":
?m1 #+ 0 == natify(?m1)
Rewriting:
0 #+ 0 == natify(0)
Applying instance of rewrite rule "Arith.natify_0":
natify(0) == 0
Rewriting:
natify(0) == 0
Applying instance of rewrite rule "IFOL.IFOL_simps_1":
?a1 = ?a1 == True
Rewriting:
0 = 0 == True
Level 3 (1 subgoal)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> by (Asm_simp_tac 1);
SIMPLIFIER INVOKED ON THE FOLLOWING TERM:
!!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
Adding rewrite rule:
n : nat == True
Adding rewrite rule:
.x : nat == True
Adding rewrite rule:
.x #+ 0 == .x
Applying instance of rewrite rule "Arith.add_0_right_natify":
?m1 #+ 0 == natify(?m1)
Rewriting:
succ(.x) #+ 0 == natify(succ(.x))
Applying instance of rewrite rule "Arith.natify_ident":
?y : nat ==> natify(?y) == ?y
Trying to rewrite:
succ(.x) : nat ==> natify(succ(.x)) == succ(.x)
SIMPLIFIER INVOKED ON THE FOLLOWING TERM:
succ(.x) : nat
[1]Applying instance of rewrite rule "Nat.nat_succ_iff":
succ(?n1) : nat == ?n1 : nat
[1]Rewriting:
succ(.x) : nat == .x : nat
[1]Applying instance of rewrite rule:
.x : nat == True
```

```
[1]Rewriting:
.x : nat == True
[1]SUCCEEDED
natify(succ(.x)) == succ(.x)
[1]Applying instance of rewrite rule "IFOL.IFOL_simps_1":
?a1 = ?a1 == True
[1]Rewriting:
succ(.x) = succ(.x) == True
Level 4
n : nat ==> n #+ 0 = n
No subgoals!
val it = () : unit
```

ou

```
Goal "n:nat ==> n#+0=n";
Level 0 (1 subgoal)
n : nat ==> n #+ 0 = n
1. n : nat ==> n #+ 0 = n
val it = [] : Thm.thm list
> by (Asm_simp_tac 1);
[1]SIMPLIFIER INVOKED ON THE FOLLOWING TERM:
n : nat ==> n #+ 0 = n
Adding rewrite rule:
n : nat == True
Applying instance of rewrite rule "Arith.add_0_right_natify":
?m1 #+ 0 == natify(?m1)
Rewriting:
n #+ 0 == natify(n)
Applying instance of rewrite rule "Arith.natify_ident":
?y : nat ==> natify(?y) == ?y
Trying to rewrite:
n : nat ==> natify(n) == n
SIMPLIFIER INVOKED ON THE FOLLOWING TERM:
n : nat
[1]Applying instance of rewrite rule:
n : nat == True
[1]Rewriting:
n : nat == True
[1]SUCCEEDED
natify(n) == n
[1]Applying instance of rewrite rule "IFOL.IFOL_simps_1":
?a1 = ?a1 == True
[1]Rewriting:
n = n == True
```

```
Level 1
n : nat ==> n #+ 0 = n
No subgoals!
val it = () : unit
```

Capítulo 3

Conclusão

Este trabalho teve por base a teoria de conjuntos de Zermelo-Fraenkel (ZF), que foi uma das primeiras axiomatizações desta teoria.

Na primeira parte, procedeu-se à descrição desta axiomatização, bem como à apresentação das propriedades e teoremas que se obtêm desta teoria.

Na segunda parte, utilizou-se como ferramenta de trabalho o sistema de prova *Isabelle*, para provar as propriedades apresentadas na primeira parte. Estas provas foram feitas, na maioria das vezes, de forma assistida, embora tenham sido referidas e exemplificadas algumas das tácticas de demonstração automática que estão disponíveis. Foram também referidas técnicas que permitem desenvolver tácticas utilizando tacticais. Naturalmente, não se cobriram todos os desenvolvimentos desta teoria que poderão ser objecto de trabalho futuro. Realça-se em particular a demonstração em Isabelle/ZF do lema de Zorn.

Espera-se que esta tese seja útil a quem queira conhecer ou ensinar a Teoria de Conjuntos e, também, a demonstração em computador das suas propriedades.

Apêndice A

Provas em *Isabelle*

Neste apêndice apresenta-se as provas das propriedades cuja prova não foi apresentada anteriormente, uma vez que a mesma não trazia nada de novo.

A.1 FOL

4. $\vdash_{FOL} (\forall_{x,y}\ P(x) \rightarrow Q(y)) \Leftrightarrow (\exists_x\ P(x) \rightarrow \forall_y\ Q(y))$.

```
> Goalw [iff_def] "(ALL x y. P(x)->Q(y))<->((EX x. P(x))->(ALL y. Q(y)))";
Level 0 (1 subgoal)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. ((ALL x y. P(x) -> Q(y)) -> Ex(P) -> All(Q)) &
((Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y)))
val it = [] : Thm.thm list
> br conjI 1;
Level 1 (2 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. (ALL x y. P(x) -> Q(y)) -> Ex(P) -> All(Q)
2. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> br impI 1;
Level 2 (2 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. ALL x y. P(x) -> Q(y) ==> Ex(P) -> All(Q)
2. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> br impI 1;
Level 3 (2 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. [| ALL x y. P(x) -> Q(y); Ex(P) |] ==> All(Q)
2. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
```

```
> br allI 1;
Level 4 (2 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x. [| ALL x y. P(x) -> Q(y); Ex(P) |] ==> Q(x)
2. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> be exE 1;
Level 5 (2 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x xa. [| ALL x y. P(x) -> Q(y); P(xa) |] ==> Q(x)
2. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> br mp 1;
Level 6 (3 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x xa. [| ALL x y. P(x) -> Q(y); P(xa) |] ==> ?P5(x, xa) -> Q(x)
2. !!x xa. [| ALL x y. P(x) -> Q(y); P(xa) |] ==> ?P5(x, xa)
3. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> br spec 1;
Level 7 (3 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x xa.
[| ALL x y. P(x) -> Q(y); P(xa) |]
==> ALL xb. ?P8(x, xb, xa) -> Q(xb)
2. !!x xa. [| ALL x y. P(x) -> Q(y); P(xa) |] ==> ?P8(x, x, xa)
3. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> br spec 1;
Level 8 (3 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x xa.
[| ALL x y. P(x) -> Q(y); P(xa) |]
==> ALL xb xc. ?P10(x, xa, xb, xc) -> Q(xc)
2. !!x xa.
[| ALL x y. P(x) -> Q(y); P(xa) |] ==> ?P10(x, xa, ?x9(x, xa), x)
3. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> ba 1;
Level 9 (2 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x xa. [| ALL x y. P(x) -> Q(y); P(xa) |] ==> P(?x9(x, xa))
2. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> ba 1;
Level 10 (1 subgoal)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
```

```
1. (Ex(P) -> All(Q)) -> (ALL x y. P(x) -> Q(y))
val it = () : unit
> br impI 1;
Level 11 (1 subgoal)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. Ex(P) -> All(Q) ==> ALL x y. P(x) -> Q(y)
val it = () : unit
> br allI 1;
Level 12 (1 subgoal)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x. Ex(P) -> All(Q) ==> ALL y. P(x) -> Q(y)
val it = () : unit
> br allI 1;
Level 13 (1 subgoal)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x y. Ex(P) -> All(Q) ==> P(x) -> Q(y)
val it = () : unit
> br impI 1;
Level 14 (1 subgoal)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x y. [| Ex(P) -> All(Q); P(x) |] ==> Q(y)
val it = () : unit
> br spec 1;
Level 15 (1 subgoal)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x y. [| Ex(P) -> All(Q); P(x) |] ==> ALL x. Q(x)
val it = () : unit
> br mp 1;
Level 16 (2 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x y. [| Ex(P) -> All(Q); P(x) |] ==> ?P17(x, y) -> (ALL x. Q(x))
2. !!x y. [| Ex(P) -> All(Q); P(x) |] ==> ?P17(x, y)
val it = () : unit
> br exI 2;
Level 17 (2 subgoals)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x y.
[| Ex(P) -> All(Q); P(x) |]
==> (EX xa. ?P18(x, y, xa)) -> (ALL x. Q(x))
2. !!x y. [| Ex(P) -> All(Q); P(x) |] ==> ?P18(x, y, ?x18(x, y))
val it = () : unit
> ba 1;
Level 18 (1 subgoal)
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
1. !!x y. [| Ex(P) -> All(Q); P(x) |] ==> P(?x18(x, y))
val it = () : unit
> ba 1;
```

```
Level 19
(ALL x y. P(x) -> Q(y)) <-> (EX x. P(x)) -> (ALL y. Q(y))
No subgoals!
val it = () : unit
```

A.2 ZF

A.2.1 Propriedades da união

1. $\vdash_{ZF} \cup\varnothing = \varnothing$

```
> Goal "Union(0)=0";
Level 0 (1 subgoal)
Union(0) = 0
1. Union(0) = 0
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
Union(0) = 0
1. Union(0) <= 0
2. 0 <= Union(0)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
Union(0) = 0
1. !!x. x : Union(0) ==> x : 0
2. 0 <= Union(0)
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
Union(0) = 0
1. !!x. x : Union(0) ==> x : 0
2. !!x. x : 0 ==> x : Union(0)
val it = () : unit
> be emptyE 2;
Level 4 (1 subgoal)
Union(0) = 0
1. !!x. x : Union(0) ==> x : 0
val it = () : unit
> be UnionE 1;
Level 5 (1 subgoal)
Union(0) = 0
1. !!x B. [| x : B; B : 0 |] ==> x : 0
val it = () : unit
> be emptyE 1;
```

```
Level 6
Union(0) = 0
No subgoals!
val it = () : unit
```

3. $\vdash_{ZF} A \cup \varnothing = A$

```
> Goal "A Un 0=A";
Level 0 (1 subgoal)
A Un 0 = A
1. A Un 0 = A
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Un 0 = A
1. A Un 0 <= A
2. A <= A Un 0
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Un 0 = A
1. !!x. x : A Un 0 ==> x : A
2. A <= A Un 0
val it = () : unit
> > br subsetI 2;
Level 3 (2 subgoals)
A Un 0 = A
1. !!x. x : A Un 0 ==> x : A
2. !!x. x : A ==> x : A Un 0
val it = () : unit
> be UnE 1;
Level 4 (3 subgoals)
A Un 0 = A
1. !!x. x : A ==> x : A
2. !!x. x : 0 ==> x : A
3. !!x. x : A ==> x : A Un 0
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
A Un 0 = A
1. !!x. x : 0 ==> x : A
2. !!x. x : A ==> x : A Un 0
val it = () : unit
> be emptyE 1;
Level 6 (1 subgoal)
A Un 0 = A
1. !!x. x : A ==> x : A Un 0
```

```
val it = () : unit
> br UnI1 1;
Level 7 (1 subgoal)
A Un 0 = A
1. !!x. x : A ==> x : A
val it = () : unit
> ba 1;
Level 8
A Un 0 = A
No subgoals!
val it = () : unit
```

5- $\vdash_{ZF} A \cup (B \cup C) = (A \cup B) \cup C$

```
> Goal "A Un (B Un C) = (A Un B) Un C";
Level 0 (1 subgoal)
A Un (B Un C) = A Un B Un C
1. A Un (B Un C) = A Un B Un C
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. A Un (B Un C) <= A Un B Un C
2. A Un B Un C <= A Un (B Un C)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : A Un (B Un C) ==> x : A Un B Un C
2. A Un B Un C <= A Un (B Un C)
val it = () : unit
> be UnE 1;
Level 3 (3 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : A ==> x : A Un B Un C
2. !!x. x : B Un C ==> x : A Un B Un C
3. A Un B Un C <= A Un (B Un C)
val it = () : unit
> br UnI1 1;
Level 4 (3 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : A ==> x : A Un B
2. !!x. x : B Un C ==> x : A Un B Un C
3. A Un B Un C <= A Un (B Un C)
val it = () : unit
> br UnI1 1;
Level 5 (3 subgoals)
```

```
A Un (B Un C) = A Un B Un C
1. !!x. x : A ==> x : A
2. !!x. x : B Un C ==> x : A Un B Un C
3. A Un B Un C <= A Un (B Un C)
val it = () : unit
> ba 1;
Level 6 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : B Un C ==> x : A Un B Un C
2. A Un B Un C <= A Un (B Un C)
val it = () : unit
> be UnE 1;
Level 7 (3 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : B ==> x : A Un B Un C
2. !!x. x : C ==> x : A Un B Un C
3. A Un B Un C <= A Un (B Un C)
val it = () : unit
> br UnI1 1;
Level 8 (3 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : B ==> x : A Un B
2. !!x. x : C ==> x : A Un B Un C
3. A Un B Un C <= A Un (B Un C)
val it = () : unit
> br UnI2 1;
Level 9 (3 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : B ==> x : B
2. !!x. x : C ==> x : A Un B Un C
3. A Un B Un C <= A Un (B Un C)
val it = () : unit
> ba 1;
Level 10 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : C ==> x : A Un B Un C
2. A Un B Un C <= A Un (B Un C)
val it = () : unit
> br UnI2 1;
Level 11 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : C ==> x : C
2. A Un B Un C <= A Un (B Un C)
val it = () : unit
> ba 1;
Level 12 (1 subgoal)
A Un (B Un C) = A Un B Un C
```

```
1. A Un B Un C <= A Un (B Un C)
val it = () : unit
> br subsetI 1;
Level 13 (1 subgoal)
A Un (B Un C) = A Un B Un C
1. !!x. x : A Un B Un C ==> x : A Un (B Un C)
val it = () : unit
> be UnE 1;
Level 14 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : A Un B ==> x : A Un (B Un C)
2. !!x. x : C ==> x : A Un (B Un C)
val it = () : unit
> be UnE 1;
Level 15 (3 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : A ==> x : A Un (B Un C)
2. !!x. x : B ==> x : A Un (B Un C)
3. !!x. x : C ==> x : A Un (B Un C)
val it = () : unit
> br UnI1 1;
Level 16 (3 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : A ==> x : A
2. !!x. x : B ==> x : A Un (B Un C)
3. !!x. x : C ==> x : A Un (B Un C)
val it = () : unit
> ba 1;
Level 17 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : B ==> x : A Un (B Un C)
2. !!x. x : C ==> x : A Un (B Un C)
val it = () : unit
> br UnI2 1;
Level 18 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : B ==> x : B Un C
2. !!x. x : C ==> x : A Un (B Un C)
val it = () : unit
> br UnI1 1;
Level 19 (2 subgoals)
A Un (B Un C) = A Un B Un C
1. !!x. x : B ==> x : B
2. !!x. x : C ==> x : A Un (B Un C)
val it = () : unit
> ba 1;
Level 20 (1 subgoal)
```

```
A Un (B Un C) = A Un B Un C
1. !!x. x : C ==> x : A Un (B Un C)
val it = () : unit
> br UnI2 1;
Level 21 (1 subgoal)
A Un (B Un C) = A Un B Un C
1. !!x. x : C ==> x : B Un C
val it = () : unit
> br UnI2 1;
Level 22 (1 subgoal)
A Un (B Un C) = A Un B Un C
1. !!x. x : C ==> x : C
val it = () : unit
> ba 1;
Level 23
A Un (B Un C) = A Un B Un C
No subgoals!
val it = () : unit
```

6- $\vdash_{ZF} A \cup A = A$

```
> Goal " A Un A =A";
Level 0 (1 subgoal)
A Un A = A
1. A Un A = A
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Un A = A
1. A Un A <= A
2. A <= A Un A
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Un A = A
1. !!x. x : A Un A ==> x : A
2. A <= A Un A
val it = () : unit
> be UnE 1;
Level 3 (3 subgoals)
A Un A = A
1. !!x. x : A ==> x : A
2. !!x. x : A ==> x : A
3. A <= A Un A
val it = () : unit
> ba 1;
Level 4 (2 subgoals)
```

```
A Un A = A
1. !!x. x : A ==> x : A
2. A <= A Un A
val it = () : unit
> ba 1;
Level 5 (1 subgoal)
A Un A = A
1. A <= A Un A
val it = () : unit
> br subsetI 1;
Level 6 (1 subgoal)
A Un A = A
1. !!x. x : A ==> x : A Un A
val it = () : unit
> br UnI1 1;
Level 7 (1 subgoal)
A Un A = A
1. !!x. x : A ==> x : A
val it = () : unit
> ba 1;
Level 8
A Un A = A
No subgoals!
val it = () : unit
```

7- $\vdash_{ZF} A \subseteq A \cup B$

```
> Goal "A <= A Un B";
Level 0 (1 subgoal)
A <= A Un B
1. A <= A Un B
val it = [] : Thm.thm list
> br subsetI 1;
Level 1 (1 subgoal)
A <= A Un B
1. !!x. x : A ==> x : A Un B
val it = () : unit
> br UnI1 1;
Level 2 (1 subgoal)
A <= A Un B
1. !!x. x : A ==> x : A
val it = () : unit
> ba 1;
Level 3
A <= A Un B
No subgoals!
val it = () : unit
```

8- $\vdash_{ZF} A \subseteq B \Leftrightarrow A \cup B = B$

```
> Goal "A<=B <-> A Un B = B";
Level 0 (1 subgoal)
A <= B <-> A Un B = B
1. A <= B <-> A Un B = B
val it = [] : Thm.thm list
> br iffI 1;
Level 1 (2 subgoals)
A <= B <-> A Un B = B
1. A <= B ==> A Un B = B
2. A Un B = B ==> A <= B
val it = () : unit
> br equalityI 1;
Level 2 (3 subgoals)
A <= B <-> A Un B = B
1. A <= B ==> A Un B <= B
2. A <= B ==> B <= A Un B
3. A Un B = B ==> A <= B
val it = () : unit
> br subsetI 1;
Level 3 (3 subgoals)
A <= B <-> A Un B = B
1. !!x. [| A <= B; x : A Un B |] ==> x : B
2. A <= B ==> B <= A Un B
3. A Un B = B ==> A <= B
val it = () : unit
> be UnE 1;
Level 4 (4 subgoals)
A <= B <-> A Un B = B
1. !!x. [| A <= B; x : A |] ==> x : B
2. !!x. [| A <= B; x : B |] ==> x : B
3. A <= B ==> B <= A Un B
4. A Un B = B ==> A <= B
val it = () : unit
> ba 2;
Level 5 (3 subgoals)
A <= B <-> A Un B = B
1. !!x. [| A <= B; x : A |] ==> x : B
2. A <= B ==> B <= A Un B
3. A Un B = B ==> A <= B
val it = () : unit
> be subsetD 1;
Level 6 (3 subgoals)
A <= B <-> A Un B = B
1. !!x. x : A ==> x : A
2. A <= B ==> B <= A Un B
```

8- $\vdash_{ZF} A \subseteq B \Leftrightarrow A \cup B = B$

