

**Exercício 55.** Apresente derivações em *Dedução Natural* para os sequentes a seguir assumindo que  $x$  não ocorre livre em  $\psi$ , e em seguida classifique cada prova como *minimal*, *intuicionista* ou *clássica*.

1.  $(\forall x \phi) \wedge \psi \vdash \forall x (\phi \wedge \psi)$
2.  $(\exists x \phi) \wedge \psi \vdash \exists x (\phi \wedge \psi)$
3.  $\forall x (\psi \rightarrow \phi) \vdash \psi \rightarrow \forall x \phi$
4.  $\forall x (\phi \rightarrow \psi) \vdash (\exists x \phi) \rightarrow \psi$

**Exercício 56.** Prove que não existe uma derivação intuicionista para os sequentes a seguir:

1.  $\neg \exists x \neg \varphi \vdash \forall x \varphi$
2.  $\neg \forall x \neg \varphi \vdash \exists x \varphi$
3.  $\forall x \neg \neg \varphi \vdash \neg \neg \forall x \varphi$

Assim como a LP, a LPO é correta e completa, mas estes resultados não serão provados aqui (Veja, por exemplo, [4]).

### 2.2.1 Indução

Indução é uma técnica de prova muito poderosa que desempenha um papel fundamental tanto em Matemática quanto em Computação. Estudaremos esta técnica partindo dos conjuntos definidos indutivamente, mas o leitor interessado em se aprofundar no tema pode consultar, por exemplo, os livros [12, 19]. Um conjunto, digamos,  $A$ , é *definido indutivamente* se seus elementos podem ser construídos a partir de um conjunto finito de regras de inferência da forma:

$$\frac{}{a \in A} \qquad \frac{a_1 \in A \quad a_2 \in A \dots a_n \in A}{a \in A}$$

A regra da esquerda é um axioma, e diz que  $a$  é um elemento do conjunto  $A$ , enquanto que na regra da direita temos que se  $a_1, a_2, \dots, a_n$  são elementos de  $A$  então  $a$  também é um elemento de  $A$ . Por exemplo, qualquer conjunto finito pode ser definido indutivamente com um axioma para cada elemento. De fato, o conjunto *Sem* dos dias da semana pode ser definido indutivamente via 7 axiomas:

$$\begin{array}{ccc}
\frac{}{\text{domingo} \in Sem} \text{ (DOM)} & \frac{}{\text{segunda-feira} \in Sem} \text{ (SEG)} & \frac{}{\text{terça-feira} \in Sem} \text{ (TER)} \\
\frac{}{\text{quarta-feira} \in Sem} \text{ (QUA)} & \frac{}{\text{quinta-feira} \in Sem} \text{ (QUI)} & \frac{}{\text{sexta-feira} \in Sem} \text{ (SEX)} \\
& & \frac{}{\text{sábado} \in Sem} \text{ (SAB)}
\end{array}$$

Alternativamente, podemos utilizar uma notação mais compacta definir o conjunto  $Sem$ : se  $d$  é uma variável que representa um elemento qualquer de  $Sem$  então a gramática a seguir é equivalente à definição via as 7 regras de inferência apresentadas acima:

$$d ::= \text{domingo} \mid \text{segunda-feira} \mid \text{terça-feira} \mid \text{quarta-feira} \mid \text{quinta-feira} \mid \text{sexta-feira} \mid \text{sábado}$$

O conjunto  $bool$ , muito popular em Ciência da Computação, possui apenas dois elementos:

$$\frac{}{\text{true} \in bool} \text{ (TRUE)} \quad \frac{}{\text{false} \in bool} \text{ (FALSE)}$$

Alternativamente, se  $b$  denota um valor booleano, podemos descrever o conjunto  $bool$  pela gramática:

$$b ::= \text{true} \mid \text{false}$$

Mas conjuntos definidos indutivamente também podem ser infinitos. O exemplo mais conhecido provavelmente é o conjunto dos números naturais  $\mathbb{N}$ , que pode ser definido pelas regras de inferência a seguir:

$$\frac{}{0 \in \mathbb{N}} \quad \frac{n \in \mathbb{N}}{S n \in \mathbb{N}}$$

Neste caso, a regra da esquerda é um axioma que diz que o 0 é um número natural, enquanto que a regra da direita diz que se  $n$  é um número natural então  $S n$  (o sucessor de  $n$ ) também é um número natural. A gramática equivalente a estas duas regras é dada por:

$$n ::= 0 \mid S n \tag{2.6}$$

Agora que sabemos o que são conjuntos definidos indutivamente podemos voltar ao tema da indução, que é uma técnica de prova que nos permite provar propriedades de conjuntos definidos indutivamente. Como provar que os elementos de um conjunto definido indutivamente, digamos  $A$ , satisfazem uma dada propriedade  $P$ ? Se o conjunto  $A$  for finito então podemos testar individualmente se cada elemento satisfaz a propriedade  $P$ . Mesmo que  $A$  seja um conjunto grande, depois de uma quantidade finita de tempo teremos uma prova de que os elementos de  $A$  satisfazem a propriedade  $P$ . E se o conjunto  $A$  for infinito? A ideia é bastante intuitiva: suponha que os elementos deste conjunto possam ser colocados um após o outro como peças de um dominó, de tal forma que, se uma peça qualquer for derrubada então a peça que está logo em seguida também é derrubada. Então podemos concluir, que se a primeira peça for derrubada então **todas** as outras serão derrubadas. Ou seja, voltando ao contexto de propriedades de elementos de um conjunto, a ideia é provar que se um elemento arbitrário do conjunto satisfaz a propriedade então o próximo elemento também satisfaz a propriedade. Se esta prova puder ser feita juntamente com a prova de que o primeiro elemento do conjunto também satisfaz a propriedade então podemos concluir que todos os elementos do conjunto satisfazem a propriedade.

Vamos iniciar este estudo sobre indução no contexto dos números naturais, onde esta noção de ordem é bem clara: o primeiro elemento é o 0, em seguida vem o 1, depois o 2, etc. De uma forma geral, depois de um número natural  $k$  vem o natural  $S k$ , o sucessor de  $k$  que também escrevemos como  $k + 1$ . A indução no contexto dos números naturais é conhecida como *indução matemática*, e será explorada na próxima seção.

A gramática (2.6) possui dois construtores: 0 e  $S$ . O primeiro diz que 0 é um número natural, e o segundo diz que a partir de um natural já construído, digamos  $n$ , podemos construir um outro natural, a saber,  $S n$ , ou seja, o sucessor de  $n$ . Muito bem, agora considere uma propriedade qualquer dos números naturais. Por exemplo, a que diz que a soma dos  $n$  primeiros números ímpares é igual a  $n^2$ . Como podemos provar esta propriedade? Isto mesmo, por indução! O que diz mesmo o princípio de indução para os números naturais? Diz que se uma propriedade  $P$  vale para 0 (base da indução), e se, supondo que  $P$  vale para um natural arbitrário  $k$  (hipótese de indução), podemos provar que ela vale também para  $S k$  (o sucessor de  $k$ )<sup>9</sup> (passo indutivo) então podemos concluir que  $P$  vale para todos os números naturais. Esquemáticamente, podemos apresentar este princípio, denominado *Princípio da Indução Matemática (PIM)*, como a seguir:

$$\frac{P 0 \quad \forall k, P k \implies P (S k)}{\forall n, P n} \text{ (PIM)}$$

<sup>9</sup>Note que o sucessor de  $k$  pode ser escrito como  $S k$  ou  $k + 1$ .

**Exemplo 21.** Queremos provar que a soma dos  $n$  primeiros números ímpares é igual a  $n^2$ . Esta propriedade vale trivialmente para o 0 (a soma dos 0 primeiros números ímpares é igual a  $0^2$ ). Agora suponha que a soma dos  $k$  primeiros números ímpares seja igual a  $k^2$  (hipótese de indução). O  $(k + 1)$ -ésimo número ímpar é igual a  $2k + 1$  (por que?), e portanto a soma dos  $k + 1$  primeiros números ímpares é  $k^2 + 2k + 1 = (k + 1)^2$ , como queríamos provar.

Uma outra forma de resolver este problema em um contexto mais formal pode ser feita a partir de uma definição formal da soma dos  $n$  primeiros números ímpares por meio do somatório  $\sum_{i=1}^n (2i - 1)$ , que por definição é igual a 0, se  $n = 0$ . Queremos provar que  $\sum_{i=1}^n (2i - 1) = n^2$ , para todo número natural  $n$ . Aplicando o princípio da indução, teremos 2 casos para analisar:

- **(Base da indução):** A base da indução se dá quando  $n = 0$ , e é trivial porque o lado esquerdo da igualdade é igual a 0 por definição.
- **(Passo indutivo):** O passo indutivo é a parte interessante de qualquer prova por indução. Neste caso específico, vamos assumir que a propriedade que queremos provar vale para um número natural arbitrário, digamos  $k$ , e provaremos que esta propriedade continua valendo para o natural  $k + 1$ . Ou seja, assumimos que  $\sum_{i=1}^k (2i - 1) = k^2$ , e vamos provar que  $\sum_{i=1}^{k+1} (2i - 1) = (k + 1)^2$ . Partindo do lado esquerdo desta igualdade, podemos decompor o somatório da seguinte forma  $\sum_{i=1}^{k+1} (2i - 1) = \sum_{i=1}^k (2i - 1) + (2k + 1)$ , e agora podemos utilizar a hipótese de indução (h.i.) para assim chegarmos ao lado direito da igualdade:  $\sum_{i=1}^{k+1} (2i - 1) = \sum_{i=1}^k (2i - 1) + (2k + 1) \stackrel{h.i.}{=} k^2 + (2k + 1) = (k + 1)^2$ .