```
> Goal "A<=B <-> A Un B = B";
Level 0 (1 subgoal)
A <= B <-> A Un B = B
1. A <= B <-> A Un B = B
val it = [] : Thm.thm list
> br iffI 1;
Level 1 (2 subgoals)
A <= B <-> A Un B = B
1. A <= B ==> A Un B = B
2. A Un B = B ==> A <= B
val it = () : unit
> br equalityI 1;
Level 2 (3 subgoals)
A <= B <-> A Un B = B
1. A <= B ==> A Un B <= B
2. A <= B ==> B <= A Un B
3. A Un B = B ==> A <= B
val it = () : unit
> br subsetI 1;
Level 3 (3 subgoals)
A <= B <-> A Un B = B
1. !!x. [| A <= B; x : A Un B |] ==> x : B
2. A <= B ==> B <= A Un B
3. A Un B = B ==> A <= B
val it = () : unit
> be UnE 1;
Level 4 (4 subgoals)
A <= B <-> A Un B = B
1. !!x. [| A <= B; x : A |] ==> x : B
2. !!x. [| A <= B; x : B |] ==> x : B
3. A <= B ==> B <= A Un B
4. A Un B = B ==> A <= B
val it = () : unit
> ba 2;
Level 5 (3 subgoals)
A <= B <-> A Un B = B
1. !!x. [| A <= B; x : A |] ==> x : B
2. A <= B ==> B <= A Un B
3. A Un B = B ==> A <= B
val it = () : unit
> be subsetD 1;
Level 6 (3 subgoals)
A <= B <-> A Un B = B
1. !!x. x : A ==> x : A
2. A <= B ==> B <= A Un B
```

```
3. A Un B = B ==> A <= B
val it = () : unit
> ba 1;
Level 7 (2 subgoals)
A <= B <-> A Un B = B
1. A <= B ==> B <= A Un B
2. A Un B = B ==> A <= B
val it = () : unit
> br subsetI 1;
Level 8 (2 subgoals)
A <= B <-> A Un B = B
1. !!x. [| A <= B; x : B |] ==> x : A Un B
2. A Un B = B ==> A <= B
val it = () : unit
> br UnI2 1;
Level 9 (2 subgoals)
A <= B <-> A Un B = B
1. !!x. [| A <= B; x : B |] ==> x : B
2. A Un B = B ==> A <= B
val it = () : unit
> ba 1;
Level 10 (1 subgoal)
A <= B <-> A Un B = B
1. A Un B = B ==> A <= B
val it = () : unit
> br subsetI 1;
Level 11 (1 subgoal)
A <= B <-> A Un B = B
1. !!x. [| A Un B = B; x : A |] ==> x : B
val it = () : unit
> be equalityE 1;
Level 12 (1 subgoal)
A <= B <-> A Un B = B
1. !!x. [| x : A; A Un B <= B; B <= A Un B |] ==> x : B
val it = () : unit
> br subsetD 1;
Level 13 (2 subgoals)
A <= B <-> A Un B = B
1. !!x. [| x : A; A Un B <= B; B <= A Un B |] ==> ?A9(x) <= B
2. !!x. [| x : A; A Un B <= B; B <= A Un B |] ==> x : ?A9(x)
val it = () : unit
> ba 1;
Level 14 (1 subgoal)
A <= B <-> A Un B = B
1. !!x. [| x : A; A Un B <= B; B <= A Un B |] ==> x : A Un B
val it = () : unit
> br UnI1 1;
```

```
Level 15 (1 subgoal)
A <= B <-> A Un B = B
1. !!x. [| x : A; A Un B <= B; B <= A Un B |] ==> x : A
val it = () : unit
> ba 1;
Level 16
A <= B <-> A Un B = B
No subgoals!
val it = () : unit
```

A.2.2 Propriedades da intersecção

3. $\vdash_{ZF} A \cap (B \cap C) = (A \cap B) \cap C$

```
> Goal "A Int (B Int C) = (A Int B) Int C";
Level 0 (1 subgoal)
A Int (B Int C) = A Int B Int C
1. A Int (B Int C) = A Int B Int C
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. A Int (B Int C) <= A Int B Int C
2. A Int B Int C <= A Int (B Int C)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : A Int B Int C
2. A Int B Int C <= A Int (B Int C)
val it = () : unit
> br IntI 1;
Level 3 (3 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : A Int B
2. !!x. x : A Int (B Int C) ==> x : C
3. A Int B Int C <= A Int (B Int C)
val it = () : unit
> br IntI 1;
Level 4 (4 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : A
2. !!x. x : A Int (B Int C) ==> x : B
3. !!x. x : A Int (B Int C) ==> x : C
4. A Int B Int C <= A Int (B Int C)
val it = () : unit
```

```
> br IntD1 1;
Level 5 (4 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : A Int ?B4(x)
2. !!x. x : A Int (B Int C) ==> x : B
3. !!x. x : A Int (B Int C) ==> x : C
4. A Int B Int C <= A Int (B Int C)
val it = () : unit
> ba 1;
Level 6 (3 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : B
2. !!x. x : A Int (B Int C) ==> x : C
3. A Int B Int C <= A Int (B Int C)
val it = () : unit
> br IntD1 1;
Level 7 (3 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : B Int ?B5(x)
2. !!x. x : A Int (B Int C) ==> x : C
3. A Int B Int C <= A Int (B Int C)
val it = () : unit
> br IntD2 1;
Level 8 (3 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : ?A6(x) Int (B Int ?B5(x))
2. !!x. x : A Int (B Int C) ==> x : C
3. A Int B Int C <= A Int (B Int C)
val it = () : unit
> ba 1;
Level 9 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : C
2. A Int B Int C <= A Int (B Int C)
val it = () : unit
> br IntD2 1;
Level 10 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : ?A7(x) Int C
2. A Int B Int C <= A Int (B Int C)
val it = () : unit
> br IntD2 1;
Level 11 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int (B Int C) ==> x : ?A8(x) Int (?A7(x) Int C)
2. A Int B Int C <= A Int (B Int C)
val it = () : unit
```

```
> ba 1;
Level 12 (1 subgoal)
A Int (B Int C) = A Int B Int C
1. A Int B Int C <= A Int (B Int C)
val it = () : unit
> br subsetI 1;
Level 13 (1 subgoal)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : A Int (B Int C)
val it = () : unit
> br IntI 1;
Level 14 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : A
2. !!x. x : A Int B Int C ==> x : B Int C
val it = () : unit
> br IntD1 1;
Level 15 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : A Int ?B11(x)
2. !!x. x : A Int B Int C ==> x : B Int C
val it = () : unit
> br IntD1 1;
Level 16 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : A Int ?B11(x) Int ?B12(x)
2. !!x. x : A Int B Int C ==> x : B Int C
val it = () : unit
> ba 1;
Level 17 (1 subgoal)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : B Int C
val it = () : unit
> br IntI 1;
Level 18 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : B
2. !!x. x : A Int B Int C ==> x : C
val it = () : unit
> br IntD2 1;
Level 19 (2 subgoals)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : ?A14(x) Int B
2. !!x. x : A Int B Int C ==> x : C
val it = () : unit
> br IntD1 1;
Level 20 (2 subgoals)
```

```
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : ?A14(x) Int B Int ?B15(x)
2. !!x. x : A Int B Int C ==> x : C
val it = () : unit
> ba 1;
Level 21 (1 subgoal)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : C
val it = () : unit
> br IntD2 1;
Level 22 (1 subgoal)
A Int (B Int C) = A Int B Int C
1. !!x. x : A Int B Int C ==> x : ?A16(x) Int C
val it = () : unit
> ba 1;
Level 23
A Int (B Int C) = A Int B Int C
No subgoals!
val it = () : unit
```

4- $\vdash_{ZF} A \cap A = A$

```
> Goal "A Int A = A";
Level 0 (1 subgoal)
A Int A = A
1. A Int A = A
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Int A = A
1. A Int A <= A
2. A <= A Int A
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Int A = A
1. !!x. x : A Int A ==> x : A
2. A <= A Int A
val it = () : unit
> br IntD1 1;
Level 3 (2 subgoals)
A Int A = A
1. !!x. x : A Int A ==> x : A Int ?B2(x)
2. A <= A Int A
val it = () : unit
> ba 1;
Level 4 (1 subgoal)
```

```
A Int A = A
1. A <= A Int A
val it = () : unit
> br subsetI 1;
Level 5 (1 subgoal)
A Int A = A
1. !!x. x : A ==> x : A Int A
val it = () : unit
> br IntI 1;
Level 6 (2 subgoals)
A Int A = A
1. !!x. x : A ==> x : A
2. !!x. x : A ==> x : A
val it = () : unit
> ba 1;
Level 7 (1 subgoal)
A Int A = A
1. !!x. x : A ==> x : A
val it = () : unit
> ba 1;
Level 8
A Int A = A
No subgoals!
val it = () : unit
```

5- $\vdash_{ZF} A \subseteq B \Leftrightarrow A \cap B = A$

```
> Goal " A <= B <-> A Int B = A";
Level 0 (1 subgoal)
A <= B <-> A Int B = A
1. A <= B <-> A Int B = A
val it = [] : Thm.thm list
> br iffI 1;
Level 1 (2 subgoals)
A <= B <-> A Int B = A
1. A <= B ==> A Int B = A
2. A Int B = A ==> A <= B
val it = () : unit
> br equalityI 1;
Level 2 (3 subgoals)
A <= B <-> A Int B = A
1. A <= B ==> A Int B <= A
2. A <= B ==> A <= A Int B
3. A Int B = A ==> A <= B
val it = () : unit
> br subsetI 1;
Level 3 (3 subgoals)
```

```
A <= B <-> A Int B = A
1. !!x. [| A <= B; x : A Int B |] ==> x : A
2. A <= B ==> A <= A Int B
3. A Int B = A ==> A <= B
val it = () : unit
> br IntD1 1;
Level 4 (3 subgoals)
A <= B <-> A Int B = A
1. !!x. [| A <= B; x : A Int B |] ==> x : A Int ?B3(x)
2. A <= B ==> A <= A Int B
3. A Int B = A ==> A <= B
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
A <= B <-> A Int B = A
1. A <= B ==> A <= A Int B
2. A Int B = A ==> A <= B
val it = () : unit
> br subsetI 1;
Level 6 (2 subgoals)
A <= B <-> A Int B = A
1. !!x. [| A <= B; x : A |] ==> x : A Int B
2. A Int B = A ==> A <= B
val it = () : unit
> br IntI 1;
Level 7 (3 subgoals)
A <= B <-> A Int B = A
1. !!x. [| A <= B; x : A |] ==> x : A
2. !!x. [| A <= B; x : A |] ==> x : B
3. A Int B = A ==> A <= B
val it = () : unit
> ba 1;
Level 8 (2 subgoals)
A <= B <-> A Int B = A
1. !!x. [| A <= B; x : A |] ==> x : B
2. A Int B = A ==> A <= B
val it = () : unit
> br subsetD 1;
Level 9 (3 subgoals)
A <= B <-> A Int B = A
1. !!x. [| A <= B; x : A |] ==> ?A6(x) <= B
2. !!x. [| A <= B; x : A |] ==> x : ?A6(x)
3. A Int B = A ==> A <= B
val it = () : unit
> ba 1;
Level 10 (2 subgoals)
A <= B <-> A Int B = A
```

```
1. !!x. [| A <= B; x : A |] ==> x : A
2. A Int B = A ==> A <= B
val it = () : unit
> ba 1;
Level 11 (1 subgoal)
A <= B <-> A Int B = A
1. A Int B = A ==> A <= B
val it = () : unit
> be equalityE 1;
Level 12 (1 subgoal)
A <= B <-> A Int B = A
1. [| A Int B <= A; A <= A Int B |] ==> A <= B
val it = () : unit
> br subsetI 1;
Level 13 (1 subgoal)
A <= B <-> A Int B = A
1. !!x. [| A Int B <= A; A <= A Int B; x : A |] ==> x : B
val it = () : unit
> br IntD2 1;
Level 14 (1 subgoal)
A <= B <-> A Int B = A
1. !!x. [| A Int B <= A; A <= A Int B; x : A |] ==> x : ?A9(x) Int B
val it = () : unit
> br subsetD 1;
Level 15 (2 subgoals)
A <= B <-> A Int B = A
1. !!x. [| A Int B <= A; A <= A Int B; x : A |] ==> ?A10(x) <= ?A9(x) Int B
2. !!x. [| A Int B <= A; A <= A Int B; x : A |] ==> x : ?A10(x)
val it = () : unit
> ba 1;
Level 16 (1 subgoal)
A <= B <-> A Int B = A
1. !!x. [| A Int B <= A; A <= A Int B; x : A |] ==> x : A
val it = () : unit
> > ba 1;
Level 17
A <= B <-> A Int B = A
No subgoals!
val it = () : unit
```

6- $A \subseteq B \wedge A \subseteq C \rightarrow A \subseteq B \cap C$

```
> Goal "[|A<=B&A<=C|]==>A<=(B Int C)";
Level 0 (1 subgoal)
A <= B & A <= C ==> A <= B Int C
1. A <= B & A <= C ==> A <= B Int C
val it = [] : Thm.thm list
```

```
> br subsetI 1;
Level 1 (1 subgoal)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> x : B Int C
val it = () : unit
> br IntI 1;
Level 2 (2 subgoals)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> x : B
2. !!x. [| A <= B & A <= C; x : A |] ==> x : C
val it = () : unit
> br subsetD 1;
Level 3 (3 subgoals)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> ?A2(x) <= B
2. !!x. [| A <= B & A <= C; x : A |] ==> x : ?A2(x)
3. !!x. [| A <= B & A <= C; x : A |] ==> x : C
val it = () : unit
> br conjunct1 1;
Level 4 (3 subgoals)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> ?A2(x) <= B & ?Q3(x)
2. !!x. [| A <= B & A <= C; x : A |] ==> x : ?A2(x)
3. !!x. [| A <= B & A <= C; x : A |] ==> x : C
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> x : A
2. !!x. [| A <= B & A <= C; x : A |] ==> x : C
val it = () : unit
> ba 1;
Level 6 (1 subgoal)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> x : C
val it = () : unit
> br subsetD 1;
Level 7 (2 subgoals)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> ?A4(x) <= C
2. !!x. [| A <= B & A <= C; x : A |] ==> x : ?A4(x)
val it = () : unit
> br conjunct2 1;
Level 8 (2 subgoals)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> ?P5(x) & ?A4(x) <= C
2. !!x. [| A <= B & A <= C; x : A |] ==> x : ?A4(x)
```

```
val it = () : unit
> ba 1;
Level 9 (1 subgoal)
A <= B & A <= C ==> A <= B Int C
1. !!x. [| A <= B & A <= C; x : A |] ==> x : A
val it = () : unit
> ba 1;
Level 10
A <= B & A <= C ==> A <= B Int C
No subgoals!
val it = () : unit
```

A.2.3 Propriedades distributivas

1. $\vdash_{ZF} A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

```
> Goal "A Int (B Un C)=(A Int B)Un(A Int C)";
Level 0 (1 subgoal)
A Int (B Un C) = A Int B Un A Int C
1. A Int (B Un C) = A Int B Un A Int C
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. A Int (B Un C) <= A Int B Un A Int C
2. A Int B Un A Int C <= A Int (B Un C)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. x : A Int (B Un C) ==> x : A Int B Un A Int C
2. A Int B Un A Int C <= A Int (B Un C)
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. x : A Int (B Un C) ==> x : A Int B Un A Int C
2. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> br UnE 1;
Level 4 (4 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. x : A Int (B Un C) ==> ?c3(x) : ?A3(x) Un ?B3(x)
2. !!x. [| x : A Int (B Un C); ?c3(x) : ?A3(x) |]
==> x : A Int B Un A Int C
3. !!x. [| x : A Int (B Un C); ?c3(x) : ?B3(x) |]
```

```
==> x : A Int B Un A Int C
4. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> br IntD2 1;
Level 5 (4 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. x : A Int (B Un C) ==> ?c3(x) : ?A4(x) Int (?A3(x) Un ?B3(x))
2. !!x. [| x : A Int (B Un C); ?c3(x) : ?A3(x) |]
==> x : A Int B Un A Int C
3. !!x. [| x : A Int (B Un C); ?c3(x) : ?B3(x) |]
==> x : A Int B Un A Int C
4. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> ba 1;
Level 6 (3 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int (B Un C); x : B |] ==> x : A Int B Un A Int C
2. !!x. [| x : A Int (B Un C); x : C |] ==> x : A Int B Un A Int C
3. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> br UnI1 1;
Level 7 (3 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int (B Un C); x : B |] ==> x : A Int B
2. !!x. [| x : A Int (B Un C); x : C |] ==> x : A Int B Un A Int C
3. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> br IntI 1;
Level 8 (4 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int (B Un C); x : B |] ==> x : A
2. !!x. [| x : A Int (B Un C); x : B |] ==> x : B
3. !!x. [| x : A Int (B Un C); x : C |] ==> x : A Int B Un A Int C
4. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> ba 2;
Level 9 (3 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int (B Un C); x : B |] ==> x : A
2. !!x. [| x : A Int (B Un C); x : C |] ==> x : A Int B Un A Int C
3. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> be IntD1 1;
Level 10 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int (B Un C); x : C |] ==> x : A Int B Un A Int C
2. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
```

```
val it = () : unit
> br UnI2 1;
Level 11 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int (B Un C); x : C |] ==> x : A Int C
2. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> br IntI 1;
Level 12 (3 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int (B Un C); x : C |] ==> x : A
2. !!x. [| x : A Int (B Un C); x : C |] ==> x : C
3. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> ba 2;
Level 13 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int (B Un C); x : C |] ==> x : A
2. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> be IntD1 1;
Level 14 (1 subgoal)
A Int (B Un C) = A Int B Un A Int C
1. !!x. x : A Int B Un A Int C ==> x : A Int (B Un C)
val it = () : unit
> br UnE 1;
Level 15 (3 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. x : A Int B Un A Int C ==> ?c11(x) : ?A11(x) Un ?B11(x)
2. !!x. [| x : A Int B Un A Int C; ?c11(x) : ?A11(x) |]
==> x : A Int (B Un C)
3. !!x. [| x : A Int B Un A Int C; ?c11(x) : ?B11(x) |]
==> x : A Int (B Un C)
val it = () : unit
> ba 1;
Level 16 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int B Un A Int C; x : A Int B |] ==> x : A Int (B Un C)
2. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : A Int (B Un C)
val it = () : unit
> br IntI 1;
Level 17 (3 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int B Un A Int C; x : A Int B |] ==> x : A
2. !!x. [| x : A Int B Un A Int C; x : A Int B |] ==> x : B Un C
3. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : A Int (B Un C)
val it = () : unit
```

```
> be IntD1 1;
Level 18 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int B Un A Int C; x : A Int B |] ==> x : B Un C
2. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : A Int (B Un C)
val it = () : unit
> br UnI1 1;
Level 19 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int B Un A Int C; x : A Int B |] ==> x : B
2. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : A Int (B Un C)
val it = () : unit
> be IntD2 1;
Level 20 (1 subgoal)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : A Int (B Un C)
val it = () : unit
> br IntI 1;
Level 21 (2 subgoals)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : A
2. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : B Un C
val it = () : unit
> be IntD1 1;
Level 22 (1 subgoal)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : B Un C
val it = () : unit
> > br UnI2 1;
Level 23 (1 subgoal)
A Int (B Un C) = A Int B Un A Int C
1. !!x. [| x : A Int B Un A Int C; x : A Int C |] ==> x : C
val it = () : unit
> be IntD2 1;
Level 24
A Int (B Un C) = A Int B Un A Int C
No subgoals!
val it = () : unit
```

2. $\vdash_{ZF} A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

```
> Goal "A Un (B Int C) = (A Un B) Int (A Un C)";
Level 0 (1 subgoal)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C = (A Un B) Int (A Un C)
val it = [] : Thm.thm list
> br equalityI 1;
```