Por fim, apresentamos esta prova na forma de árvore:

$$\begin{array}{c}
 \frac{[\sum_{i=1}^k (2i - 1) = k^2]^u}{\sum_{i=1}^k (2i - 1) + (2k + 1) = (k + 1)^2} \\
 \frac{\sum_{i=1}^k (2i - 1) + (2k + 1) = (k + 1)^2}{\sum_{i=1}^{k+1} (2i - 1) = (k + 1)^2} \quad (\rightarrow_i) u \\
 \frac{\sum_{i=1}^k (2i - 1) = k^2 \rightarrow \sum_{i=1}^{k+1} (2i - 1) = (k + 1)^2}{\sum_{i=1}^0 (2i - 1) = 0^2} \\
 \frac{\sum_{i=1}^0 (2i - 1) = 0^2}{\sum_{i=1}^n (2i - 1) = n^2} \quad (\text{Ind. em } n)
 \end{array}$$

Existem propriedades que valem apenas para um subconjunto próprio dos números naturais:

Por exemplo,  $2^n < n!$  só vale para  $n \geq 4$ . Para este tipo de problema utilizamos uma generalização do PIM onde a base de indução não precisa ser o 0. Chamaremos esta variação de *Princípio da Indução Generalizado (PIG)*:

$$\frac{P m \quad \forall k, P k \implies P (S k)}{\forall n, n \geq m \implies P n} \text{ (PIG)}$$

**Exemplo 22.** Prove que  $2^n < n!, \forall n \geq 4$ .

1. (Base de indução) A propriedade vale para  $n = 4$ , o que é trivial, e;
2. (Passo indutivo) Mostraremos que  $2^{(S k)} < (S k)!$  assumindo que  $2^k < k!, \forall k \geq 4$ . De fato, temos que  $2^{(S k)} = 2 \cdot 2^k \stackrel{(h.i)}{<} 2 \cdot k! \stackrel{(*)}{<} (S k) \cdot k! = (S k)!$ , onde a desigualdade (\*) se justifica pelo fato de  $k$  ser maior ou igual a 4.

Uma variação do PIM bastante útil é conhecida como *Princípio da Indução Forte (PIF)*:

$$\frac{\forall k, (\forall m, m < k \implies P m) \implies P k}{\forall n, P n} \text{ (PIF)}$$

**Exercício 57.** Prove que qualquer inteiro  $n \geq 2$  é um número primo ou pode ser escrito como um produto de primos (não necessariamente distintos), i.e. na forma  $n = p_1 \cdot p_2 \cdot \dots \cdot p_r$ , onde os fatores  $p_i$  ( $1 \leq i \leq r$ ) são primos.

**Exercício 58.** Mostre que PIM e PIF são princípios equivalentes.

**Exercício 59.** Prove que a soma dos  $n$  primeiros números naturais é igual a  $\frac{n(n+1)}{2}$ . Ou seja, mostre que  $\sum_{i=1}^n i = \frac{n(n+1)}{2}$ .

**Exercício 60.** Prove que a soma dos  $n$  primeiros quadrados é igual a  $\frac{n(n+1)(2n+1)}{6}$ . Ou seja, mostre que  $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$ .

**Exercício 61.** Prove que  $\sum_{i=0}^n i(i+1) = \frac{n \cdot (n+1) \cdot (n+2)}{3}$ .

**Exercício 62.** Prove que  $\sum_{i=0}^n 2^i = 2^{n+1} - 1$ .

**Exercício 63.** Prove que a soma dos  $n$  primeiros cubos é igual ao quadrado da soma de 1 até  $n$ , ou seja, que  $1^3 + 2^3 + \dots + n^3 = (1 + 2 + \dots + n)^2$ .

**Exercício 64.** Prove que  $2^n - 1$  é múltiplo de 3, para todo número natural  $n$  par.

**Exercício 65.** Prove que  $3 \mid (2^{2n} - 1)$  para todo  $n \geq 0$ .

**Exercício 66.** Prove que  $3^n \geq n^2 + 3$  para todo  $n \geq 2$ .

**Exercício 67.** Prove que  $n^2 < 4^{n-1}$  para todo  $n \geq 3$ .

**Exercício 68.** Prove que  $n! > 3^n$  para todo  $n \geq 7$ .

**Exercício 69.** Prove que  $n! \leq n^n$  para todo  $n \geq 1$ .

## Indução no assistente de provas Coq

Nesta seção, veremos como construir provas utilizando a indução matemática no assistente de provas Coq (<https://coq.inria.fr>). Um conjunto indutivamente definido pode ser construído com a palavra reservada `Inductive`. Por exemplo, o conjunto finito *Sem* dos dias da semana apresentado na seção anterior pode ser definido como a seguir:

```
Inductive Sem :=
| domingo: Sem
| segunda-feira: Sem
| terca-feira: Sem
| quarta-feira: Sem
| quinta-feira: Sem
| sexta-feira: Sem
| sabado: Sem.
```

O conjunto dos números naturais é uma construção nativa do Coq, *i.e.* ela está disponível uma vez que o sistema é iniciado, e pode ser vista a partir do comando `Print nat`.

```
Inductive nat : Set := 0 : nat | S : nat -> nat.
```