```
Level 1 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. (A Un B) Int (A Un C) <= A Un B Int C
val it = () : unit
> br subsetI 2;
Level 2 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. x : (A Un B) Int (A Un C) ==> x : A Un B Int C
val it = () : unit
> be IntE 2;
Level 3 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un B; x : A Un C |] ==> x : A Un B Int C
val it = () : unit
> be UnE 2;
Level 4 (3 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un C; x : A |] ==> x : A Un B Int C
3. !!x. [| x : A Un C; x : B |] ==> x : A Un B Int C
val it = () : unit
> be UnE 3;
Level 5 (4 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un C; x : A |] ==> x : A Un B Int C
3. !!x. [| x : B; x : A |] ==> x : A Un B Int C
4. !!x. [| x : B; x : C |] ==> x : A Un B Int C
val it = () : unit
> br UnI1 3;
Level 6 (4 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un C; x : A |] ==> x : A Un B Int C
3. !!x. [| x : B; x : A |] ==> x : A
4. !!x. [| x : B; x : C |] ==> x : A Un B Int C
val it = () : unit
> ba 3;
Level 7 (3 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un C; x : A |] ==> x : A Un B Int C
3. !!x. [| x : B; x : C |] ==> x : A Un B Int C
val it = () : unit
```

```
> br UnI2 3;
Level 8 (3 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un C; x : A |] ==> x : A Un B Int C
3. !!x. [| x : B; x : C |] ==> x : B Int C
val it = () : unit
> br IntI 3;
Level 9 (4 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un C; x : A |] ==> x : A Un B Int C
3. !!x. [| x : B; x : C |] ==> x : B
4. !!x. [| x : B; x : C |] ==> x : C
val it = () : unit
> ba 3;
Level 10 (3 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un C; x : A |] ==> x : A Un B Int C
3. !!x. [| x : B; x : C |] ==> x : C
val it = () : unit
> ba 3;
Level 11 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A Un C; x : A |] ==> x : A Un B Int C
val it = () : unit
> be UnE 2;
Level 12 (3 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A; x : A |] ==> x : A Un B Int C
3. !!x. [| x : A; x : C |] ==> x : A Un B Int C
val it = () : unit
> br UnI1 3;
Level 13 (3 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A; x : A |] ==> x : A Un B Int C
3. !!x. [| x : A; x : C |] ==> x : A
val it = () : unit
> ba 3;
Level 14 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A; x : A |] ==> x : A Un B Int C
```

```
val it = () : unit
> br UnI1 2;
Level 15 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
2. !!x. [| x : A; x : A |] ==> x : A
val it = () : unit
> ba 2;
Level 16 (1 subgoal)
A Un B Int C = (A Un B) Int (A Un C)
1. A Un B Int C <= (A Un B) Int (A Un C)
val it = () : unit
> br subsetI 1;
Level 17 (1 subgoal)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : A Un B Int C ==> x : (A Un B) Int (A Un C)
val it = () : unit
> be UnE 1;
Level 18 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : A ==> x : (A Un B) Int (A Un C)
2. !!x. x : B Int C ==> x : (A Un B) Int (A Un C)
val it = () : unit
> br IntI 1;
Level 19 (3 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : A ==> x : A Un B
2. !!x. x : A ==> x : A Un C
3. !!x. x : B Int C ==> x : (A Un B) Int (A Un C)
val it = () : unit
> br UnI1 1;
Level 20 (3 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : A ==> x : A
2. !!x. x : A ==> x : A Un C
3. !!x. x : B Int C ==> x : (A Un B) Int (A Un C)
val it = () : unit
> ba 1;
Level 21 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : A ==> x : A Un C
2. !!x. x : B Int C ==> x : (A Un B) Int (A Un C)
val it = () : unit
> br UnI1 1;
Level 22 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : A ==> x : A
```

```
2. !!x. x : B Int C ==> x : (A Un B) Int (A Un C)
val it = () : unit
> ba 1;
Level 23 (1 subgoal)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : B Int C ==> x : (A Un B) Int (A Un C)
val it = () : unit
> br IntI 1;
Level 24 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : B Int C ==> x : A Un B
2. !!x. x : B Int C ==> x : A Un C
val it = () : unit
> br UnI2 1;
Level 25 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : B Int C ==> x : B
2. !!x. x : B Int C ==> x : A Un C
val it = () : unit
> br IntD1 1;
Level 26 (2 subgoals)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : B Int C ==> x : B Int ?B18(x)
2. !!x. x : B Int C ==> x : A Un C
val it = () : unit
> ba 1;
Level 27 (1 subgoal)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : B Int C ==> x : A Un C
val it = () : unit
> br UnI2 1;
Level 28 (1 subgoal)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : B Int C ==> x : C
val it = () : unit
> br IntD2 1;
Level 29 (1 subgoal)
A Un B Int C = (A Un B) Int (A Un C)
1. !!x. x : B Int C ==> x : ?A20(x) Int C
val it = () : unit
> ba 1;
Level 30
A Un B Int C = (A Un B) Int (A Un C)
No subgoals!
val it = () : unit
```

A.2.4 Propriedades da complementação

2. $\vdash_{ZF} \varnothing' = E$

```
> Goal "E - 0 = E";
Level 0 (1 subgoal)
E - 0 = E
1. E - 0 = E
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
E - 0 = E
1. E - 0 <= E
2. E <= E - 0
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
E - 0 = E
1. !!x. x : E - 0 ==> x : E
2. E <= E - 0
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
E - 0 = E
1. !!x. x : E - 0 ==> x : E
2. !!x. x : E ==> x : E - 0
val it = () : unit
> be DiffE 1;
Level 4 (2 subgoals)
E - 0 = E
1. !!x. [| x : E; x ~: 0 |] ==> x : E
2. !!x. x : E ==> x : E - 0
val it = () : unit
> ba 1;
Level 5 (1 subgoal)
E - 0 = E
1. !!x. x : E ==> x : E - 0
val it = () : unit
> br DiffI 1;
Level 6 (2 subgoals)
E - 0 = E
1. !!x. x : E ==> x : E
2. !!x. x : E ==> x ~: 0
val it = () : unit
> ba 1;
Level 7 (1 subgoal)
E - 0 = E
```

```
1. !!x. x : E ==> x ~: 0
val it = () : unit
> br notI 1;
Level 8 (1 subgoal)
E - 0 = E
1. !!x. [| x : E; x : 0 |] ==> False
val it = () : unit
> be emptyE 1;
Level 9
E - 0 = E
No subgoals!
val it = () : unit
```

3. $\vdash_{ZF} E' = \varnothing$

```
> Goal "E-E=0";
Level 0 (1 subgoal)
E - E = 0
1. E - E = 0
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
E - E = 0
1. E - E <= 0
2. 0 <= E - E
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
E - E = 0
1. !!x. x : E - E ==> x : 0
2. 0 <= E - E
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
E - E = 0
1. !!x. x : E - E ==> x : 0
2. !!x. x : 0 ==> x : E - E
val it = () : unit
> be emptyE 2;
Level 4 (1 subgoal)
E - E = 0
1. !!x. x : E - E ==> x : 0
val it = () : unit
> be DiffE 1;
Level 5 (1 subgoal)
E - E = 0
1. !!x. [| x : E; x ~: E |] ==> x : 0
```

```
val it = () : unit
> be notE 1;
Level 6 (1 subgoal)
E - E = 0
1. !!x. x : E ==> x : E
val it = () : unit
> ba 1;
Level 7
E - E = 0
No subgoals!
val it = () : unit
```

4. $\vdash_{ZF} A \cap A' = \varnothing$

```
> Goal "A<=E ==> (A Int (E - A)= 0)";
Level 0 (1 subgoal)
A <= E ==> A Int (E - A) = 0
1. A <= E ==> A Int (E - A) = 0
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A <= E ==> A Int (E - A) = 0
1. A <= E ==> A Int (E - A) <= 0
2. A <= E ==> 0 <= A Int (E - A)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A <= E ==> A Int (E - A) = 0
1. !!x. [| A <= E; x : A Int (E - A) |] ==> x : 0
2. A <= E ==> 0 <= A Int (E - A)
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
A <= E ==> A Int (E - A) = 0
1. !!x. [| A <= E; x : A Int (E - A) |] ==> x : 0
2. !!x. [| A <= E; x : 0 |] ==> x : A Int (E - A)
val it = () : unit
> be IntE 1;
Level 4 (2 subgoals)
A <= E ==> A Int (E - A) = 0
1. !!x. [| A <= E; x : A; x : E - A |] ==> x : 0
2. !!x. [| A <= E; x : 0 |] ==> x : A Int (E - A)
val it = () : unit
> be DiffE 1;
Level 5 (2 subgoals)
A <= E ==> A Int (E - A) = 0
1. !!x. [| A <= E; x : A; x : E; x ~: A |] ==> x : 0
```

```
2. !!x. [| A <= E; x : 0 |] ==> x : A Int (E - A)
val it = () : unit
> be notE 1;
Level 6 (2 subgoals)
A <= E ==> A Int (E - A) = 0
1. !!x. [| A <= E; x : A; x : E |] ==> x : A
2. !!x. [| A <= E; x : 0 |] ==> x : A Int (E - A)
val it = () : unit
> ba 1;
Level 7 (1 subgoal)
A <= E ==> A Int (E - A) = 0
1. !!x. [| A <= E; x : 0 |] ==> x : A Int (E - A)
val it = () : unit
> br IntI 1;
Level 8 (2 subgoals)
A <= E ==> A Int (E - A) = 0
1. !!x. [| A <= E; x : 0 |] ==> x : A
2. !!x. [| A <= E; x : 0 |] ==> x : E - A
val it = () : unit
> be emptyE 1;
Level 9 (1 subgoal)
A <= E ==> A Int (E - A) = 0
1. !!x. [| A <= E; x : 0 |] ==> x : E - A
val it = () : unit
> be emptyE 1;
Level 10
A <= E ==> A Int (E - A) = 0
No subgoals!
val it = () : unit
```

5. $\vdash_{ZF} A \cup A' = E$

```
> Goal "(A<=E)==>A Un (E-A) = E";
Level 0 (1 subgoal)
A <= E ==> A Un (E - A) = E
1. A <= E ==> A Un (E - A) = E
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A <= E ==> A Un (E - A) = E
1. A <= E ==> A Un (E - A) <= E
2. A <= E ==> E <= A Un (E - A)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : A Un (E - A) |] ==> x : E
```

```
2. A <= E ==> E <= A Un (E - A)
val it = () : unit
> be UnE 1;
Level 3 (3 subgoals)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : A |] ==> x : E
2. !!x. [| A <= E; x : E - A |] ==> x : E
3. A <= E ==> E <= A Un (E - A)
val it = () : unit
> be subsetD 1;
Level 4 (3 subgoals)
A <= E ==> A Un (E - A) = E
1. !!x. x : A ==> x : A
2. !!x. [| A <= E; x : E - A |] ==> x : E
3. A <= E ==> E <= A Un (E - A)
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : E - A |] ==> x : E
2. A <= E ==> E <= A Un (E - A)
val it = () : unit
> be DiffE 1;
Level 6 (2 subgoals)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : E; x ~: A |] ==> x : E
2. A <= E ==> E <= A Un (E - A)
val it = () : unit
> ba 1;
Level 7 (1 subgoal)
A <= E ==> A Un (E - A) = E
1. A <= E ==> E <= A Un (E - A)
val it = () : unit
> br subsetI 1;
Level 8 (1 subgoal)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : E |] ==> x : A Un (E - A)
val it = () : unit
> br UnCI 1;
Level 9 (1 subgoal)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : E; x ~: E - A |] ==> x : A
val it = () : unit
> be swap 1;
Level 10 (1 subgoal)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : E; x ~: A |] ==> x : E - A
```

```
val it = () : unit
> br DiffI 1;
Level 11 (2 subgoals)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : E; x ~: A |] ==> x : E
2. !!x. [| A <= E; x : E; x ~: A |] ==> x ~: A
val it = () : unit
> ba 1;
Level 12 (1 subgoal)
A <= E ==> A Un (E - A) = E
1. !!x. [| A <= E; x : E; x ~: A |] ==> x ~: A
val it = () : unit
> ba 1;
Level 13
A <= E ==> A Un (E - A) = E
No subgoals!
val it = () : unit
```

7. $\vdash_{ZF} (A \cup B)' = A' \cap B'$

```
> Goal "[|A<=E;B<=E|]==>(E- (A Un B))=((E - A) Int (E - B))";
Level 0 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. [| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. [| A <= E; B <= E |] ==> E - (A Un B) <= (E - A) Int (E - B)
2. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : (E - A) Int (E - B)
2. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br IntI 1;
Level 3 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - A
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
3. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br DiffI 1;
Level 4 (4 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
```

```
1. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x ~: A
3. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
4. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> be DiffE 1;
Level 5 (4 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E; x ~: A Un B |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x ~: A
3. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
4. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> ba 1;
Level 6 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x ~: A
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
3. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> be DiffE 1;
Level 7 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E; x ~: A Un B |] ==> x ~: A
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
3. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br notI 1;
Level 8 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E; x ~: A Un B; x : A |] ==> False
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
3. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> be notE 1;
Level 9 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E; x : A |] ==> x : A Un B
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
3. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br UnI1 1;
Level 10 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E; x : A |] ==> x : A
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
3. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
```

```
val it = () : unit
> ba 1;
Level 11 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E - B
2. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br DiffI 1;
Level 12 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x ~: B
3. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> be DiffE 1;
Level 13 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E; x ~: A Un B |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x ~: B
3. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> ba 1;
Level 14 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - (A Un B) |] ==> x ~: B
2. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br notI 1;
Level 15 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - (A Un B); x : B |] ==> False
2. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> be DiffE 1;
Level 16 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : B; x : E; x ~: A Un B |] ==> False
2. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> be notE 1;
Level 17 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : B; x : E |] ==> x : A Un B
2. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br UnI2 1;
Level 18 (2 subgoals)
```

```
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : B; x : E |] ==> x : B
2. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> ba 1;
Level 19 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. [| A <= E; B <= E |] ==> (E - A) Int (E - B) <= E - (A Un B)
val it = () : unit
> br subsetI 1;
Level 20 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B) |] ==> x : E - (A Un B)
val it = () : unit
> br DiffI 1;
Level 21 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B) |] ==> x : E
2. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B) |] ==> x ~: A Un B
val it = () : unit
> be IntE 1;
Level 22 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - A; x : E - B |] ==> x : E
2. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B) |] ==> x ~: A Un B
val it = () : unit
> be DiffE 1;
Level 23 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : E - B; x : E; x ~: A |] ==> x : E
2. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B) |] ==> x ~: A Un B
val it = () : unit
> ba 1;
Level 24 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B) |] ==> x ~: A Un B
val it = () : unit
> br notI 1;
Level 25 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B); x : A Un B |] ==> False
val it = () : unit
> be UnE 1;
Level 26 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B); x : A |] ==> False
2. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B); x : B |] ==> False
```

```
val it = () : unit
> be IntE 1;
Level 27 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : A; x : E - A; x : E - B |] ==> False
2. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B); x : B |] ==> False
val it = () : unit
> be DiffE 1;
Level 28 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : A; x : E - B; x : E; x ~: A |] ==> False
2. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B); x : B |] ==> False
val it = () : unit
> br notE 1;
Level 29 (3 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : A; x : E - B; x : E; x ~: A |] ==> ~?P23(x)
2. !!x. [| A <= E; B <= E; x : A; x : E - B; x : E; x ~: A |] ==> ?P23(x)
3. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B); x : B |] ==> False
val it = () : unit
> ba 1;
Level 30 (2 subgoals)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : A; x : E - B; x : E; x ~: A |] ==> x : A
2. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B); x : B |] ==> False
val it = () : unit
> ba 1;
Level 31 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : (E - A) Int (E - B); x : B |] ==> False
val it = () : unit
> be IntE 1;
Level 32 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : B; x : E - A; x : E - B |] ==> False
val it = () : unit
> by (rotate_tac 4 1);
Level 33 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| x : E - B; A <= E; B <= E; x : B; x : E - A |] ==> False
val it = () : unit
> be DiffE 1;
Level 34 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : B; x : E - A; x : E; x ~: B |] ==> False
val it = () : unit
> be notE 1;
```

```
Level 35 (1 subgoal)
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
1. !!x. [| A <= E; B <= E; x : B; x : E - A; x : E |] ==> x : B
val it = () : unit
> ba 1;
Level 36
[| A <= E; B <= E |] ==> E - (A Un B) = (E - A) Int (E - B)
No subgoals!
val it = () : unit
```

8. $\vdash_{ZF} (A \cap B)' = A' \cup B'$

```
> Goal "[|A<=E;B<=E|]==>(E -(A Int B))=((E - A)Un(E - B))";
Level 0 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. [| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. [| A <= E; B <= E |] ==> E - A Int B <= E - A Un (E - B)
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B |] ==> x : E - A Un (E - B)
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br classical 1;
Level 3 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B) |]
==> x : E - A Un (E - B)
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br UnI2 1;
Level 4 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B) |]
==> x : E - B
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br DiffI 1;
Level 5 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B) |]
```

```
==> x : E
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B) |]
==> x ~: B
3. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> be DiffD1 1;
Level 6 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B) |]
==> x ~: B
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br notI 1;
Level 7 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> False
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br notE 1;
Level 8 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> ~?P7(x)
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> ?P7(x)
3. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> ba 1;
Level 9 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> x : E - A Un (E - B)
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br UnI1 1;
Level 10 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> x : E - A
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br DiffI 1;
Level 11 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> x : E
```

```
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> x ~: A
3. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br DiffD1 1;
Level 12 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> x : E - ?B10(x)
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> x ~: A
3. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> ba 1;
Level 13 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B |]
==> x ~: A
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br notI 1;
Level 14 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> False
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br notE 1;
Level 15 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> ~?P12(x)
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> ?P12(x)
3. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br IntI 2;
Level 16 (4 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> ?c13(x) ~: ?A13(x) Int ?B13(x)
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
```

```
==> ?c13(x) : ?A13(x)
3. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> ?c13(x) : ?B13(x)
4. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> ba 2;
Level 17 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> x ~: (E - A Int B) Int ?B13(x)
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> x : ?B13(x)
3. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> back();
Level 17 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> x ~: B Int ?B13(x)
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> x : ?B13(x)
3. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> back();
Level 17 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> x ~: A Int ?B13(x)
2. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> x : ?B13(x)
3. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> ba 2;
Level 18 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> x ~: A Int (E - A Int B)
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
```

```
> back();
Level 18 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Int B; x ~: E - A Un (E - B); x : B;
x : A |]
==> x ~: A Int B
2. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> be DiffD2 1;
Level 19 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. [| A <= E; B <= E |] ==> E - A Un (E - B) <= E - A Int B
val it = () : unit
> br subsetI 1;
Level 20 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Un (E - B) |] ==> x : E - A Int B
val it = () : unit
> br DiffI 1;
Level 21 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Un (E - B) |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - A Un (E - B) |] ==> x ~: A Int B
val it = () : unit
> be UnE 1;
Level 22 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - B |] ==> x : E
3. !!x. [| A <= E; B <= E; x : E - A Un (E - B) |] ==> x ~: A Int B
val it = () : unit
> be DiffE 1;
Level 23 (3 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E; x ~: A |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - B |] ==> x : E
3. !!x. [| A <= E; B <= E; x : E - A Un (E - B) |] ==> x ~: A Int B
val it = () : unit
> ba 1;
Level 24 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - B |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - A Un (E - B) |] ==> x ~: A Int B
val it = () : unit
> be DiffE 1;
Level 25 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
```

```
1. !!x. [| A <= E; B <= E; x : E; x ~: B |] ==> x : E
2. !!x. [| A <= E; B <= E; x : E - A Un (E - B) |] ==> x ~: A Int B
val it = () : unit
> ba 1;
Level 26 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Un (E - B) |] ==> x ~: A Int B
val it = () : unit
> br notI 1;
Level 27 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A Un (E - B); x : A Int B |] ==> False
val it = () : unit
> be UnE 1;
Level 28 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : A Int B; x : E - A |] ==> False
2. !!x. [| A <= E; B <= E; x : A Int B; x : E - B |] ==> False
val it = () : unit
> be IntE 1;
Level 29 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - A; x : A; x : B |] ==> False
2. !!x. [| A <= E; B <= E; x : A Int B; x : E - B |] ==> False
val it = () : unit
> be DiffE 1;
Level 30 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : A; x : B; x : E; x ~: A |] ==> False
2. !!x. [| A <= E; B <= E; x : A Int B; x : E - B |] ==> False
val it = () : unit
> be notE 1;
Level 31 (2 subgoals)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : A; x : B; x : E |] ==> x : A
2. !!x. [| A <= E; B <= E; x : A Int B; x : E - B |] ==> False
val it = () : unit
> ba 1;
Level 32 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : A Int B; x : E - B |] ==> False
val it = () : unit
> be IntE 1;
Level 33 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : E - B; x : A; x : B |] ==> False
val it = () : unit
```

```
> be DiffE 1;
Level 34 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : A; x : B; x : E; x ~: B |] ==> False
val it = () : unit
> be notE 1;
Level 35 (1 subgoal)
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
1. !!x. [| A <= E; B <= E; x : A; x : B; x : E |] ==> x : B
val it = () : unit
> ba 1;
Level 36
[| A <= E; B <= E |] ==> E - A Int B = E - A Un (E - B)
No subgoals!
val it = () : unit
```

9. $\vdash_{ZF} A - (B \cup C) = (A - B) \cap (A - C)$

```
> Goal "A-(B Un C)=(A-B)Int(A-C)";
Level 0 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. A - (B Un C) = (A - B) Int (A - C)
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. A - (B Un C) <= (A - B) Int (A - C)
2. (A - B) Int (A - C) <= A - (B Un C)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : A - (B Un C) ==> x : (A - B) Int (A - C)
2. (A - B) Int (A - C) <= A - (B Un C)
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : A - (B Un C) ==> x : (A - B) Int (A - C)
2. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> br IntI 1;
Level 4 (3 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : A - (B Un C) ==> x : A - B
2. !!x. x : A - (B Un C) ==> x : A - C
3. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
```

```
val it = () : unit
> br DiffI 1;
Level 5 (4 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : A - (B Un C) ==> x : A
2. !!x. x : A - (B Un C) ==> x ~: B
3. !!x. x : A - (B Un C) ==> x : A - C
4. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> be DiffD1 1;
Level 6 (3 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : A - (B Un C) ==> x ~: B
2. !!x. x : A - (B Un C) ==> x : A - C
3. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> be DiffE 1;
Level 7 (3 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A; x ~: B Un C |] ==> x ~: B
2. !!x. x : A - (B Un C) ==> x : A - C
3. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> br notI 1;
Level 8 (3 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A; x ~: B Un C; x : B |] ==> False
2. !!x. x : A - (B Un C) ==> x : A - C
3. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> be notE 1;
Level 9 (3 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A; x : B |] ==> x : B Un C
2. !!x. x : A - (B Un C) ==> x : A - C
3. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> br UnI1 1;
Level 10 (3 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A; x : B |] ==> x : B
2. !!x. x : A - (B Un C) ==> x : A - C
3. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> ba 1;
Level 11 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
```

```
1. !!x. x : A - (B Un C) ==> x : A - C
2. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> br DiffI 1;
Level 12 (3 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : A - (B Un C) ==> x : A
2. !!x. x : A - (B Un C) ==> x ~: C
3. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> be DiffD1 1;
Level 13 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : A - (B Un C) ==> x ~: C
2. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> be DiffE 1;
Level 14 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A; x ~: B Un C |] ==> x ~: C
2. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> br notI 1;
Level 15 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A; x ~: B Un C; x : C |] ==> False
2. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> be notE 1;
Level 16 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A; x : C |] ==> x : B Un C
2. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> br UnI2 1;
Level 17 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A; x : C |] ==> x : C
2. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> ba 1;
Level 18 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : (A - B) Int (A - C) ==> x : A - (B Un C)
val it = () : unit
> br DiffI 1;
Level 19 (2 subgoals)
```

```
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : (A - B) Int (A - C) ==> x : A
2. !!x. x : (A - B) Int (A - C) ==> x ~: B Un C
val it = () : unit
> be IntE 1;
Level 20 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A - B; x : A - C |] ==> x : A
2. !!x. x : (A - B) Int (A - C) ==> x ~: B Un C
val it = () : unit
> be DiffD1 1;
Level 21 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. x : (A - B) Int (A - C) ==> x ~: B Un C
val it = () : unit
> be IntE 1;
Level 22 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A - B; x : A - C |] ==> x ~: B Un C
val it = () : unit
> br notI 1;
Level 23 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A - B; x : A - C; x : B Un C |] ==> False
val it = () : unit
> be UnE 1;
Level 24 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A - B; x : A - C; x : B |] ==> False
2. !!x. [| x : A - B; x : A - C; x : C |] ==> False
val it = () : unit
> be DiffE 1;
Level 25 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A - C; x : B; x : A; x ~: B |] ==> False
2. !!x. [| x : A - B; x : A - C; x : C |] ==> False
val it = () : unit
> be notE 1;
Level 26 (2 subgoals)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A - C; x : B; x : A |] ==> x : B
2. !!x. [| x : A - B; x : A - C; x : C |] ==> False
val it = () : unit
> ba 1;
Level 27 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A - B; x : A - C; x : C |] ==> False
```

```
val it = () : unit
> by (rotate_tac 1 1);
Level 28 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : A - C; x : C; x : A - B |] ==> False
val it = () : unit
> be DiffE 1;
Level 29 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : C; x : A - B; x : A; x ~: C |] ==> False
val it = () : unit
> be notE 1;
Level 30 (1 subgoal)
A - (B Un C) = (A - B) Int (A - C)
1. !!x. [| x : C; x : A - B; x : A |] ==> x : C
val it = () : unit
> ba 1;
Level 31
A - (B Un C) = (A - B) Int (A - C)
No subgoals!
val it = () : unit
```

10. $\vdash_{ZF} A - (B \cap C) = (A - B) \cup (A - C)$

```
> Goal "(A -(B Int C)) = ((A - B) Un (A - C))";
Level 0 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. A - B Int C = A - B Un (A - C)
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. A - B Int C <= A - B Un (A - C)
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. x : A - B Int C ==> x : A - B Un (A - C)
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br classical 1;
Level 3 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x ~: A - B Un (A - C) |]
==> x : A - B Un (A - C)
2. A - B Un (A - C) <= A - B Int C
```

```
val it = () : unit
> br UnI2 1;
Level 4 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x ~: A - B Un (A - C) |] ==> x : A - C
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br DiffI 1;
Level 5 (3 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x ~: A - B Un (A - C) |] ==> x : A
2. !!x. [| x : A - B Int C; x ~: A - B Un (A - C) |] ==> x ~: C
3. A - B Un (A - C) <= A - B Int C
val it = () : unit
> be DiffD1 1;
Level 6 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x ~: A - B Un (A - C) |] ==> x ~: C
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br notI 1;
Level 7 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x ~: A - B Un (A - C); x : C |] ==> False
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> be notE 1;
Level 8 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x : C |] ==> x : A - B Un (A - C)
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br UnI1 1;
Level 9 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x : C |] ==> x : A - B
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br DiffI 1;
Level 10 (3 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x : C |] ==> x : A
2. !!x. [| x : A - B Int C; x : C |] ==> x ~: B
3. A - B Un (A - C) <= A - B Int C
val it = () : unit
> be DiffD1 1;
Level 11 (2 subgoals)
```

```
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x : C |] ==> x ~: B
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br notI 1;
Level 12 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Int C; x : C; x : B |] ==> False
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> be DiffE 1;
Level 13 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : C; x : B; x : A; x ~: B Int C |] ==> False
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> be notE 1;
Level 14 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : C; x : B; x : A |] ==> x : B Int C
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br IntI 1;
Level 15 (3 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : C; x : B; x : A |] ==> x : B
2. !!x. [| x : C; x : B; x : A |] ==> x : C
3. A - B Un (A - C) <= A - B Int C
val it = () : unit
> ba 1;
Level 16 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : C; x : B; x : A |] ==> x : C
2. A - B Un (A - C) <= A - B Int C
val it = () : unit
> ba 1;
Level 17 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. A - B Un (A - C) <= A - B Int C
val it = () : unit
> br subsetI 1;
Level 18 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. !!x. x : A - B Un (A - C) ==> x : A - B Int C
val it = () : unit
> br DiffI 1;
Level 19 (2 subgoals)
```

```
A - B Int C = A - B Un (A - C)
1. !!x. x : A - B Un (A - C) ==> x : A
2. !!x. x : A - B Un (A - C) ==> x ~: B Int C
val it = () : unit
> be UnE 1;
Level 20 (3 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. x : A - B ==> x : A
2. !!x. x : A - C ==> x : A
3. !!x. x : A - B Un (A - C) ==> x ~: B Int C
val it = () : unit
> be DiffD1 1;
Level 21 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. x : A - C ==> x : A
2. !!x. x : A - B Un (A - C) ==> x ~: B Int C
val it = () : unit
> be DiffD1 1;
Level 22 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. !!x. x : A - B Un (A - C) ==> x ~: B Int C
val it = () : unit
> br notI 1;
Level 23 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B Un (A - C); x : B Int C |] ==> False
val it = () : unit
> be UnE 1;
Level 24 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : B Int C; x : A - B |] ==> False
2. !!x. [| x : B Int C; x : A - C |] ==> False
val it = () : unit
> be IntE 1;
Level 25 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - B; x : B; x : C |] ==> False
2. !!x. [| x : B Int C; x : A - C |] ==> False
val it = () : unit
> be DiffE 1;
Level 26 (2 subgoals)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : B; x : C; x : A; x ~: B |] ==> False
2. !!x. [| x : B Int C; x : A - C |] ==> False
val it = () : unit
> be notE 1;
Level 27 (2 subgoals)
```

```
A - B Int C = A - B Un (A - C)
1. !!x. [| x : B; x : C; x : A |] ==> x : B
2. !!x. [| x : B Int C; x : A - C |] ==> False
val it = () : unit
> ba 1;
Level 28 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : B Int C; x : A - C |] ==> False
val it = () : unit
> be IntE 1;
Level 29 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : A - C; x : B; x : C |] ==> False
val it = () : unit
> be DiffE 1;
Level 30 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : B; x : C; x : A; x ~: C |] ==> False
val it = () : unit
> be notE 1;
Level 31 (1 subgoal)
A - B Int C = A - B Un (A - C)
1. !!x. [| x : B; x : C; x : A |] ==> x : C
val it = () : unit
> ba 1;
Level 32
A - B Int C = A - B Un (A - C)
No subgoals!
val it = () : unit
```

11. $\vdash_{ZF} A - B = A \cap B'$

```
> Goal "[|A<=E;B<=E|]==> A - B = A Int (E - B)";
Level 0 (1 subgoal)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. [| A <= E; B <= E |] ==> A - B = A Int (E - B)
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. [| A <= E; B <= E |] ==> A - B <= A Int (E - B)
2. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A - B |] ==> x : A Int (E - B)
```

```
2. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> br IntI 1;
Level 3 (3 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A - B |] ==> x : A
2. !!x. [| A <= E; B <= E; x : A - B |] ==> x : E - B
3. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> be DiffE 1;
Level 4 (3 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A; x ~: B |] ==> x : A
2. !!x. [| A <= E; B <= E; x : A - B |] ==> x : E - B
3. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A - B |] ==> x : E - B
2. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> br DiffI 1;
Level 6 (3 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A - B |] ==> x : E
2. !!x. [| A <= E; B <= E; x : A - B |] ==> x ~: B
3. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> be DiffE 1;
Level 7 (3 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A; x ~: B |] ==> x : E
2. !!x. [| A <= E; B <= E; x : A - B |] ==> x ~: B
3. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> be subsetD 1;
Level 8 (3 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| B <= E; x : A; x ~: B |] ==> x : A
2. !!x. [| A <= E; B <= E; x : A - B |] ==> x ~: B
3. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> ba 1;
Level 9 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A - B |] ==> x ~: B
```

```
2. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> br notI 1;
Level 10 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A - B; x : B |] ==> False
2. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> be DiffE 1;
Level 11 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : B; x : A; x ~: B |] ==> False
2. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> be notE 1;
Level 12 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : B; x : A |] ==> x : B
2. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> ba 1;
Level 13 (1 subgoal)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. [| A <= E; B <= E |] ==> A Int (E - B) <= A - B
val it = () : unit
> br subsetI 1;
Level 14 (1 subgoal)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A Int (E - B) |] ==> x : A - B
val it = () : unit
> br DiffI 1;
Level 15 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A Int (E - B) |] ==> x : A
2. !!x. [| A <= E; B <= E; x : A Int (E - B) |] ==> x ~: B
val it = () : unit
> be IntD1 1;
Level 16 (1 subgoal)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A Int (E - B) |] ==> x ~: B
val it = () : unit
> br notI 1;
Level 17 (1 subgoal)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A Int (E - B); x : B |] ==> False
val it = () : unit
> br IntE 1;
```

```
Level 18 (2 subgoals)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A Int (E - B); x : B |]
==> ?c14(x) : ?A14(x) Int ?B14(x)
2. !!x. [| A <= E; B <= E; x : A Int (E - B); x : B; ?c14(x) : ?A14(x);
?c14(x) : ?B14(x) |]
==> False
val it = () : unit
> ba 1;
Level 19 (1 subgoal)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A Int (E - B); x : B; x : A; x : E - B |]
==> False
val it = () : unit
> be DiffE 1;
Level 20 (1 subgoal)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A Int (E - B); x : B; x : A; x : E;
x ~: B |]
==> False
val it = () : unit
> be notE 1;
Level 21 (1 subgoal)
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
1. !!x. [| A <= E; B <= E; x : A Int (E - B); x : B; x : A; x : E |]
==> x : B
val it = () : unit
> ba 1;
Level 22
[| A <= E; B <= E |] ==> A - B = A Int (E - B)
No subgoals!
val it = () : unit
```

12. $\vdash_{ZF} A \subseteq B \Leftrightarrow A - B = \varnothing$

```
> Goal "A<=B <-> A - B = 0";
Level 0 (1 subgoal)
A <= B <-> A - B = 0
1. A <= B <-> A - B = 0
val it = [] : Thm.thm list
> br iffI 1;
Level 1 (2 subgoals)
A <= B <-> A - B = 0
1. A <= B ==> A - B = 0
2. A - B = 0 ==> A <= B
val it = () : unit
> br equalityI 1;
```

```
Level 2 (3 subgoals)
A <= B <-> A - B = 0
1. A <= B ==> A - B <= 0
2. A <= B ==> 0 <= A - B
3. A - B = 0 ==> A <= B
val it = () : unit
> br subsetI 1;
Level 3 (3 subgoals)
A <= B <-> A - B = 0
1. !!x. [| A <= B; x : A - B |] ==> x : 0
2. A <= B ==> 0 <= A - B
3. A - B = 0 ==> A <= B
val it = () : unit
> be DiffE 1;
Level 4 (3 subgoals)
A <= B <-> A - B = 0
1. !!x. [| A <= B; x : A; x ~: B |] ==> x : 0
2. A <= B ==> 0 <= A - B
3. A - B = 0 ==> A <= B
val it = () : unit
> be notE 1;
Level 5 (3 subgoals)
A <= B <-> A - B = 0
1. !!x. [| A <= B; x : A |] ==> x : B
2. A <= B ==> 0 <= A - B
3. A - B = 0 ==> A <= B
val it = () : unit
> be subsetD 1;
Level 6 (3 subgoals)
A <= B <-> A - B = 0
1. !!x. x : A ==> x : A
2. A <= B ==> 0 <= A - B
3. A - B = 0 ==> A <= B
val it = () : unit
> ba 1;
Level 7 (2 subgoals)
A <= B <-> A - B = 0
1. A <= B ==> 0 <= A - B
2. A - B = 0 ==> A <= B
val it = () : unit
> br subsetI 1;
Level 8 (2 subgoals)
A <= B <-> A - B = 0
1. !!x. [| A <= B; x : 0 |] ==> x : A - B
2. A - B = 0 ==> A <= B
val it = () : unit
> be emptyE 1;
```

```
Level 9 (1 subgoal)
A <= B <-> A - B = 0
1. A - B = 0 ==> A <= B
val it = () : unit
> br subsetI 1;
Level 10 (1 subgoal)
A <= B <-> A - B = 0
1. !!x. [| A - B = 0; x : A |] ==> x : B
val it = () : unit
> br classical 1;
Level 11 (1 subgoal)
A <= B <-> A - B = 0
1. !!x. [| A - B = 0; x : A; x ~: B |] ==> x : B
val it = () : unit
> br emptyE 1;
Level 12 (1 subgoal)
A <= B <-> A - B = 0
1. !!x. [| A - B = 0; x : A; x ~: B |] ==> ?a10(x) : 0
val it = () : unit
> br subsetD 1;
Level 13 (2 subgoals)
A <= B <-> A - B = 0
1. !!x. [| A - B = 0; x : A; x ~: B |] ==> ?A11(x) <= 0
2. !!x. [| A - B = 0; x : A; x ~: B |] ==> ?a10(x) : ?A11(x)
val it = () : unit
> be equalityD1 1;
Level 14 (1 subgoal)
A <= B <-> A - B = 0
1. !!x. [| A - B = 0; x : A; x ~: B |] ==> ?a10(x) : A - B
val it = () : unit
> br DiffI 1;
Level 15 (2 subgoals)
A <= B <-> A - B = 0
1. !!x. [| A - B = 0; x : A; x ~: B |] ==> ?a10(x) : A
2. !!x. [| A - B = 0; x : A; x ~: B |] ==> ?a10(x) ~: B
val it = () : unit
> ba 1;
Level 16 (1 subgoal)
A <= B <-> A - B = 0
1. !!x. [| A - B = 0; x : A; x ~: B |] ==> x ~: B
val it = () : unit
> ba 1;
Level 17
A <= B <-> A - B = 0
No subgoals!
val it = () : unit
```

13. $\vdash_{ZF} A - (A - B) = A \cap B$

```
> Goal "A - (A - B) = A Int B";
Level 0 (1 subgoal)
A - (A - B) = A Int B
1. A - (A - B) = A Int B
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A - (A - B) = A Int B
1. A - (A - B) <= A Int B
2. A Int B <= A - (A - B)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A - (A - B) = A Int B
1. !!x. x : A - (A - B) ==> x : A Int B
2. A Int B <= A - (A - B)
val it = () : unit
> br IntI 1;
Level 3 (3 subgoals)
A - (A - B) = A Int B
1. !!x. x : A - (A - B) ==> x : A
2. !!x. x : A - (A - B) ==> x : B
3. A Int B <= A - (A - B)
val it = () : unit
> br DiffD1 1;
Level 4 (3 subgoals)
A - (A - B) = A Int B
1. !!x. x : A - (A - B) ==> x : A - ?B3(x)
2. !!x. x : A - (A - B) ==> x : B
3. A Int B <= A - (A - B)
val it = () : unit
> ba 1;
Level 5 (2 subgoals)
A - (A - B) = A Int B
1. !!x. x : A - (A - B) ==> x : B
2. A Int B <= A - (A - B)
val it = () : unit
> br classical 1;
Level 6 (2 subgoals)
A - (A - B) = A Int B
1. !!x. [| x : A - (A - B); x ~: B |] ==> x : B
2. A Int B <= A - (A - B)
val it = () : unit
> be DiffE 1;
Level 7 (2 subgoals)
```