Ou seja, o conjunto `nat` dos números naturais é definido indutivamente e possui 2 construtores: o `0` (zero), e `S` (sucessor). Claramente, esta definição corresponde à gramática (2.6). Toda definição indutiva possui um princípio indutivo associado, e que é automaticamente gerado pelo Coq. Por padrão, o nome do princípio indutivo associado a uma definição indutiva, digamos `mdef`, é `mdef_ind`. No caso de `nat` podemos acessar este princípio pelo comando `Print nat_ind`:

```
nat_ind =
fun (P : nat -> Prop) (f : P 0) (f0 : forall n : nat, P n -> P (S n)) =>
fix F (n : nat) : P n := match n as n0 return (P n0) with
| 0 => f
| S n0 => f0 n0 (F n0)
end
:forall P: nat -> Prop, P 0 ->
(forall n: nat, P n -> P (S n)) ->
forall n: nat, P n
```

A parte que nos interessa desta saída está em azul. Como em (*PIM*), a base de indução diz que `P 0`, e o passo indutivo corresponde ao trecho `(forall n : nat, P n -> P (S n))`. A conclusão como esperado, diz que `forall n : nat, P n`.

A seguir faremos a prova de que a soma dos  $n$  primeiros números ímpares é igual a  $n^2$ . Inicialmente precisamos expressar a "soma dos  $n$  primeiros números ímpares" em Coq. Para isto, definiremos um somatório. Antes disto carregamos a biblioteca *Lia*, que vai nos ajudar com a simplificação de expressões aritméticas nos inteiros.

```

Require Import Lia.

Fixpoint msum (n:nat) :=
  match n with
  | 0 => 0
  |S k => (msum k) + (2*k+1)
  end.

```

A palavra reservada `Fixpoint` é utilizada para definir funções recursivas. Note que  $\sum_{i=1}^n (2.i - 1)$  corresponde a `msum n`. Podemos fazer alguns testes com esta definição:

```

Eval compute in (msum 1).

```

```

= 1
: nat

```

O valor retornado é 1 porque que é igual ao primeiro número ímpar.

```

Eval compute in (msum 2).

```

```

= 4
: nat

```

Aqui a resposta é igual a 4 porque corresponde à soma dos dois primeiros números ímpares, ou seja, 1+3.

```

Eval compute in (msum 3).

```

```

= 9
: nat

```

O valor retornado corresponde à soma dos 3 primeiros números ímpares: 1+3+5=9.

```

Eval compute in (msum 4).

```

```

= 16
: nat

```

Por fim, temos que a soma dos 4 primeiros números ímpares é 1+3+5+7=16.

De acordo com estes testes, nossa definição de somatório está funcionando corretamente, e portanto podemos escrever o lema que queremos provar, a saber, que a soma dos `n` primeiros números naturais é igual a `n*n`:

```

Lemma msum_square: forall n, msum n = n*n.

```



Faremos a mesma prova apresentada na árvore construída na seção anterior. Iniciamos a prova por indução em  $n$  com a tática (ou comando) `induction n`.

```
Lemma msum_square: forall n, msum n = n*n.
Proof.
  induction n.
```

```
2 goals (ID 11)

=====
msum 0 = 0 * 0

goal 2 (ID 14) is:
msum (S n) = S n * S n
```

Temos 2 casos para analisar (2 `goals`): o primeiro corresponde a base da indução, e o segundo é o passo indutivo.

O primeiro caso é trivial, e a tática `reflexivity` é capaz de concluir que os lados esquerdo e direito da igualdade são iguais a 0, uma vez que `msum 0` é igual a 0.

No segundo caso, temos como hipótese de indução que a soma dos  $k$  primeiros números ímpares é igual a  $k*k$ , e precisamos provar que a soma dos  $(S k)$  primeiros números ímpares é igual a  $(S k)*(S K)$ :

```
1 goal (ID 14)

k : nat
IHk : msum k = k * k
=====
msum (S k) = S k * S k
```

Podemos, por exemplo, aplicar a tática `simpl` para simplificar a expressão `msum (S k)`, ou seja, para aplicarmos a definição de `msum`. Agora podemos substituir o lado esquerdo da hipótese de indução pelo lado direito via o comando `rewrite IHn`. A expressão resultante é uma igualdade envolvendo somas e multiplicações de números naturais:

```
1 goal (ID 24)

k : nat
IHk : msum k = k * k
=====
k * k + (k + (k + 0) + 1) = S (k + k * S k)
```

As simplificações algébricas necessárias para que possamos concluir que os lados esquerdo e direito da igualdade coincidem são feitas pela tática `lia`, e a prova completa, disponível no arquivo `pim.v`<sup>10</sup>, tem a seguinte forma:

<sup>10</sup><http://flaviomoura.info/files/lca/pim.v>. Os exercícios propostos na seção anterior também estão disponíveis neste arquivo.

```

Lemma msum_square: forall n, msum n = n*n.
Proof.
  induction n.
  - reflexivity.
  - simpl. rewrite IHn. lia.
Qed.