```
A - (A - B) = A Int B
1. !!x. [| x ~: B; x : A; x ~: A - B |] ==> x : B
2. A Int B <= A - (A - B)
val it = () : unit
> be notE 1;
Level 8 (2 subgoals)
A - (A - B) = A Int B
1. !!x. [| x : A; x ~: A - B |] ==> x : B
2. A Int B <= A - (A - B)
val it = () : unit
> back();
Level 8 (2 subgoals)
A - (A - B) = A Int B
1. !!x. [| x ~: B; x : A |] ==> x : A - B
2. A Int B <= A - (A - B)
val it = () : unit
> br DiffI 1;
Level 9 (3 subgoals)
A - (A - B) = A Int B
1. !!x. [| x ~: B; x : A |] ==> x : A
2. !!x. [| x ~: B; x : A |] ==> x ~: B
3. A Int B <= A - (A - B)
val it = () : unit
> ba 1;
Level 10 (2 subgoals)
A - (A - B) = A Int B
1. !!x. [| x ~: B; x : A |] ==> x ~: B
2. A Int B <= A - (A - B)
val it = () : unit
> ba 1;
Level 11 (1 subgoal)
A - (A - B) = A Int B
1. A Int B <= A - (A - B)
val it = () : unit
> br subsetI 1;
Level 12 (1 subgoal)
A - (A - B) = A Int B
1. !!x. x : A Int B ==> x : A - (A - B)
val it = () : unit
> br DiffI 1;
Level 13 (2 subgoals)
A - (A - B) = A Int B
1. !!x. x : A Int B ==> x : A
2. !!x. x : A Int B ==> x ~: A - B
val it = () : unit
> be IntD1 1;
Level 14 (1 subgoal)
```

```
A - (A - B) = A Int B
1. !!x. x : A Int B ==> x ~: A - B
val it = () : unit
> br notI 1;
Level 15 (1 subgoal)
A - (A - B) = A Int B
1. !!x. [| x : A Int B; x : A - B |] ==> False
val it = () : unit
> be IntE 1;
Level 16 (1 subgoal)
A - (A - B) = A Int B
1. !!x. [| x : A - B; x : A; x : B |] ==> False
val it = () : unit
> be DiffE 1;
Level 17 (1 subgoal)
A - (A - B) = A Int B
1. !!x. [| x : A; x : B; x : A; x ~: B |] ==> False
val it = () : unit
> be notE 1;
Level 18 (1 subgoal)
A - (A - B) = A Int B
1. !!x. [| x : A; x : B; x : A |] ==> x : B
val it = () : unit
> ba 1;
Level 19
A - (A - B) = A Int B
No subgoals!
val it = () : unit
```

14. $\vdash_{ZF} A \cap (B - C) = (A \cap B) - (A \cap C)$

```
> Goal "A Int(B - C)=(A Int B) - (A Int C)";
Level 0 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. A Int (B - C) = A Int B - A Int C
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. A Int (B - C) <= A Int B - A Int C
2. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int (B - C) ==> x : A Int B - A Int C
2. A Int B - A Int C <= A Int (B - C)
```

```
val it = () : unit
> br DiffI 1;
Level 3 (3 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int (B - C) ==> x : A Int B
2. !!x. x : A Int (B - C) ==> x ~: A Int C
3. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> br IntI 1;
Level 4 (4 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int (B - C) ==> x : A
2. !!x. x : A Int (B - C) ==> x : B
3. !!x. x : A Int (B - C) ==> x ~: A Int C
4. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> be IntD1 1 ;
Level 5 (3 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int (B - C) ==> x : B
2. !!x. x : A Int (B - C) ==> x ~: A Int C
3. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> be IntE 1;
Level 6 (3 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A; x : B - C |] ==> x : B
2. !!x. x : A Int (B - C) ==> x ~: A Int C
3. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> be DiffE 1;
Level 7 (3 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A; x : B; x ~: C |] ==> x : B
2. !!x. x : A Int (B - C) ==> x ~: A Int C
3. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> ba 1;
Level 8 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int (B - C) ==> x ~: A Int C
2. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> br notI 1;
Level 9 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A Int (B - C); x : A Int C |] ==> False
```

```
2. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> be IntE 1;
Level 10 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A Int C; x : A; x : B - C |] ==> False
2. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> be DiffE 1;
Level 11 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A Int C; x : A; x : B; x ~: C |] ==> False
2. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> be IntE 1;
Level 12 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A; x : B; x ~: C; x : A; x : C |] ==> False
2. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> be notE 1;
Level 13 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A; x : B; x : A; x : C |] ==> x : C
2. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> ba 1;
Level 14 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. A Int B - A Int C <= A Int (B - C)
val it = () : unit
> br subsetI 1;
Level 15 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int B - A Int C ==> x : A Int (B - C)
val it = () : unit
> br IntI 1;
Level 16 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int B - A Int C ==> x : A
2. !!x. x : A Int B - A Int C ==> x : B - C
val it = () : unit
> be DiffE 1;
Level 17 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A Int B; x ~: A Int C |] ==> x : A
2. !!x. x : A Int B - A Int C ==> x : B - C
```

```
val it = () : unit
> be IntD1 1;
Level 18 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int B - A Int C ==> x : B - C
val it = () : unit
> br DiffI 1;
Level 19 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int B - A Int C ==> x : B
2. !!x. x : A Int B - A Int C ==> x ~: C
val it = () : unit
> be DiffE 1;
Level 20 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A Int B; x ~: A Int C |] ==> x : B
2. !!x. x : A Int B - A Int C ==> x ~: C
val it = () : unit
> be IntD2 1;
Level 21 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. !!x. x : A Int B - A Int C ==> x ~: C
val it = () : unit
> br notI 1;
Level 22 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : A Int B - A Int C; x : C |] ==> False
val it = () : unit
> be DiffE 1;
Level 23 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : C; x : A Int B; x ~: A Int C |] ==> False
val it = () : unit
> be notE 1;
Level 24 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : C; x : A Int B |] ==> x : A Int C
val it = () : unit
> br IntI 1;
Level 25 (2 subgoals)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : C; x : A Int B |] ==> x : A
2. !!x. [| x : C; x : A Int B |] ==> x : C
val it = () : unit
> be IntE 1;
Level 26 (2 subgoals)
A Int (B - C) = A Int B - A Int C
```

```
1. !!x. [| x : C; x : A; x : B |] ==> x : A
2. !!x. [| x : C; x : A Int B |] ==> x : C
val it = () : unit
> ba 1;
Level 27 (1 subgoal)
A Int (B - C) = A Int B - A Int C
1. !!x. [| x : C; x : A Int B |] ==> x : C
val it = () : unit
> ba 1;
Level 28
A Int (B - C) = A Int B - A Int C
No subgoals!
val it = () : unit
```

15. $\vdash_{ZF} A \cap B \subseteq (A \cap C) \cup (B \cap C')$

```
> Goal "[|A<=E;B<=E;C<=E|]==>(A Int B)<=((A Int C)Un (B Int (E-
C)))";
Level 0 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. [| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
val it = [] : Thm.thm list
> br subsetI 1;
Level 1 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B |]
==> x : A Int C Un B Int (E - C)
val it = () : unit
> br classical 1;
Level 2 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x : A Int C Un B Int (E - C)
val it = () : unit
> br UnI2 1;
Level 3 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x : B Int (E - C)
val it = () : unit
> br IntI 1;
Level 4 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
```

```
==> x : B
2. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x : E - C
val it = () : unit
> be IntD2 1;
Level 5 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x : E - C
val it = () : unit
> br DiffI 1;
Level 6 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x : E
2. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x ~: C
val it = () : unit
> be IntE 1;
Level 7 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x ~: A Int C Un B Int (E - C); x : A;
x : B |]
==> x : E
2. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x ~: C
val it = () : unit
> be subsetD 1;
Level 8 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| B <= E; C <= E; x ~: A Int C Un B Int (E - C); x : A; x : B |]
==> x : A
2. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x ~: C
val it = () : unit
> ba 1;
Level 9 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C) |]
==> x ~: C
```

```
val it = () : unit
> br notI 1;
Level 10 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B;
x ~: A Int C Un B Int (E - C); x : C |]
==> False
val it = () : unit
> be notE 1;
Level 11 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B; x : C |]
==> x : A Int C Un B Int (E - C)
val it = () : unit
> br UnI1 1;
Level 12 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B; x : C |] ==> x : A Int C
val it = () : unit
> br IntI 1;
Level 13 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B; x : C |] ==> x : A
2. !!x. [| A <= E; B <= E; C <= E; x : A Int B; x : C |] ==> x : C
val it = () : unit
> be IntE 1;
Level 14 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : C; x : A; x : B |] ==> x : A
2. !!x. [| A <= E; B <= E; C <= E; x : A Int B; x : C |] ==> x : C
val it = () : unit
> ba 1;
Level 15 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
1. !!x. [| A <= E; B <= E; C <= E; x : A Int B; x : C |] ==> x : C
val it = () : unit
> ba 1;
Level 16
[| A <= E; B <= E; C <= E |] ==> A Int B <= A Int C Un B Int (E - C)
No subgoals!
val it = () : unit
```

16. $\vdash_{ZF} (A \cup C) \cap (B \cup C') \subseteq A \cup B$

```
> Goal "[|A<=E;B<=E;C<=E|]==>((A Un C)Int(B Un (E-C)))<=(A Un B)";
Level 0 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
```

```
1. [| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A
Un B
val it = [] : Thm.thm list
> br subsetI 1;
Level 1 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : (A Un C) Int (B Un (E - C)) |]
==> x : A Un B
val it = () : unit
> be IntE 1;
Level 2 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : A Un C; x : B Un (E - C) |]
==> x : A Un B
val it = () : unit
> be UnE 1;
Level 3 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : B Un (E - C); x : A |]
==> x : A Un B
2. !!x. [| A <= E; B <= E; C <= E; x : B Un (E - C); x : C |]
==> x : A Un B
val it = () : unit
> br UnI1 1;
Level 4 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : B Un (E - C); x : A |] ==> x : A
2. !!x. [| A <= E; B <= E; C <= E; x : B Un (E - C); x : C |]
==> x : A Un B
val it = () : unit
> ba 1;
Level 5 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : B Un (E - C); x : C |]
==> x : A Un B
val it = () : unit
> be UnE 1;
Level 6 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : C; x : B |] ==> x : A Un B
2. !!x. [| A <= E; B <= E; C <= E; x : C; x : E - C |] ==> x : A Un B
val it = () : unit
> br UnI2 1;
Level 7 (2 subgoals)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : C; x : B |] ==> x : B
2. !!x. [| A <= E; B <= E; C <= E; x : C; x : E - C |] ==> x : A Un B
```

```
val it = () : unit
> ba 1;
Level 8 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : C; x : E - C |] ==> x : A Un B
val it = () : unit
> be DiffE 1;
Level 9 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : C; x : E; x ~: C |] ==> x : A Un B
val it = () : unit
> be notE 1;
Level 10 (1 subgoal)
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
1. !!x. [| A <= E; B <= E; C <= E; x : C; x : E |] ==> x : C
val it = () : unit
> ba 1;
Level 11
[| A <= E; B <= E; C <= E |] ==> (A Un C) Int (B Un (E - C)) <= A Un B
No subgoals!
val it = () : unit
```

A.2.5 Propriedades do conjunto das partes

2. $\vdash_{ZF} \mathcal{P}(E) \cap \mathcal{P}(F) = \mathcal{P}(E \cap F)$;

```
> Goal " Pow(E) Int Pow(F) = Pow(E Int F)";
Level 0 (1 subgoal)
Pow(E) Int Pow(F) = Pow(E Int F)
1. Pow(E) Int Pow(F) = Pow(E Int F)
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. Pow(E) Int Pow(F) <= Pow(E Int F)
2. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E) Int Pow(F) ==> x : Pow(E Int F)
2. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> br PowI 1;
Level 3 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
```

```
1. !!x. x : Pow(E) Int Pow(F) ==> x <= E Int F
2. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> br Int_greatest 1;
Level 4 (3 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E) Int Pow(F) ==> x <= E
2. !!x. x : Pow(E) Int Pow(F) ==> x <= F
3. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> br PowD 1;
Level 5 (3 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E) Int Pow(F) ==> x : Pow(E)
2. !!x. x : Pow(E) Int Pow(F) ==> x <= F
3. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> br PowD 2;
Level 6 (3 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E) Int Pow(F) ==> x : Pow(E)
2. !!x. x : Pow(E) Int Pow(F) ==> x : Pow(F)
3. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> be IntE 1;
Level 7 (3 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. [| x : Pow(E); x : Pow(F) |] ==> x : Pow(E)
2. !!x. x : Pow(E) Int Pow(F) ==> x : Pow(F)
3. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> ba 1;
Level 8 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E) Int Pow(F) ==> x : Pow(F)
2. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> be IntE 1;
Level 9 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. [| x : Pow(E); x : Pow(F) |] ==> x : Pow(F)
2. Pow(E Int F) <= Pow(E) Int Pow(F)
val it = () : unit
> ba 1;
Level 10 (1 subgoal)
Pow(E) Int Pow(F) = Pow(E Int F)
1. Pow(E Int F) <= Pow(E) Int Pow(F)
```

```
val it = () : unit
> br subsetI 1;
Level 11 (1 subgoal)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> x : Pow(E) Int Pow(F)
val it = () : unit
> br IntI 1;
Level 12 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> x : Pow(E)
2. !!x. x : Pow(E Int F) ==> x : Pow(F)
val it = () : unit
> br PowI 1;
Level 13 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> x <= E
2. !!x. x : Pow(E Int F) ==> x : Pow(F)
val it = () : unit
> br PowI 2;
Level 14 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> x <= E
2. !!x. x : Pow(E Int F) ==> x <= F
val it = () : unit
> br subset_trans 1;
Level 15 (3 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> x <= ?B12(x)
2. !!x. x : Pow(E Int F) ==> ?B12(x) <= E
3. !!x. x : Pow(E Int F) ==> x <= F
val it = () : unit
> br PowD 1;
Level 16 (3 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> x : Pow(?B12(x))
2. !!x. x : Pow(E Int F) ==> ?B12(x) <= E
3. !!x. x : Pow(E Int F) ==> x <= F
val it = () : unit
> ba 1;
Level 17 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> E Int F <= E
2. !!x. x : Pow(E Int F) ==> x <= F
val it = () : unit
> br Int_lower1 1;
Level 18 (1 subgoal)
Pow(E) Int Pow(F) = Pow(E Int F)
```

```
1. !!x. x : Pow(E Int F) ==> x <= F
val it = () : unit
> br subset_trans 1;
Level 19 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> x <= ?B15(x)
2. !!x. x : Pow(E Int F) ==> ?B15(x) <= F
val it = () : unit
> br PowD 1;
Level 20 (2 subgoals)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> x : Pow(?B15(x))
2. !!x. x : Pow(E Int F) ==> ?B15(x) <= F
val it = () : unit
> ba 1;
Level 21 (1 subgoal)
Pow(E) Int Pow(F) = Pow(E Int F)
1. !!x. x : Pow(E Int F) ==> E Int F <= F
val it = () : unit
> br Int_lower2 1;
Level 22
Pow(E) Int Pow(F) = Pow(E Int F)
No subgoals!
val it = () : unit
```

3. $\vdash_{ZF} \mathcal{P}(E) \cup \mathcal{P}(F) \subseteq \mathcal{P}(E \cup F)$

```
> Goal " Pow(E) Un Pow(F)<= Pow(E Un F)";
Level 0 (1 subgoal)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. Pow(E) Un Pow(F) <= Pow(E Un F)
val it = [] : Thm.thm list
> br subsetI 1;
Level 1 (1 subgoal)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(E) Un Pow(F) ==> x : Pow(E Un F)
val it = () : unit
> br PowI 1;
Level 2 (1 subgoal)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(E) Un Pow(F) ==> x <= E Un F
val it = () : unit
> be UnE 1;
Level 3 (2 subgoals)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(E) ==> x <= E Un F
2. !!x. x : Pow(F) ==> x <= E Un F
```

```
val it = () : unit
> br subset_trans 1;
Level 4 (3 subgoals)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(E) ==> x <= ?B3(x)
2. !!x. x : Pow(E) ==> ?B3(x) <= E Un F
3. !!x. x : Pow(F) ==> x <= E Un F
val it = () : unit
> br PowD 1;
Level 5 (3 subgoals)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(E) ==> x : Pow(?B3(x))
2. !!x. x : Pow(E) ==> ?B3(x) <= E Un F
3. !!x. x : Pow(F) ==> x <= E Un F
val it = () : unit
> ba 1;
Level 6 (2 subgoals)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(E) ==> E <= E Un F
2. !!x. x : Pow(F) ==> x <= E Un F
val it = () : unit
> br Un_upper1 1;
Level 7 (1 subgoal)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(F) ==> x <= E Un F
val it = () : unit
> br subset_trans 1;
Level 8 (2 subgoals)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(F) ==> x <= ?B6(x)
2. !!x. x : Pow(F) ==> ?B6(x) <= E Un F
val it = () : unit
> br PowD 1;
Level 9 (2 subgoals)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(F) ==> x : Pow(?B6(x))
2. !!x. x : Pow(F) ==> ?B6(x) <= E Un F
val it = () : unit
> ba 1;
Level 10 (1 subgoal)
Pow(E) Un Pow(F) <= Pow(E Un F)
1. !!x. x : Pow(F) ==> F <= E Un F
val it = () : unit
> br Un_upper2 1;
Level 11
Pow(E) Un Pow(F) <= Pow(E Un F)
No subgoals!
```

```
val it = () : unit
```

4. $\vdash_{ZF} \cap\mathcal{P}\,(E) = \varnothing$

```
> Goal " Inter(Pow(E))= 0";
Level 0 (1 subgoal)
Inter(Pow(E)) = 0
1. Inter(Pow(E)) = 0
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
Inter(Pow(E)) = 0
1. Inter(Pow(E)) <= 0
2. 0 <= Inter(Pow(E))
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
Inter(Pow(E)) = 0
1. !!x. x : Inter(Pow(E)) ==> x : 0
2. 0 <= Inter(Pow(E))
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
Inter(Pow(E)) = 0
1. !!x. x : Inter(Pow(E)) ==> x : 0
2. !!x. x : 0 ==> x : Inter(Pow(E))
val it = () : unit
> br emptyE 2;
Level 4 (2 subgoals)
Inter(Pow(E)) = 0
1. !!x. x : Inter(Pow(E)) ==> x : 0
2. !!x. x : 0 ==> ?a3(x) : 0
val it = () : unit
> ba 2;
Level 5 (1 subgoal)
Inter(Pow(E)) = 0
1. !!x. x : Inter(Pow(E)) ==> x : 0
val it = () : unit
> be InterE 1;
Level 6 (2 subgoals)
Inter(Pow(E)) = 0
1. !!x. ?B4(x) ~: Pow(E) ==> x : 0
2. !!x. x : ?B4(x) ==> x : 0
val it = () : unit
> ba 2;
Level 7 (1 subgoal)
Inter(Pow(E)) = 0
```