```

Agora vamos estabelecer a equivalência entre PIM e o PIG:

**Exercício 70.** *Complete a prova a seguir:*

```

Lemma PIG: forall (P : nat -> Prop) (k : nat), P k ->
  (forall n, n >= k -> P n -> P (S n)) ->
  forall n : nat, n >= k -> P n.
Proof.
  intros P k H1 IH n H2.
  assert (H := nat_ind (fun n => n >= k -> P n)).
  Admitted.

```

Observe que a prova do exercício anterior utiliza o PIM via o comando `nat_ind`, e portanto temos uma prova de PIG via PIM. No outro sentido, vamos enunciar PIM como um lema:

**Exercício 71.** *Complete a prova a seguir:*

```

Lemma PIM : forall P: nat -> Prop,
  (P 0) ->
  (forall n, P n -> P (S n)) ->
  forall n, P n.
Proof.
  intros P H IH n.
  apply PIG with 0.
  Admitted.

```

## Indução Estrutural

Nesta seção veremos que o princípio de indução matemática (PIM) visto anteriormente é um caso particular de um princípio geral que está associado a qualquer conjunto definido indutivamente. Vimos dois tipos de regras utilizadas na construção de um conjunto definido indutivamente:

1. As regras não recursivas, ou seja, aquelas que definem diretamente um elemento do conjunto definido indutivamente;
2. As regras recursivas, ou seja, aquelas que constroem novos elementos a partir de elementos já construídos.

Como veremos no próximo exemplo, estas regras podem fazer uso de elementos de outros conjuntos previamente definidos. Formalmente, se  $A_1, A_2, \dots$  são conjuntos então a estrutura geral das regras de

um conjunto definido indutivamente  $B$  é como a seguir:

1. Inicialmente temos as regras não recursivas que definem diretamente os elementos  $b_1, \dots, b_m$  de  $B$ :

$$\frac{a_1 \in A_1 \quad a_2 \in A_2 \dots a_{j_1} \in A_{j_1}}{b_1[a_1, \dots, a_{j_1}] \in B} \quad \dots \quad \frac{a_1 \in A_1 \quad a_2 \in A_2 \dots a_{j_m} \in A_{j_m}}{b_m[x_1, \dots, x_{j_m}] \in B}$$

2. Em seguida, temos as regras recursivas que constroem novos elementos a partir de elementos já construídos:

$$\frac{a_1 \in A_1 \dots a_{j'_1} \in A_{j'_1} \quad d_1, \dots, d_{k_1} \in B}{c_1[x_1, \dots, x_{j'_1}, d_1, \dots, d_{k_1}] \in B} \quad \dots \quad \frac{a_1 \in A_1 \dots a_{j'_n} \in A_{j'_n} \quad d_1, \dots, d_{k_n} \in B}{c_n[a_1, \dots, a_{j'_n}, d_1, \dots, d_{k_n}] \in B}$$

Qualquer elemento de um conjunto definido indutivamente pode ser construído após um número finito de aplicações das regras que o definem (e somente com estas regras). Os elementos  $d_1, d_2, \dots, d_{k_i}$  são ditos *estruturalmente menores* do que o elemento  $c_i[a_1, \dots, a_{j_i}, d_1, \dots, d_{k_i}]$ . Isto significa que os elementos  $d_1, d_2, \dots, d_{k_i}$  são subtermos próprios de  $c_i[a_1, \dots, a_{j_i}, d_1, \dots, d_{k_i}]$ .

Podemos associar um princípio de indução a qualquer conjunto definido indutivamente. No contexto genérico acima, teremos um caso base (base da indução) para cada regra não recursiva, e um passo indutivo para cada regra recursiva. O esquema simplificado (omitindo os parâmetros por falta de espaço) tem a seguinte forma:

$$\frac{\overbrace{P(b_1) \dots P(b_m)}^{\text{casos base}} \quad \overbrace{(\forall d_1 \dots d_{k_1}, P(d_1), \dots, P(d_{k_1}) \Rightarrow P(c_1)) \dots (\forall d_1 \dots d_{k_n}, P(d_1), \dots, P(d_{k_n}) \Rightarrow P(c_n))}^{\text{casos indutivos}}}{\forall x \in B, P x}$$