```
1. !!x. 0 ~: Pow(E) ==> x : 0
val it = () : unit
> be notE 1;
Level 8 (1 subgoal)
Inter(Pow(E)) = 0
1. !!x. 0 : Pow(E)
val it = () : unit
> br PowI 1;
Level 9 (1 subgoal)
Inter(Pow(E)) = 0
1. !!x. 0 <= E
val it = () : unit
> br empty_subsetI 1;
Level 10
Inter(Pow(E)) = 0
No subgoals!
val it = () : unit
```

5. $\vdash_{ZF} E = \cup\mathcal{P}\,(E)$

```
> Goal "E = Union(Pow(E))";
Level 0 (1 subgoal)
E = Union(Pow(E))
1. E = Union(Pow(E))
val it = [] : Thm.thm list
> br equalityI 1;
Level 1 (2 subgoals)
E = Union(Pow(E))
1. E <= Union(Pow(E))
2. Union(Pow(E)) <= E
val it = () : unit
> br subsetI 1;
Level 2 (2 subgoals)
E = Union(Pow(E))
1. !!x. x : E ==> x : Union(Pow(E))
2. Union(Pow(E)) <= E
val it = () : unit
> br subsetI 2;
Level 3 (2 subgoals)
E = Union(Pow(E))
1. !!x. x : E ==> x : Union(Pow(E))
2. !!x. x : Union(Pow(E)) ==> x : E
val it = () : unit
> br UnionI 1;
Level 4 (3 subgoals)
E = Union(Pow(E))
1. !!x. x : E ==> ?B3(x) : Pow(E)
```

```
2. !!x. x : E ==> x : ?B3(x)
3. !!x. x : Union(Pow(E)) ==> x : E
val it = () : unit
> ba 2;
Level 5 (2 subgoals)
E = Union(Pow(E))
1. !!x. x : E ==> E : Pow(E)
2. !!x. x : Union(Pow(E)) ==> x : E
val it = () : unit
> br PowI 1;
Level 6 (2 subgoals)
E = Union(Pow(E))
1. !!x. x : E ==> E <= E
2. !!x. x : Union(Pow(E)) ==> x : E
val it = () : unit
> br subsetI 1;
Level 7 (2 subgoals)
E = Union(Pow(E))
1. !!x xa. [| x : E; xa : E |] ==> xa : E
2. !!x. x : Union(Pow(E)) ==> x : E
val it = () : unit
> ba 1;
Level 8 (1 subgoal)
E = Union(Pow(E))
1. !!x. x : Union(Pow(E)) ==> x : E
val it = () : unit
> be UnionE 1;
Level 9 (1 subgoal)
E = Union(Pow(E))
1. !!x B. [| x : B; B : Pow(E) |] ==> x : E
val it = () : unit
> br subsetD 1;
Level 10 (2 subgoals)
E = Union(Pow(E))
1. !!x B. [| x : B; B : Pow(E) |] ==> ?A7(x, B) <= E
2. !!x B. [| x : B; B : Pow(E) |] ==> x : ?A7(x, B)
val it = () : unit
> ba 2;
Level 11 (1 subgoal)
E = Union(Pow(E))
1. !!x B. [| x : B; B : Pow(E) |] ==> B <= E
val it = () : unit
> br PowD 1;
Level 12 (1 subgoal)
E = Union(Pow(E))
1. !!x B. [| x : B; B : Pow(E) |] ==> B : Pow(E)
val it = () : unit
```

```
> ba 1;
Level 13
E = Union(Pow(E))
No subgoals!
val it = () : unit
```

A.2.6 Propriedades dos pares ordenados

3. $\vdash_{ZF} (A - B) \times X = (A \times X) - (B \times X)$

```
> Goal "(A - B)*X = (A*X) - (B*X)";
Level 0 (1 subgoal)
(A - B) * X = A * X - B * X
1. (A - B) * X = A * X - B * X
val it = [] : Thm.thm list
> by (Blast_tac 1);
### Search depth = 0
### Search depth = 1
Level 1
(A - B) * X = A * X - B * X
No subgoals!
val it = () : unit
```

4. $\vdash_{ZF} (A = \varnothing \mid B = \varnothing) \Leftrightarrow A \times B = \varnothing$

```
> Goal "(A=0|B=0)<->A*B=0";
Level 0 (1 subgoal)
A = 0 | B = 0 <-> A * B = 0
1. A = 0 | B = 0 <-> A * B = 0
val it = [] : Thm.thm list
> by (Blast_tac 1);
### Search depth = 0
### Search depth = 1
### Search depth = 2
### Search depth = 3
### Search depth = 4
Level 1
A = 0 | B = 0 <-> A * B = 0
No subgoals!
val it = () : unit
```

5. $\{A \subseteq X; B \subseteq Y\} \vdash_{ZF} A \times B \subseteq X \times Y$

```
> Goal "[|A<=X;B<=Y|]==>A*B<=X*Y";
Level 0 (1 subgoal)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. [| A <= X; B <= Y |] ==> A * B <= X * Y
val it = [] : Thm.thm list
> br subsetI 1;
Level 1 (1 subgoal)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. !!x. [| A <= X; B <= Y; x : A * B |] ==> x : X * Y
val it = () : unit
> be SigmaE 1;
Level 2 (1 subgoal)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. !!x xa y. [| A <= X; B <= Y; xa : A; y : B; x = <xa, y> |] ==> x : X * Y
val it = () : unit
> be lema1 1;
Level 3 (1 subgoal)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. !!x xa y. [| A <= X; B <= Y; xa : A; y : B |] ==> <xa, y> : X * Y
val it = () : unit
> br SigmaI 1;
Level 4 (2 subgoals)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. !!x xa y. [| A <= X; B <= Y; xa : A; y : B |] ==> xa : X
2. !!x xa y. [| A <= X; B <= Y; xa : A; y : B |] ==> y : Y
val it = () : unit
> be subsetD 1;
Level 5 (2 subgoals)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. !!x xa y. [| B <= Y; xa : A; y : B |] ==> xa : A
2. !!x xa y. [| A <= X; B <= Y; xa : A; y : B |] ==> y : Y
val it = () : unit
> ba 1;
Level 6 (1 subgoal)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. !!x xa y. [| A <= X; B <= Y; xa : A; y : B |] ==> y : Y
val it = () : unit
> be subsetD 1;
Level 7 (1 subgoal)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. !!x xa y. [| A <= X; xa : A; y : B |] ==> y : B
val it = () : unit
> ba 1;
```

```
Level 8
[| A <= X; B <= Y |] ==> A * B <= X * Y
No subgoals!
val it = () : unit
```

Ou

```
> Goal "[|A<=X;B<=Y|]==>A*B<=X*Y";
Level 0 (1 subgoal)
[| A <= X; B <= Y |] ==> A * B <= X * Y
1. [| A <= X; B <= Y |] ==> A * B <= X * Y
val it = [] : Thm.thm list
> by (Blast_tac 1);
### Search depth = 0
### Search depth = 1
### Search depth = 2
Level 1
[| A <= X; B <= Y |] ==> A * B <= X * Y
No subgoals!
val it = () : unit
```

6. $\{A \times B \neq \varnothing; A \times B \subseteq X \times Y\} \vdash_{ZF} (A \subseteq X) \ \& \ (B \subseteq Y)$

```
Goal "[|A*B~=0;A*B<=X*Y|]==>(A<=X)&(B<=Y)";
Level 0 (1 subgoal)
[| A * B ~= 0; A * B <= X * Y |] ==> A <= X & B <= Y
1. [| A * B ~= 0; A * B <= X * Y |] ==> A <= X & B <= Y
val it = [] : Thm.thm list
> by (Blast_tac 1);
### Search depth = 0
### Search depth = 1
### Search depth = 2
### Search depth = 3
Level 1
[| A * B ~= 0; A * B <= X * Y |] ==> A <= X & B <= Y
No subgoals!
val it = () : unit
```

A.2.7 Propriedades da aritmética dos naturais

As duas provas seguintes são, respectivamente, as provas das regras *add_0* e *add_0_right*.

```
> Goal "n:nat ==>0#+n=n";
Level 0 (1 subgoal)
n : nat ==> 0 #+ n = n
1. n : nat ==> 0 #+ n = n
```

```
val it = [] : Thm.thm list
> by (subgoal_tac "0#+n=natify(n)" 1);
Level 1 (2 subgoals)
n : nat ==> 0 #+ n = n
1. [| n : nat; 0 #+ n = natify(n) |] ==> 0 #+ n = n
2. n : nat ==> 0 #+ n = natify(n)
val it = () : unit
> by (subgoal_tac "natify(n)=n" 1);
Level 2 (3 subgoals)
n : nat ==> 0 #+ n = n
1. [| n : nat; 0 #+ n = natify(n); natify(n) = n |] ==> 0 #+ n = n
2. [| n : nat; 0 #+ n = natify(n) |] ==> natify(n) = n
3. n : nat ==> 0 #+ n = natify(n)
val it = () : unit
> br add_0_natify 3;
Level 3 (2 subgoals)
n : nat ==> 0 #+ n = n
1. [| n : nat; 0 #+ n = natify(n); natify(n) = n |] ==> 0 #+ n = n
2. [| n : nat; 0 #+ n = natify(n) |] ==> natify(n) = n
val it = () : unit
> br natify_ident 2;
Level 4 (2 subgoals)
n : nat ==> 0 #+ n = n
1. [| n : nat; 0 #+ n = natify(n); natify(n) = n |] ==> 0 #+ n = n
2. [| n : nat; 0 #+ n = natify(n) |] ==> n : nat
val it = () : unit
> ba 2;
Level 5 (1 subgoal)
n : nat ==> 0 #+ n = n
1. [| n : nat; 0 #+ n = natify(n); natify(n) = n |] ==> 0 #+ n = n
val it = () : unit
> be trans 1;
Level 6 (1 subgoal)
n : nat ==> 0 #+ n = n
1. [| n : nat; natify(n) = n |] ==> natify(n) = n
val it = () : unit
> ba 1;
Level 7
n : nat ==> 0 #+ n = n
No subgoals!
val it = () : unit

> Goal "n:nat ==> n#+0=n";
Level 0 (1 subgoal)
n : nat ==> n #+ 0 = n
1. n : nat ==> n #+ 0 = n
val it = [] : Thm.thm list
```

```
> br nat_induct 1;
Level 1 (3 subgoals)
n : nat ==> n #+ 0 = n
1. n : nat ==> n : nat
2. n : nat ==> 0 #+ 0 = 0
3. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> ba 1;
Level 2 (2 subgoals)
n : nat ==> n #+ 0 = n
1. n : nat ==> 0 #+ 0 = 0
2. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> by (subgoal_tac "0#+0=natify(0)" 1);
Level 3 (3 subgoals)
n : nat ==> n #+ 0 = n
1. [| n : nat; 0 #+ 0 = natify(0) |] ==> 0 #+ 0 = 0
2. n : nat ==> 0 #+ 0 = natify(0)
3. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> by (subgoal_tac "natify(0)=0" 1);
Level 4 (4 subgoals)
n : nat ==> n #+ 0 = n
1. [| n : nat; 0 #+ 0 = natify(0); natify(0) = 0 |] ==> 0 #+ 0 = 0
2. [| n : nat; 0 #+ 0 = natify(0) |] ==> natify(0) = 0
3. n : nat ==> 0 #+ 0 = natify(0)
4. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> br natify_0 2;
Level 5 (3 subgoals)
n : nat ==> n #+ 0 = n
1. [| n : nat; 0 #+ 0 = natify(0); natify(0) = 0 |] ==> 0 #+ 0 = 0
2. n : nat ==> 0 #+ 0 = natify(0)
3. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> br add_0_natify 2;
Level 6 (2 subgoals)
n : nat ==> n #+ 0 = n
1. [| n : nat; 0 #+ 0 = natify(0); natify(0) = 0 |] ==> 0 #+ 0 = 0
2. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> br trans 1;
Level 7 (3 subgoals)
n : nat ==> n #+ 0 = n
1. [| n : nat; 0 #+ 0 = natify(0); natify(0) = 0 |] ==> 0 #+ 0 = ?b4
2. [| n : nat; 0 #+ 0 = natify(0); natify(0) = 0 |] ==> ?b4 = 0
3. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
```

```
val it = () : unit
> ba 1;
Level 8 (2 subgoals)
n : nat ==> n #+ 0 = n
1. [| n : nat; 0 #+ 0 = natify(0); natify(0) = 0 |] ==> natify(0) = 0
2. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> ba 1;
Level 9 (1 subgoal)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x)
val it = () : unit
> by (subgoal_tac "succ(x)#+0=succ(x#+0)" 1);
Level 10 (2 subgoals)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x) #+ 0 = succ(x #+ 0) |]
==> succ(x) #+ 0 = succ(x)
2. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x #+ 0)
val it = () : unit
> by (subgoal_tac "succ(x#+0)=succ(x)" 1);
Level 11 (3 subgoals)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x) #+ 0 = succ(x #+ 0);
succ(x #+ 0) = succ(x) |]
==> succ(x) #+ 0 = succ(x)
2. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x) #+ 0 = succ(x #+ 0) |]
==> succ(x #+ 0) = succ(x)
3. !!x. [| n : nat; x : nat; x #+ 0 = x |] ==> succ(x) #+ 0 = succ(x #+ 0)
val it = () : unit
> br add_succ 3;
Level 12 (2 subgoals)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x) #+ 0 = succ(x #+ 0);
succ(x #+ 0) = succ(x) |]
==> succ(x) #+ 0 = succ(x)
2. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x) #+ 0 = succ(x #+ 0) |]
==> succ(x #+ 0) = succ(x)
val it = () : unit
> br subsucc 2;
Level 13 (2 subgoals)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x) #+ 0 = succ(x #+ 0);
succ(x #+ 0) = succ(x) |]
==> succ(x) #+ 0 = succ(x)
2. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x) #+ 0 = succ(x #+ 0) |]
==> x #+ 0 = x
val it = () : unit
```

```
> ba 2;
Level 14 (1 subgoal)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x) #+ 0 = succ(x #+ 0);
succ(x #+ 0) = succ(x) |]
==> succ(x) #+ 0 = succ(x)
val it = () : unit
> be trans 1;
Level 15 (1 subgoal)
n : nat ==> n #+ 0 = n
1. !!x. [| n : nat; x : nat; x #+ 0 = x; succ(x #+ 0) = succ(x) |]
==> succ(x #+ 0) = succ(x)
val it = () : unit
> ba 1;
Level 16
n : nat ==> n #+ 0 = n
No subgoals!
val it = () : unit
```

De seguida pretende-se provar a propriedade associativa da adição. No entanto, para facilitar a prova da mesma é necessário em primeiro lugar provar o lema seguinte, que será designado *id.*

```
> Goal "[|m:nat;n:nat;k:nat;m=n|]==> k#+m = k#+n";
Level 0 (1 subgoal)
[| m : nat; n : nat; k : nat; m = n |] ==> k #+ m = k #+ n
1. [| m : nat; n : nat; k : nat; m = n |] ==> k #+ m = k #+ n
val it = [] : Thm.thm list
> be subst 1;
Level 1 (1 subgoal)
[| m : nat; n : nat; k : nat; m = n |] ==> k #+ m = k #+ n
1. [| m : nat; n : nat; k : nat |] ==> k #+ m = k #+ m
val it = () : unit
> br refl 1;
Level 2
[| m : nat; n : nat; k : nat; m = n |] ==> k #+ m = k #+ n
No subgoals!
val it = () : unit
> qed "id";
val id =
"[| ?m : nat; ?n : nat; ?k : nat; ?m = ?n |] ==> ?k #+ ?m = ?k #+ ?n"
: Thm.thm
val it = () : unit
```

Agora está-se em condições de provar a associativa da adição. Chama-se à atenção o leitor para o facto de que a propriedade associativa é a regra *add_ assoc,* em *Isabelle* ZF.

```
> Goal "[|n:nat;k:nat;m:nat|]==>(k#+m)#+n=k#+(m#+n)";
Level 0 (1 subgoal)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
val it = [] : Thm.thm list
> br nat_induct 1;
Level 1 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat |] ==> k #+ (m #+ n) : nat
2. [| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = 0
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ n = x |]
==> k #+ m #+ n = succ(x)
val it = () : unit
> back();
Level 1 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat |] ==> m #+ n : nat
2. [| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ 0
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ n = k #+ x |]
==> k #+ m #+ n = k #+ succ(x)
val it = () : unit
> back();
Level 1 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat |] ==> n : nat
2. [| n : nat; k : nat; m : nat |] ==> k #+ m #+ 0 = k #+ (m #+ 0)
3. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> ba 1;
Level 2 (2 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat |] ==> k #+ m #+ 0 = k #+ (m #+ 0)
2. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> by (subgoals_tac ["(k#+m)#+0 = k#+m","k#+m = k#+(m#+0)"] 1);
Level 3 (4 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m;
k #+ m = k #+ (m #+ 0) |]
==> k #+ m #+ 0 = k #+ (m #+ 0)
2. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |]
==> k #+ m = k #+ (m #+ 0)
3. [| n : nat; k : nat; m : nat |] ==> k #+ m #+ 0 = k #+ m
```

```
4. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> br add_0_right 3;
Level 4 (4 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m;
k #+ m = k #+ (m #+ 0) |]
==> k #+ m #+ 0 = k #+ (m #+ 0)
2. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |]
==> k #+ m = k #+ (m #+ 0)
3. [| n : nat; k : nat; m : nat |] ==> k #+ m : nat
4. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> br add_type 3;
Level 5 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m;
k #+ m = k #+ (m #+ 0) |]
==> k #+ m #+ 0 = k #+ (m #+ 0)
2. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |]
==> k #+ m = k #+ (m #+ 0)
3. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> be trans 1;
Level 6 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m = k #+ (m #+ 0) |]
==> k #+ m = k #+ (m #+ 0)
2. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |]
==> k #+ m = k #+ (m #+ 0)
3. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> ba 1;
Level 7 (2 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |]
==> k #+ m = k #+ (m #+ 0)
2. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
```

```
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> br id 1;
Level 8 (5 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m : nat
2. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m #+ 0 : nat
3. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> k : nat
4. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m = m #+ 0
5. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> ba 1;
Level 9 (4 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m #+ 0 : nat
2. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> k : nat
3. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m = m #+ 0
4. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> br add_type 1;
Level 10 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> k : nat
2. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m = m #+ 0
3. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> ba 1;
Level 11 (2 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m = m #+ 0
2. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> br sym 1;
Level 12 (2 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m #+ 0 = m
2. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
```

```
val it = () : unit
> br add_0_right 1;
Level 13 (2 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. [| n : nat; k : nat; m : nat; k #+ m #+ 0 = k #+ m |] ==> m : nat
2. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> ba 1;
Level 14 (1 subgoal)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> by (subgoals_tac ["k #+ m #+ succ(x)=succ((k #+ m)#+x)",
"succ((k #+ m)#+x)=succ(k#+(m#+x))", "succ(k#+(m#+x))=k#+succ(m#+x)",
"k#+succ(m#+x)=k #+ (m #+ succ(x))"] 1);
Level 15 (5 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k #+ succ(m #+ x) = k #+ (m #+ succ(x))
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x)) |]
==> succ(k #+ (m #+ x)) = k #+ succ(m #+ x)
4. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x) |]
==> succ(k #+ m #+ x) = succ(k #+ (m #+ x))
5. !!x. [| n : nat; k : nat; m : nat; x : nat;
k #+ m #+ x = k #+ (m #+ x) |]
==> k #+ m #+ succ(x) = succ(k #+ m #+ x)
val it = () : unit
> br add_succ_right 5;
Level 16 (4 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
```

```
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k #+ succ(m #+ x) = k #+ (m #+ succ(x))
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x)) |]
==> succ(k #+ (m #+ x)) = k #+ succ(m #+ x)
4. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x) |]
==> succ(k #+ m #+ x) = succ(k #+ (m #+ x))
val it = () : unit
> br subsucc 4;
Level 17 (4 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k #+ succ(m #+ x) = k #+ (m #+ succ(x))
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x)) |]
==> succ(k #+ (m #+ x)) = k #+ succ(m #+ x)
4. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x) |]
==> k #+ m #+ x = k #+ (m #+ x)
val it = () : unit
> ba 4;
Level 18 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
```

```
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k #+ succ(m #+ x) = k #+ (m #+ succ(x))
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x)) |]
==> succ(k #+ (m #+ x)) = k #+ succ(m #+ x)
val it = () : unit
> br sym 3;
Level 19 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k #+ succ(m #+ x) = k #+ (m #+ succ(x))
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x)) |]
==> k #+ succ(m #+ x) = succ(k #+ (m #+ x))
val it = () : unit
> br add_succ_right 3;
Level 20 (2 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k #+ succ(m #+ x) = k #+ (m #+ succ(x))
val it = () : unit
> br id 2;
```

```
Level 21 (5 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> succ(m #+ x) : nat
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> m #+ succ(x) : nat
4. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k : nat
5. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> succ(m #+ x) = m #+ succ(x)
val it = () : unit
> br sym 5;
Level 22 (5 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> succ(m #+ x) : nat
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
```

```
==> m #+ succ(x) : nat
4. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k : nat
5. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> m #+ succ(x) = succ(m #+ x)
val it = () : unit
> br add_succ_right 5;
Level 23 (4 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> succ(m #+ x) : nat
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> m #+ succ(x) : nat
4. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> k : nat
val it = () : unit
> ba 4;
Level 24 (3 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
```

```
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> succ(m #+ x) : nat
3. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> m #+ succ(x) : nat
val it = () : unit
> br add_type 3;
Level 25 (2 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> succ(m #+ x) : nat
val it = () : unit
> br nat_succI 2;
Level 26 (2 subgoals)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
2. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x) |]
==> m #+ x : nat
val it = () : unit
> br add_type 2;
Level 27 (1 subgoal)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ m #+ succ(x) = succ(k #+ m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
```

```
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ m #+ succ(x) = k #+ (m #+ succ(x))
val it = () : unit
> be trans 1;
Level 28 (1 subgoal)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
succ(k #+ m #+ x) = succ(k #+ (m #+ x));
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> succ(k #+ m #+ x) = k #+ (m #+ succ(x))
val it = () : unit
> be trans 1;
Level 29 (1 subgoal)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
succ(k #+ (m #+ x)) = k #+ succ(m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> succ(k #+ (m #+ x)) = k #+ (m #+ succ(x))
val it = () : unit
> be trans 1;
Level 30 (1 subgoal)
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
1. !!x. [| n : nat; k : nat; m : nat; x : nat; k #+ m #+ x = k #+ (m #+ x);
k #+ succ(m #+ x) = k #+ (m #+ succ(x)) |]
==> k #+ succ(m #+ x) = k #+ (m #+ succ(x))
val it = () : unit
> ba 1;
Level 31
[| n : nat; k : nat; m : nat |] ==> k #+ m #+ n = k #+ (m #+ n)
No subgoals!
val it = () : unit
```

De seguida, apresenta-se a prova da propriedade comutativa da multiplicação. Esta propriedade é a regra *mult_ commute,* em *Isabelle* ZF.

```
> Goal "[|n:nat;m:nat|]==>m#*n=n#*m";
Level 0 (1 subgoal)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat |] ==> m #* n = n #* m
val it = [] : Thm.thm list
> br nat_induct 1;
Level 1 (3 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat |] ==> n #* m : nat
2. [| n : nat; m : nat |] ==> m #* n = 0
3. !!x. [| n : nat; m : nat; x : nat; m #* n = x |] ==> m #* n = succ(x)
```

```
val it = () : unit
> back();
Level 1 (3 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat |] ==> m : nat
2. [| n : nat; m : nat |] ==> 0 #* n = n #* 0
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #* succ(x)
val it = () : unit
> ba 1;
Level 2 (2 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat |] ==> 0 #* n = n #* 0
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #* succ(x)
val it = () : unit
> by (subgoals_tac ["0#*n=0","0=n#*0"] 1);
Level 3 (4 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat; 0 #* n = 0; 0 = n #* 0 |] ==> 0 #* n = n #* 0
2. [| n : nat; m : nat; 0 #* n = 0 |] ==> 0 = n #* 0
3. [| n : nat; m : nat |] ==> 0 #* n = 0
4. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #* succ(x)
val it = () : unit
> br mult_0 3;
Level 4 (3 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat; 0 #* n = 0; 0 = n #* 0 |] ==> 0 #* n = n #* 0
2. [| n : nat; m : nat; 0 #* n = 0 |] ==> 0 = n #* 0
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #* succ(x)
val it = () : unit
> br sym 2;
Level 5 (3 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat; 0 #* n = 0; 0 = n #* 0 |] ==> 0 #* n = n #* 0
2. [| n : nat; m : nat; 0 #* n = 0 |] ==> n #* 0 = 0
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #* succ(x)
val it = () : unit
> br mult_0_right 2;
Level 6 (2 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat; 0 #* n = 0; 0 = n #* 0 |] ==> 0 #* n = n #* 0
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #* succ(x)
```

```
val it = () : unit
> be trans 1;
Level 7 (2 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. [| n : nat; m : nat; 0 = n #* 0 |] ==> 0 = n #* 0
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #* succ(x)
val it = () : unit
> ba 1;
Level 8 (1 subgoal)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #* succ(x)
val it = () : unit
> by (subgoals_tac ["succ(x)#*n = n#+x#*n", "n#+x#*n = n#+n#*x", "
n#+n#*x = n#*succ(x)"] 1);
Level 9 (4 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n; n #+ x #* n = n #+ n #* x;
n #+ n #* x = n #* succ(x) |]
==> succ(x) #* n = n #* succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n; n #+ x #* n = n #+ n #* x |]
==> n #+ n #* x = n #* succ(x)
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #+ x #* n = n #+ n #* x
4. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #+ x #* n
val it = () : unit
> be trans 1;
Level 10 (4 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
n #+ x #* n = n #+ n #* x; n #+ n #* x = n #* succ(x) |]
==> n #+ x #* n = n #* succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n; n #+ x #* n = n #+ n #* x |]
==> n #+ n #* x = n #* succ(x)
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #+ x #* n = n #+ n #* x
4. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #+ x #* n
val it = () : unit
> be trans 1;
```

```
Level 11 (4 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
n #+ n #* x = n #* succ(x) |]
==> n #+ n #* x = n #* succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n; n #+ x #* n = n #+ n #* x |]
==> n #+ n #* x = n #* succ(x)
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #+ x #* n = n #+ n #* x
4. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #+ x #* n
val it = () : unit
> ba 1;
Level 12 (3 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n; n #+ x #* n = n #+ n #* x |]
==> n #+ n #* x = n #* succ(x)
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #+ x #* n = n #+ n #* x
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #+ x #* n
val it = () : unit
> br sym 1;
Level 13 (3 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n; n #+ x #* n = n #+ n #* x |]
==> n #* succ(x) = n #+ n #* x
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #+ x #* n = n #+ n #* x
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #+ x #* n
val it = () : unit
> br mult_succ_right 1;
Level 14 (2 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #+ x #* n = n #+ n #* x
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x |]
==> succ(x) #* n = n #+ x #* n
val it = () : unit
```

```
> br mult_succ 2;
Level 15 (1 subgoal)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #+ x #* n = n #+ n #* x
val it = () : unit
> br id 1;
Level 16 (4 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> x #* n : nat
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #* x : nat
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n : nat
4. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> x #* n = n #* x
val it = () : unit
> br mult_type 1;
Level 17 (3 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n #* x : nat
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n : nat
3. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> x #* n = n #* x
val it = () : unit
> br mult_type 1;
Level 18 (2 subgoals)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> n : nat
2. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> x #* n = n #* x
val it = () : unit
> ba 1;
```

```
Level 19 (1 subgoal)
[| n : nat; m : nat |] ==> m #* n = n #* m
1. !!x. [| n : nat; m : nat; x : nat; x #* n = n #* x;
succ(x) #* n = n #+ x #* n |]
==> x #* n = n #* x
val it = () : unit
> ba 1;
Level 20
[| n : nat; m : nat |] ==> m #* n = n #* m
No subgoals!
val it = () : unit
```

A próxima propriedade a provar é a propriedade associativa da multiplicação. No entanto, antes do fazer é necessário estabelecer o lema seguinte, que será designado *id2*.

```
> Goal "[|k:nat;m:nat;k:nat;m=n|]==>k#*m=k#*n";
Level 0 (1 subgoal)
[| k : nat; m : nat; k : nat; m = n |] ==> k #* m = k #* n
1. [| k : nat; m : nat; k : nat; m = n |] ==> k #* m = k #* n
val it = [] : Thm.thm list
> be subst 1;
Level 1 (1 subgoal)
[| k : nat; m : nat; k : nat; m = n |] ==> k #* m = k #* n
1. [| k : nat; m : nat; k : nat |] ==> k #* m = k #* m
val it = () : unit
> br refl 1;
Level 2
[| k : nat; m : nat; k : nat; m = n |] ==> k #* m = k #* n
No subgoals!
val it = () : unit
> qed "id2";
val id2 =
"[| ?k : nat; ?m : nat; ?k : nat; ?m = ?n |] ==> ?k #* ?m = ?k #* ?n"
: Thm.thm
val it = () : unit
```

Segue-se assim, a prova da associativa da multiplicação.

```
> Goal "[|k:nat;m:nat;n:nat|]==> (k#*m)#*n = k#*(m#*n)";
Level 0 (1 subgoal)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
val it = [] : Thm.thm list
> br nat_induct 1;
Level 1 (3 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat |] ==> k #* (m #* n) : nat
```

```
  2. [| k : nat; m : nat; n : nat |] ==> k #* m #* n = 0
  3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* n = x |]
  ==> k #* m #* n = succ(x)
  val it = () : unit
  > back();
  Level 1 (3 subgoals)
  [| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
  1. [| k : nat; m : nat; n : nat |] ==> m #* n : nat
  2. [| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* 0
  3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* n = k #* x |]
  ==> k #* m #* n = k #* succ(x)
  val it = () : unit
  > back();
  Level 1 (3 subgoals)
  [| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
  1. [| k : nat; m : nat; n : nat |] ==> n : nat
  2. [| k : nat; m : nat; n : nat |] ==> k #* m #* 0 = k #* (m #* 0)
  3. !!x. [| k : nat; m : nat; n : nat; x : nat;
  k #* m #* x = k #* (m #* x) |]
  ==> k #* m #* succ(x) = k #* (m #* succ(x))
  val it = () : unit
  > ba 1;
  Level 2 (2 subgoals)
  [| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
  1. [| k : nat; m : nat; n : nat |] ==> k #* m #* 0 = k #* (m #* 0)
  2. !!x. [| k : nat; m : nat; n : nat; x : nat;
  k #* m #* x = k #* (m #* x) |]
  ==> k #* m #* succ(x) = k #* (m #* succ(x))
  val it = () : unit
  > by (subgoals_tac ["k #* m #* 0 = 0","0 = k#*0","k#*0= k #* (m #* 0)"]
1);
  Level 3 (5 subgoals)
  [| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
  1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
  k #* 0 = k #* (m #* 0) |]
  ==> k #* m #* 0 = k #* (m #* 0)
  2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |]
  ==> k #* 0 = k #* (m #* 0)
  3. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0 |] ==> 0 = k #* 0
  4. [| k : nat; m : nat; n : nat |] ==> k #* m #* 0 = 0
  5. !!x. [| k : nat; m : nat; n : nat; x : nat;
  k #* m #* x = k #* (m #* x) |]
  ==> k #* m #* succ(x) = k #* (m #* succ(x))
  val it = () : unit
  > br mult_0_right 4;
  Level 4 (4 subgoals)
  [| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
```

```
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |]
==> k #* 0 = k #* (m #* 0)
3. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0 |] ==> 0 = k #* 0
4. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> br sym 3;
Level 5 (4 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |]
==> k #* 0 = k #* (m #* 0)
3. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0 |] ==> k #* 0 = 0
4. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> br mult_0_right 3;
Level 6 (3 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |]
==> k #* 0 = k #* (m #* 0)
3. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> br id2 2;
Level 7 (6 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> k : nat
3. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> 0 : nat
4. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> k : nat
5. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |]
==> 0 = m #* 0
6. !!x. [| k : nat; m : nat; n : nat; x : nat;
```

```
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> br sym 5;
Level 8 (6 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> k : nat
3. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> 0 : nat
4. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> k : nat
5. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |]
==> m #* 0 = 0
6. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> br mult_0_right 5;
Level 9 (5 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> k : nat
3. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> 0 : nat
4. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> k : nat
5. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> ba 2;
Level 10 (4 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> 0 : nat
3. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> k : nat
4. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> br nat_0I 2;
Level 11 (3 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
```

```
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0 |] ==> k : nat
3. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> ba 2;
Level 12 (2 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* m #* 0 = 0; 0 = k #* 0;
k #* 0 = k #* (m #* 0) |]
==> k #* m #* 0 = k #* (m #* 0)
2. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> be trans 1;
Level 13 (2 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; 0 = k #* 0; k #* 0 = k #* (m #* 0) |]
==> 0 = k #* (m #* 0)
2. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> be trans 1;
Level 14 (2 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. [| k : nat; m : nat; n : nat; k #* 0 = k #* (m #* 0) |]
==> k #* 0 = k #* (m #* 0)
2. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> ba 1;
Level 15 (1 subgoal)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> by (subgoals_tac [" k #* m #* succ(x) = (k#*m)#+(k#*m)#*x",
"(k#*m)#+(k#*m)#*x = (k#*m)#+ k#*(m#*x)", "(k#*m)#+ k#*(m#*x) =
k#*(m#+m#*x)", "k#*(m#+m#*x)= k#*(m#*succ(x))"] 1);
```

```
Level 16 (5 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x) |]
==> k #* m #+ k #* (m #* x) = k #* (m #+ m #* x)
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x)
5. !!x. [| k : nat; m : nat; n : nat; x : nat;
k #* m #* x = k #* (m #* x) |]
==> k #* m #* succ(x) = k #* m #+ k #* m #* x
val it = () : unit
> br mult_succ_right 5;
Level 17 (4 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x) |]
==> k #* m #+ k #* (m #* x) = k #* (m #+ m #* x)
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x)
val it = () : unit
> br id 4;
```

```
Level 18 (7 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x) |]
==> k #* m #+ k #* (m #* x) = k #* (m #+ m #* x)
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m #* x : nat
5. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* (m #* x) : nat
6. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m : nat
7. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m #* x = k #* (m #* x)
val it = () : unit
> ba 7;
Level 19 (6 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x) |]
```

```
==> k #* m #+ k #* (m #* x) = k #* (m #+ m #* x)
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m #* x : nat
5. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* (m #* x) : nat
6. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m : nat
val it = () : unit
> br mult_type 6;
Level 20 (5 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x) |]
==> k #* m #+ k #* (m #* x) = k #* (m #+ m #* x)
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m #* x : nat
5. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* (m #* x) : nat
val it = () : unit
> br mult_type 5;
Level 21 (4 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
```

```
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x) |]
==> k #* m #+ k #* (m #* x) = k #* (m #+ m #* x)
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x |]
==> k #* m #* x : nat
val it = () : unit
> br mult_type 4;
Level 22 (3 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x) |]
==> k #* m #+ k #* (m #* x) = k #* (m #+ m #* x)
val it = () : unit
> br sym 3;
Level 23 (3 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x) |]
```

```
==> k #* (m #+ m #* x) = k #* m #+ k #* (m #* x)
val it = () : unit
> br add_mult_distrib_left 3;
Level 24 (2 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
val it = () : unit
> br id2 2;
Level 25 (5 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k : nat
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> m #+ m #* x : nat
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k : nat
5. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> m #+ m #* x = m #* succ(x)
val it = () : unit
```

```
> br sym 5;
Level 26 (5 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k : nat
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> m #+ m #* x : nat
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k : nat
5. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> m #* succ(x) = m #+ m #* x
val it = () : unit
> br mult_succ_right 5;
Level 27 (4 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k : nat
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
```

```
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> m #+ m #* x : nat
4. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k : nat
val it = () : unit
> ba 4;
Level 28 (3 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k : nat
3. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> m #+ m #* x : nat
val it = () : unit
> br add_type 3;
Level 29 (2 subgoals)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
2. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x) |]
==> k : nat
val it = () : unit
> ba 2;
Level 30 (1 subgoal)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
```

```
k #* m #* succ(x) = k #* m #+ k #* m #* x;
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #* succ(x) = k #* (m #* succ(x))
val it = () : unit
> be trans 1;
Level 31 (1 subgoal)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #+ k #* m #* x = k #* m #+ k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #+ k #* m #* x = k #* (m #* succ(x))
val it = () : unit
> be trans 1;
Level 32 (1 subgoal)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* m #+ k #* (m #* x) = k #* (m #+ m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* m #+ k #* (m #* x) = k #* (m #* succ(x))
val it = () : unit
> be trans 1;
Level 33 (1 subgoal)
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
1. !!x. [| k : nat; m : nat; n : nat; x : nat; k #* m #* x = k #* (m #* x);
k #* (m #+ m #* x) = k #* (m #* succ(x)) |]
==> k #* (m #+ m #* x) = k #* (m #* succ(x))
val it = () : unit
> ba 1;
Level 34
[| k : nat; m : nat; n : nat |] ==> k #* m #* n = k #* (m #* n)
No subgoals!
val it = () : unit
```

A.3 Simplificação

A.3.1 FOL_ss

O conteúdo do conjunto de simplificação da teoria FOL, como foi referido na secção 2.4, pode ser visto se se digitar o seguinte comando

```
print_ss FOL_ss;
```

Este conjunto contém regras de simplificação, procedimentos de simplificação e congruências, como se pode ver de seguida.