Retornando ao caso do conjunto dos números naturais, temos um princípio indutivo com apenas um caso base e um caso indutivo:

$$\frac{P 0 \quad \forall k, P k \Rightarrow P (S k)}{\forall n, P n}$$

O conjunto dos booleanos possui um princípio indutivo com dois casos base, e nenhum caso indutivo:

$$\frac{P \text{ true} \quad P \text{ false}}{\forall b, P b}$$

A gramática 2.2 nos diz como as fórmulas da LP podem ser construídas. Observe, em particular, seus os construtores recursivos: por exemplo, a negação de uma fórmula é construída a partir de outra

fórmula já construída; a conjunção, a disjunção e a implicação são construídas a partir de duas fórmulas previamente construídas. Podemos derivar o princípio de indução para o conjunto das fórmulas da LP de maneira análoga aos casos anteriores, ou seja, como um caso particular da generalização apresentada acima. Como temos dois construtores não recursivos (variáveis proposicionais e a constante  $\perp$ ) e quatro construtores recursivos (negação, conjunção, disjunção e implicação), o princípio indutivo correspondente terá a seguinte forma, considerando uma propriedade  $Q$  qualquer das fórmulas da LP:

$$\frac{(Q \ p) \quad (Q \ \perp) \quad (\forall \varphi, Q \ \varphi \implies Q \ (\neg \varphi)) \quad (\forall \varphi_1, Q \ \varphi_1 \wedge \forall \varphi_2, Q \ \varphi_2 \implies Q \ (\varphi_1 \star \varphi_2))}{\forall \varphi, Q \ \varphi}$$

onde  $\star \in \{\wedge, \vee, \rightarrow\}$ . Chamamos o princípio de indução construído a partir de uma gramática recursiva de *indução estrutural*. Note que, para cada um dos conectivos binários (conjunção, disjunção e implicação) temos duas hipóteses de indução.

No exemplo a seguir, vamos mostrar que a gramática acima possui redundâncias, isto é, que existem conectivos que podem ser escritos a partir de outros:

**Exemplo 23.** *Prove, sem utilizar tabela de verdade, que para qualquer fórmula  $\varphi$ , existe uma fórmula  $\varphi'$  equivalente a  $\varphi$  construída apenas com os conectivos  $\vee$  e  $\neg$ , e com os símbolos proposicionais que ocorrem em  $\varphi$ .*

*Dizemos que duas fórmulas  $\varphi$  e  $\psi$  da LP são equivalentes se  $\varphi \leftrightarrow \psi$  é uma tautologia. Provaremos este exercício por indução estrutural, isto é, indução na estrutura de  $\varphi$ :*

- *Se  $\varphi$  é uma variável proposicional ou a constante  $\perp$  então tome  $\varphi' = \varphi$ .*
- *Se  $\varphi = \neg \psi$  então, por hipótese de indução, existe uma fórmula  $\psi'$  equivalente a  $\psi$  construída apenas com os conectivos  $\vee$  e  $\neg$ , e os símbolos proposicionais que ocorrem em  $\psi$ . Neste caso, basta tomar  $\varphi' = \neg \psi'$ , e estamos prontos.*
- *Se  $\varphi = \psi_1 \vee \psi_2$  então, por hipótese de indução, existem fórmulas  $\psi'_i (i = 1, 2)$ , equivalentes respectivamente a  $\psi_i (i = 1, 2)$ , e construídas apenas com os conectivos  $\vee$  e  $\neg$ , e os símbolos proposicionais que ocorrem em  $\psi_i (i = 1, 2)$ . Neste caso, basta tomar  $\varphi' = \psi'_1 \vee \psi'_2$  e estamos prontos.*
- *Se  $\varphi = \psi_1 \wedge \psi_2$  então, por hipótese de indução, existem fórmulas  $\psi'_i (i = 1, 2)$ , equivalentes respectivamente a  $\psi_i (i = 1, 2)$ , e construídas apenas com os conectivos  $\vee$  e  $\neg$ , e os símbolos proposicionais que ocorrem em  $\psi_i (i = 1, 2)$ . Pelo exercício 2.1.6 sabemos que  $\psi_1 \wedge \psi_2 \dashv\vdash \neg(\neg \psi_1 \vee \neg \psi_2)$ . Então basta tomar  $\varphi' = \neg(\neg \psi'_1 \vee \neg \psi'_2)$ , e estamos prontos.*
- *Por fim, se  $\varphi = \psi_1 \rightarrow \psi_2$  então, por hipótese de indução, existem fórmulas  $\psi'_i (i = 1, 2)$ , equivalentes respectivamente a  $\psi_i (i = 1, 2)$ , e construídas apenas com os conectivos  $\vee$  e  $\neg$ , e os símbolos proposicionais que ocorrem em  $\psi_i (i = 1, 2)$ . Pelo exercício 2.1.6 da lista sabemos que  $\psi_1 \rightarrow \psi_2 \dashv\vdash (\neg \psi_1) \vee \psi_2$ . Então basta tomar  $\varphi' = (\neg \psi'_1) \vee \psi'_2$  e estamos prontos.*

Agora é a sua vez! Resolva os exercícios a seguir:

**Exercício 72.** Prove, sem utilizar tabela de verdade, que para qualquer fórmula  $\varphi$ , existe uma fórmula  $\varphi'$  equivalente a  $\varphi$  construída apenas com os conectivos  $\rightarrow$  e  $\neg$ , e com os símbolos proposicionais que ocorrem em  $\varphi$ .

**Exercício 73.** Prove, sem utilizar tabela de verdade, que para qualquer fórmula  $\varphi$ , existe uma fórmula  $\varphi'$  equivalente a  $\varphi$  construída apenas com os conectivos  $\wedge$  e  $\neg$ , e com os símbolos proposicionais que ocorrem em  $\varphi$ .

Considere a gramática da LPO:

$$\varphi ::= p(t, \dots, t) \mid \perp \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid \exists_x \varphi \mid \forall_x \varphi$$

o princípio de indução correspondente é dado como a seguir:

$$\frac{(\forall t_1, \dots, t_n, Q p(t_1, \dots, t_n)) \quad (Q \perp) \quad (\forall \varphi, Q \varphi \implies Q (\neg\varphi)) \quad (*) \quad (**)}{\forall \varphi, Q \varphi}$$

onde

(\*) é igual a  $(\forall \varphi_1, Q \varphi_1 \wedge \forall \varphi_2, Q \varphi_2 \implies Q (\varphi_1 \star \varphi_2))$ ,  $\star \in \{\wedge, \vee, \rightarrow\}$ ;

(\*\*) é igual a  $(\forall x, \varphi, Q \varphi(x) \implies Q (R_x \varphi(x)))$ ,  $R \in \{\exists, \forall\}$ .

**Exercício 74.** Seja  $\varphi$  uma fórmula da lógica de predicados. Definimos a tradução negativa de Gödel-Gentzen de  $\varphi$ , denotada por  $\varphi^N$ , indutivamente por:

$$\varphi^N = \begin{cases} \neg\neg\varphi & \text{se } \varphi \text{ é uma fórmula atômica, ou a constante } \perp \\ \neg(\psi^N) & \text{se } \varphi = \neg\psi \\ \varphi_1^N \wedge \varphi_2^N & \text{se } \varphi = \varphi_1 \wedge \varphi_2 \\ \neg(\neg(\varphi_1^N) \wedge \neg(\varphi_2^N)) & \text{se } \varphi = \varphi_1 \vee \varphi_2 \\ \varphi_1^N \rightarrow \varphi_2^N & \text{se } \varphi = \varphi_1 \rightarrow \varphi_2 \\ \forall_x(\psi^N) & \text{se } \varphi = \forall_x \psi \\ \neg(\forall_x \neg(\psi^N)) & \text{se } \varphi = \exists_x \psi \end{cases}$$

Construa uma prova intuicionista para o sequente a seguir:  $\neg\neg(\varphi^N) \vdash_i \varphi^N$

**Exercício 75.** Uma fórmula da lógica de predicados  $\phi$  pertence ao fragmento negativo se  $\phi$  pode ser construída a partir da seguinte gramática, onde  $t_1, t_2, \dots, t_n$  ( $n > 0$ ) são termos:

$$\phi ::= \neg p(t_1, t_2, \dots, t_n) \mid \perp \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \rightarrow \phi) \mid (\forall_x \phi)$$

Prove na lógica minimal que  $\neg\neg\theta \vdash_m \theta$  para qualquer fórmula  $\theta$  pertencente ao fragmento negativo. Use indução na estrutura de  $\theta$ .

Nos próximos capítulos estudaremos diversos algoritmos que utilizam a estrutura de lista encadeada, definida pela seguinte gramática  $l ::= nil \mid a :: l$ , onde  $nil$  representa a lista vazia, e  $a :: l$  representa a lista com primeiro elemento  $a$  e cauda  $l$ . Como esta gramática possui um construtor não recursivo, e um construtor recursivo, teremos um princípio de indução com um caso base, e um passo indutivo:

$$\frac{P \text{ nil} \quad \forall l, h, P l \implies P (h :: l)}{\forall l, P l}$$

O comprimento de uma lista, isto é, o número de elementos que a lista possui, é definido recursivamente por:

$$|l| = \begin{cases} 0, & \text{se } l = nil \\ 1 + |l'|, & \text{se } l = a :: l' \end{cases}$$

Uma operação importante que nos permite construir uma nova lista a partir de duas listas já construídas é a concatenação. Podemos definir a concatenação de duas listas por meio da seguinte função recursiva:

$$l_1 \circ l_2 = \begin{cases} l_2, & \text{se } l_1 = nil \\ a :: (l' \circ l_2), & \text{se } l_1 = a :: l' \end{cases}$$

Por fim, o reverso de uma lista é definido recursivamente por:

$$rev(l) = \begin{cases} l, & \text{se } l = nil \\ (rev(l')) \circ (a :: nil), & \text{se } l = a :: l' \end{cases}$$

Os exercícios a seguir expressam diversas propriedades envolvendo estas operações. Resolva cada um deles utilizando indução.

**Exercício 76.** Prove que  $|l_1 \circ l_2| = |l_1| + |l_2|$ , quaisquer que sejam as listas  $l_1, l_2$ .

**Exercício 77.** Prove que  $l \circ nil = l$ , qualquer que seja a lista  $l$ .

**Exercício 78.** Prove que a concatenação de listas é associativa, isto é,  $(l_1 \circ l_2) \circ l_3 = l_1 \circ (l_2 \circ l_3)$  quaisquer que sejam as listas  $l_1, l_2$  e  $l_3$ .