```
simplification rules:
"(True ==> PROP ?P) == PROP ?P"
"(?P1 & ?Q1) & ?R1 == ?P1 & ?Q1 & ?R1"
"(?P1 -> ?Q) & (~?P1 -> ?Q) == ?Q"
"~?P1 & ?P1 == False"
"?P1 & ?P1 & ?Q1 == ?P1 & ?Q1"
"?P1 & ~?P1 == False"
"?Q & ?Q == ?Q"
"?P1 & False == False"
"?Q & True == ?Q"
"False & ?P1 == False"
"True & ?Q == ?Q"
"~?Q -> ?Q == ?Q"
"?P1 -> ~?P1 == ~?P1"
"?P1 -> ?P1 == True"
"?P1 -> False == ~?P1"
"?P1 -> True == True"
"False -> ?P1 == True"
"True -> ?Q == ?Q"
"~?P1 <-> ~?Q1 == ?P1 <-> ?Q1"
"?P1 <-> ?P1 == True"
"?P1 <-> False == ~?P1"
"?Q <-> True == ?Q"
"False <-> ?P1 == ~?P1"
"True <-> ?Q == ?Q"
"?a1 = ?a1 == True"
"(?P1 -> ?Q1) | ?R1 == ?P1 -> ?Q1 | ?R1"
"(?P1 | ?Q1) | ?R1 == ?P1 | ?Q1 | ?R1"
"~?P1 | ?P1 == True"
"?Q1 | (?P1 -> ?R1) == ?P1 -> ?Q1 | ?R1"
"?P1 | ?P1 | ?Q1 == ?P1 | ?Q1"
"?P1 | ~?P1 == True"
"?Q | ?Q == ?Q"
"?Q | False == ?Q"
"?P1 | True == True"
"False | ?Q == ?Q"
"True | ?P1 == True"
"ALL x. ?P1 | ?Q1(x) == ?P1 | (ALL x. ?Q1(x))"
"ALL x. ?P1(x) | ?Q1 == (ALL x. ?P1(x)) | ?Q1"
"ALL x. ?P1 -> ?Q1(x) == ?P1 -> (ALL x. ?Q1(x))"
"ALL x. ?P1(x) -> ?Q1 == (EX x. ?P1(x)) -> ?Q1"
"ALL x. ?P1 & ?Q1(x) == ?P1 & (ALL x. ?Q1(x))"
"ALL x. ?P1(x) & ?Q1 == (ALL x. ?P1(x)) & ?Q1"
"ALL x. ?t1 = x -> ?P1(x) == ?P1(?t1)"
```

```
"ALL x. x = ?t1 -> ?P1(x) == ?P1(?t1)"
"ALL x. ?Q == ?Q"
"EX x. ?t1 = x == True"
"EX x. ?P1 -> ?Q1(x) == ?P1 -> (EX x. ?Q1(x))"
"EX x. ?P1(x) -> ?Q1 == (ALL x. ?P1(x)) -> ?Q1"
"EX x. ?P1 | ?Q1(x) == ?P1 | (EX x. ?Q1(x))"
"EX x. ?P1(x) | ?Q1 == (EX x. ?P1(x)) | ?Q1"
"EX x. ?P1 & ?Q1(x) == ?P1 & (EX x. ?Q1(x))"
"EX x. ?P1(x) & ?Q1 == (EX x. ?P1(x)) & ?Q1"
"EX x. ?t1 = x & ?P1(x) == ?P1(?t1)"
"EX x. x = ?t1 & ?P1(x) == ?P1(?t1)"
"EX x. x = ?t1 == True"
"EX x. ?Q == ?Q"
"~(?P1 & ?Q1) == ~?P1 | ~?Q1"
"~(?P1 -> ?Q1) == ?P1 & ~?Q1"
"~(?P1 | ?Q1) == ~?P1 & ~?Q1"
"~(ALL x. ?P1(x)) == EX x. ~?P1(x)"
"~(EX x. ?P1(x)) == ALL x. ~?P1(x)"
"~~?Q == ?Q"
"~False == True"
"~True == False"
"(!!x. PROP ?V) == PROP ?V"
simplification procedures:
defined ALL:
ALL x. ?P(x)
defined EX:
EX x. ?P(x)
congruences:
"[| ?P == ?P'; ?P' ==> ?Q == ?Q' |] ==> ?P -> ?Q == ?P' -> ?Q'"
val it = () : unit
```

A.3.2 ZF_ss

Para se visualizar o conteúdo do conjunto de simplificação da teoria ZF, procede-se de forma análoga à do FOL, tendo em conta que, neste caso, o conjunto de simplificação é designado por ZF_ss.

```
> print_ss ZF_ss;
simplification rules:
"if False then ?a1 else ?y == ?y" [.]
"if True then ?y else ?b1 == ?y" [.]
"(True ==> PROP ?P) == PROP ?P"
"ALL x:Collect(?A1, ?Q1). ?P1(x) == ALL x:?A1. ?Q1(x) -> ?P1(x)"
"ALL x:RepFun(?A1, ?f1). ?P1(x) == ALL y:?A1. ?P1(?f1(y))"
"ALL x:cons(?a1, ?B1). ?P1(x) == ?P1(?a1) & (ALL x:?B1. ?P1(x))"
"ALL x:Union(?A1). ?P1(x) == ALL y:?A1. ALL x:y. ?P1(x)"
```

```
"ALL x:succ(?i1). ?P1(x) == ?P1(?i1) & (ALL x:?i1. ?P1(x))"
"ALL x:?A1. ?a1 = x -> ?P1(x) == ?a1 : ?A1 -> ?P1(?a1)"
"ALL x:?A1. x = ?a1 -> ?P1(x) == ?a1 : ?A1 -> ?P1(?a1)"
"ALL x:?A1. ?P1 -> ?Q1(x) == ?P1 -> (ALL x:?A1. ?Q1(x))"
"ALL x:?A1. ?P1 | ?Q1(x) == ?P1 | (ALL x:?A1. ?Q1(x))"
"ALL x:?A1. ?P1 & ?Q1(x) == (?A1 = 0 | ?P1) & (ALL x:?A1. ?Q1(x))"
"ALL x:?A1. ?P1(x) -> ?Q1 == (EX x:?A1. ?P1(x)) -> ?Q1"
"ALL x:?A1. ?P1(x) | ?Q1 == (ALL x:?A1. ?P1(x)) | ?Q1"
"ALL x:?A1. ?P1(x) & ?Q1 == (ALL x:?A1. ?P1(x)) & (?A1 = 0 | ?Q1)"
"ALL x:?A1. ?P1 == (EX x. x : ?A1) -> ?P1"
"ALL x:0. ?P1(x) == True"
"EX x:Collect(?A1, ?Q1). ?P1(x) == EX x:?A1. ?Q1(x) & ?P1(x)"
"EX x:RepFun(?A1, ?f1). ?P1(x) == EX y:?A1. ?P1(?f1(y))"
"EX x:cons(?a1, ?B1). ?P1(x) == ?P1(?a1) | (EX x:?B1. ?P1(x))"
"EX x:Union(?A1). ?P1(x) == EX y:?A1. EX x:y. ?P1(x)"
"EX x:succ(?i1). ?P1(x) == ?P1(?i1) | (EX x:?i1. ?P1(x))"
"EX x:?A1. ?a1 = x == ?a1 : ?A1"
"EX x:?A1. ?a1 = x & ?P1(x) == ?a1 : ?A1 & ?P1(?a1)"
"EX x:?A1. x = ?a1 & ?P1(x) == ?a1 : ?A1 & ?P1(?a1)"
"EX x:?A1. x = ?a1 == ?a1 : ?A1"
"EX x:?A1. ?P1 -> ?Q1(x) == ?A1 = 0 | ?P1 -> (EX x:?A1. ?Q1(x))"
"EX x:?A1. ?P1 | ?Q1(x) == ?A1 ~= 0 & ?P1 | (EX x:?A1. ?Q1(x))"
"EX x:?A1. ?P1 & ?Q1(x) == ?P1 & (EX x:?A1. ?Q1(x))"
"EX x:?A1. ?P1(x) -> ?Q1 == (ALL x:?A1. ?P1(x)) -> ?A1 ~= 0 & ?Q1"
"EX x:?A1. ?P1(x) | ?Q1 == (EX x:?A1. ?P1(x)) | ?A1 ~= 0 & ?Q1"
"EX x:?A1. ?P1(x) & ?Q1 == (EX x:?A1. ?P1(x)) & ?Q1"
"EX x:?A1. ?P1 == (EX x. x : ?A1) & ?P1"
"EX x:0. ?P1(x) == False"
"{x: ?A1 . ?Q1} == if ?Q1 then ?A1 else 0" [.]
"{x: 0 . ?P1(x)} == 0" [.]
"RepFun(cons(?a1, ?B1), ?f1) == cons(?f1(?a1), RepFun(?B1, ?f1))" [.]
"RepFun(succ(?i1), ?f1) == cons(?f1(?i1), RepFun(?i1, ?f1))" [.]
"{x . x: ?y} == ?y" [.]
"RepFun(0, ?f1) == 0" [.]
"{x . y: 0, ?R1(x, y)} == 0" [.]
"(?P1 & ?Q1) & ?R1 == ?P1 & ?Q1 & ?R1"
"(?P1 -> ?Q) & (~?P1 -> ?Q) == ?Q"
"~?P1 & ?P1 == False"
"?P1 & ?P1 & ?Q1 == ?P1 & ?Q1"
"?P1 & ~?P1 == False"
"?Q & ?Q == ?Q"
"?P1 & False == False"
"?Q & True == ?Q"
"False & ?P1 == False"
"True & ?Q == ?Q"
"?y - 0 == ?y" [.]
"0 - ?A1 == 0" [.]
```

```
"~?Q -> ?Q == ?Q"
"?P1 -> ~?P1 == ~?P1"
"?P1 -> ?P1 == True"
"?P1 -> False == ~?P1"
"?P1 -> True == True"
"False -> ?P1 == True"
"True -> ?Q == ?Q"
"?a1 : {x: ?A1 . ?P1(x)} == ?a1 : ?A1 & ?P1(?a1)"
"?b1 : {?f1(x) . x: ?A1} == EX x:?A1. ?b1 = ?f1(x)"
"?c1 : Upair(?a1, ?b1) == ?c1 = ?a1 | ?c1 = ?b1"
"?a1 : cons(?a1, ?B1) == True"
"?a1 : cons(?b1, ?A1) == ?a1 = ?b1 | ?a1 : ?A1"
"?c1 : ?A1 - ?B1 == ?c1 : ?A1 & ?c1 ~: ?B1"
"?c1 : ?A1 Int ?B1 == ?c1 : ?A1 & ?c1 : ?B1"
"?c1 : ?A1 Un ?B1 == ?c1 : ?A1 | ?c1 : ?B1"
"?A1 : Pow(?B1) == ?A1 <= ?B1"
"?b1 : (UN x:?A1. ?B1(x)) == EX x:?A1. ?b1 : ?B1(x)"
"?A1 : Union(?C1) == EX B:?C1. ?A1 : B"
"?i1 : succ(?i1) == True"
"?a1 : 0 == False"
"(?P1 -> ?Q1) <-> ?P1 -> ?R1 == ?P1 -> ?Q1 <-> ?R1"
"(?P1 -> ?R1) <-> ?Q1 -> ?R1 == (?P1 <-> ?Q1) | ?R1"
"~?P1 <-> ~?Q1 == ?P1 <-> ?Q1"
"?P1 <-> ?P1 == True"
"?P1 <-> False == ~?P1"
"?Q <-> True == ?Q"
"False <-> ?P1 == ~?P1"
"True <-> ?Q == ?Q"
"?A1 <= ?A1 == True"
"0 <= ?A1 == True"
"cons(?a1, ?B1) = 0 == False"
"succ(?m1) = succ(?n1) == ?m1 = ?n1"
"succ(?n1) = 0 == False"
"?a1 = ?a1 == True"
"0 = cons(?a2, ?B2) == False" [.]
"0 = succ(?n2) == False" [.]
"?A1 Int 0 == 0" [.]
"0 Int ?A1 == 0" [.]
"?y Un 0 == ?y" [.]
"0 Un ?y == ?y" [.]
"(?P1 -> ?Q1) | ?R1 == ?P1 -> ?Q1 | ?R1"
"(?P1 | ?Q1) | ?R1 == ?P1 | ?Q1 | ?R1"
"~?P1 | ?P1 == True"
"?Q1 | (?P1 -> ?R1) == ?P1 -> ?Q1 | ?R1"
"?P1 | ?P1 | ?Q1 == ?P1 | ?Q1"
"?P1 | ~?P1 == True"
"?Q | ?Q == ?Q"
```

```
"?Q | False == ?Q"
"?P1 | True == True"
"False | ?Q == ?Q"
"True | ?P1 == True"
"ALL x. ?P1 | ?Q1(x) == ?P1 | (ALL x. ?Q1(x))"
"ALL x. ?P1(x) | ?Q1 == (ALL x. ?P1(x)) | ?Q1"
"ALL x. ?P1 -> ?Q1(x) == ?P1 -> (ALL x. ?Q1(x))"
"ALL x. ?P1(x) -> ?Q1 == (EX x. ?P1(x)) -> ?Q1"
"ALL x. ?P1 & ?Q1(x) == ?P1 & (ALL x. ?Q1(x))"
"ALL x. ?P1(x) & ?Q1 == (ALL x. ?P1(x)) & ?Q1"
"ALL x. ?t1 = x -> ?P1(x) == ?P1(?t1)"
"ALL x. x = ?t1 -> ?P1(x) == ?P1(?t1)"
"ALL x. ?Q == ?Q"
"EX x. ?t1 = x == True"
"EX x. ?P1 -> ?Q1(x) == ?P1 -> (EX x. ?Q1(x))"
"EX x. ?P1(x) -> ?Q1 == (ALL x. ?P1(x)) -> ?Q1"
"EX x. ?P1 | ?Q1(x) == ?P1 | (EX x. ?Q1(x))"
"EX x. ?P1(x) | ?Q1 == (EX x. ?P1(x)) | ?Q1"
"EX x. ?P1 & ?Q1(x) == ?P1 & (EX x. ?Q1(x))"
"EX x. ?P1(x) & ?Q1 == (EX x. ?P1(x)) & ?Q1"
"EX x. ?t1 = x & ?P1(x) == ?P1(?t1)"
"EX x. x = ?t1 & ?P1(x) == ?P1(?t1)"
"EX x. x = ?t1 == True"
"EX x. ?Q == ?Q"
"INT x:?C1. ?A1 Un ?B1(x) ==
if ?C1 = 0 then 0 else ?A1 Un (INT x:?C1. ?B1(x))"
[.]
"INT x:?C1. cons(?a1, ?B1(x)) ==
if ?C1 = 0 then 0 else cons(?a1, INT x:?C1. ?B1(x))"
[.]
"INT x:?C1. ?A1 - ?B1(x) ==
if ?C1 = 0 then 0 else ?A1 - (UN x:?C1. ?B1(x))"
[.]
"INT x:?C1. ?A1 Int ?B1(x) == ?A1 Int (INT x:?C1. ?B1(x))" [.]
"INT x:?C1. ?A1(x) Un ?B1 ==
if ?C1 = 0 then 0 else (INT x:?C1. ?A1(x)) Un ?B1"
[.]
"INT x:?C1. ?A1(x) - ?B1 == (INT x:?C1. ?A1(x)) - ?B1" [.]
"INT x:?C1. ?A1(x) Int ?B1 == (INT x:?C1. ?A1(x)) Int ?B1" [.]
"Inter({?y}) == ?y" [.]
"~(ALL x:?A1. ?P1(x)) == EX x:?A1. ~?P1(x)"
"~(EX x:?A1. ?P1(x)) == ALL x:?A1. ~?P1(x)"
"~(?P1 & ?Q1) == ~?P1 | ~?Q1"
"~(?P1 -> ?Q1) == ?P1 & ~?Q1"
"~(?P1 | ?Q1) == ~?P1 & ~?Q1"
"~(ALL x. ?P1(x)) == EX x. ~?P1(x)"
"~(EX x. ?P1(x)) == ALL x. ~?P1(x)"
```

```
"~~?Q == ?Q"
"~False == True"
"~True == False"
"THE x. ?y = x == ?y" [.]
"THE x. x = ?y == ?y" [.]
"UN x:RepFun(?A1, ?f1). ?B1(x) == UN a:?A1. ?B1(?f1(a))" [.]
"UN z:UN x:?A1. ?B1(x). ?C1(z) == UN x:?A1. UN z:?B1(x). ?C1(z)" [.]
"UN x:Union(?A1). ?B1(x) == UN y:?A1. UN x:y. ?B1(x)" [.]
"UN x:?C1. ?A'1 - ?B1(x) ==
if ?C1 = 0 then 0 else ?A'1 - (INT x:?C1. ?B1(x))"
[.]
"UN x:?C1. ?A1(x) - ?B'1 == (UN x:?C1. ?A1(x)) - ?B'1" [.]
"UN x:?C1. ?A'1 Int ?B1(x) == ?A'1 Int (UN x:?C1. ?B1(x))" [.]
"UN x:?C1. ?A1(x) Int ?B'1 == (UN x:?C1. ?A1(x)) Int ?B'1" [.]
"UN x:?C1. ?A'1 Un ?B1(x) ==
if ?C1 = 0 then 0 else ?A'1 Un (UN x:?C1. ?B1(x))"
[.]
"UN x:?C1. ?A1(x) Un ?B'1 ==
if ?C1 = 0 then 0 else (UN x:?C1. ?A1(x)) Un ?B'1"
[.]
"UN x:?C1. cons(?a1, ?B1(x)) ==
if ?C1 = 0 then 0 else cons(?a1, UN x:?C1. ?B1(x))"
[.]
"Union(cons(?b1, ?A1)) == ?b1 Un Union(?A1)" [.]
"Union(0) == 0" [.]
"(!!x. PROP ?V) == PROP ?V"
simplification procedures:
defined ALL:
ALL x. ?P(x)
defined BALL:
ALL x:?A. ?P(x) -> ?Q(x)
defined BEX:
EX x:?A. ?P(x) & ?Q(x)
defined EX:
EX x. ?P(x)
congruences:
"[| ?P == ?P'; ?P' ==> ?Q == ?Q' |] ==> ?P -> ?Q == ?P' -> ?Q'"
"[| ?A == ?A'; !!x. x : ?A' ==> ?P(x) == ?P'(x) |]
==> ALL x:?A. ?P(x) == ALL x:?A'. ?P'(x)"
"[| ?A == ?A'; !!x. x : ?A' ==> ?P(x) == ?P'(x) |]
==> EX x:?A. ?P(x) == EX x:?A'. ?P'(x)"
"[| ?A == ?B; !!x y. x : ?B ==> ?P(x, y) == ?Q(x, y) |]
==> Replace(?A, ?P) == Replace(?B, ?Q)"
"[| ?A == ?B; !!x. x : ?B ==> ?f(x) == ?g(x) |]
==> RepFun(?A, ?f) == RepFun(?B, ?g)"
"[| ?A == ?B; !!x. x : ?B ==> ?P(x) == ?Q(x) |]
==> {x: ?A . ?P(x)} == {x: ?B . ?Q(x)}"
```

```
"?P == ?Q ==> if ?P then ?x else ?y == if ?Q then ?x else ?y"
val it = () : unit
```

Bibliografia

[1] Paul Halmos,. Naive Set Theory. Van Nostrand, 1968.

[2] R. Cori and D. Lascar,. Mathematical Logic: A Course With Exercises - Recursion Theory, Godel's Theorems, Set Theory, Model Theory. Oxford University Press, 2001.

[3] J.-L. Krivine,. Introduction to Axiomatic Set Theory. D. Reidel Publishing Company, 1971.

[4] F. M. Dionísio, P. Gouveia, and J. Marcos,. Defining and using deductive systems with Isabelle. In L. Magnani and R. Dossena (eds.) (2005), Computing, Philosophy and Cognition, King's College Publications, London (Proceedings of the conference E-CAP2004).

[5] P. Gouveia and F. M. Dionísio,. Lógica Computacional (Computational Logic). Preprint, CLC, Department of Mathematics, Instituto Superior Técnico, 1049-001 Lisboa, Portugal, 2006, (Submitted to IST Press).

[6] J. Carmo, Elementos de Matemática do Discreto, Texto de Apoio à Disciplina de Estruturas Discretas, Departamento de Matemática e Engenharias, Universidade da Madeira, 2005.

[7] L. C. Paulson,. Isabelle's Logics FOL and ZF. Available as part of Isabelle's documentation. 2003

[8] L. C. Paulson,. The Isabelle Reference Manual, Computer Laboratory University of Cambridge, 2004.

[9] L. C. Paulson,. Isabelle - A Generic Theorem Prover. Springer-Verlag, 1994.