**Exercício 79.** Prove que  $|\text{rev}(l)| = |l|$ , qualquer que seja a lista  $l$ .

**Exercício 80.** Prove que  $\text{rev}(l_1 \circ l_2) = (\text{rev}(l_2)) \circ (\text{rev}(l_1))$ , quaisquer que sejam as listas  $l_1, l_2$ .

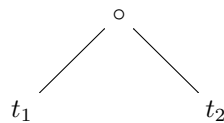
**Exercício 81.** Prove que  $\text{rev}(\text{rev}(l)) = l$ , qualquer que seja a lista  $l$ .

Outra estrutura de dados importante em Computação é a estrutura de árvores. O caso particular do conjunto *btree* das árvores binárias, isto é, as árvores cujos nós têm dois filhos, ou são folhas (não têm filhos) pode ser definido indutivamente pelas seguintes regras:

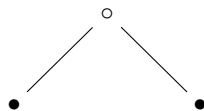
$$\frac{}{\bullet \in \text{btree}} \qquad \frac{t_1 \in \text{btree} \quad t_2 \in \text{btree}}{\circ(t_1, t_2)}$$

A gramática correspondente é dada por  $t ::= \bullet \mid \circ(t, t)$ .

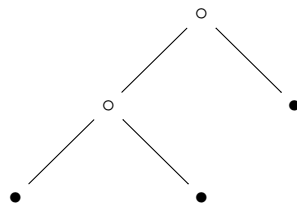
Assim, um nó sem filhos representa uma árvore (caso não recursivo), e  $t_1$  e  $t_2$  são duas árvores binárias então podemos construir uma nova árvore como na figura abaixo:



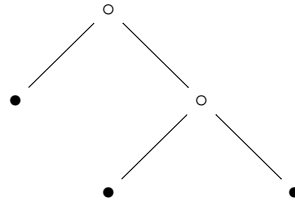
Observe que a árvore



é escrita como  $\circ(\bullet, \bullet)$  na sintaxe da regra recursiva acima. Enquanto que



corresponde a  $\circ(\circ(\bullet, \bullet), \bullet)$ . Ou ainda,



corresponde a  $o(\bullet, o(\bullet, \bullet))$ .

O princípio de indução sobre *btree* terá um caso base, e um caso indutivo com duas hipóteses de indução. Nesta representação a raiz da árvore está no topo, e as folhas ficam para baixo (como se a árvore estivesse de cabeça para baixo), mas veremos outras situações em que a raiz fica na parte inferior, e as folhas ficam no topo da árvore (como ocorre na natureza). As duas representações são utilizadas em Computação, como veremos.

Podemos definir a altura de uma árvore binária da seguinte forma:

$$h(t) = \begin{cases} 0, & \text{se } t = \bullet \\ 1 + \max(h(t_1), h(t_2)), & \text{se } t = o(t_1, t_2) \end{cases}$$

O número de nós de uma árvore binária é dado por:

$$n(t) = \begin{cases} 1, & \text{se } t = \bullet \\ 1 + n(t_1) + n(t_2), & \text{se } t = o(t_1, t_2) \end{cases}$$

**Exercício 82.** *Mostre que  $n(t) \leq 2^{h(t)+1} - 1$ , para qualquer árvore binária  $t$ .*

Podemos flexibilizar um pouco a definição de árvore binária e permitir que um nó tenha, no máximo, dois filhos. Neste caso, acrescentaremos mais uma regra à nossa definição:

$$\frac{}{\bullet \in btree} \qquad \frac{t \in btree}{o(t)} \qquad \frac{t_1 \in btree \quad t_2 \in btree}{o(t_1, t_2)}$$

A gramática correspondente é dada por  $t ::= \bullet \mid o(t) \mid o(t, t)$ .

Podemos estender esta definição para árvores cujos nós possuem até  $k \geq 0$  filhos, mas na prática ficaremos restritos a  $k = 3$  por conta da estrutura das regras do sistema de dedução natural tanto na lógica proposicional quanto na lógica de predicados. Como exemplo, provaremos o Teorema de Glivenko:

**Exemplo 24.** *O teorema de Glivenko diz que se  $\Gamma \vdash_c \varphi$  então  $\Gamma \vdash_i \neg\neg\varphi$  na lógica proposicional, ou seja, se  $\varphi$  tem uma prova clássica a partir de  $\Gamma$ , então  $\neg\neg\varphi$  tem uma prova intuicionista a partir de  $\Gamma$  na lógica proposicional. Faremos a prova deste teorema por indução na derivação  $\Gamma \vdash_c \varphi$ , isto é, indução na estrutura da árvore correspondente à derivação clássica de  $\varphi$  a partir de  $\Gamma$ . Queremos provar  $\Gamma \vdash_i \neg\neg\varphi$ , quais quer que sejam  $\Gamma$  e  $\varphi$ .*