4 O Mecanismo de Prova – outros tópicos

4.1 As conectivas que faltam II

A forma como o **Coq** trata as conectivas habituais da Lógica, além das básicas *implicação* e *quantificação universal*, passa pela utilização de tipos indutivos. A sua utilização pode no entanto ser feita de forma *naïf*, antes do estudo detalhado daqueles tipos.

Quando no objectivo de uma prova ocorre uma fórmula composta por uma das conectivas $\wedge$, ou $\vee$, é possível utilizar tácticas específicas para tratar a sua introdução.[11] A aplicação destas tácticas é semelhante à das regras de introdução, em Dedução Natural: `Split` corresponde à regra de introdução da conjunção, e `Left` e `Right` às duas regras de introdução da disjunção.

Por outro lado, quando numa hipótese aberta no estado de prova corrente ocorre uma fórmula construída por uma das conectivas referidas, ou ainda por $\perp$, o mecanismo de prova a utilizar está intimamente ligado à utilização de tipos indutivos mas pode ser entendido relembrando as definições de segunda ordem das ditas conectivas:

$$\begin{aligned} \perp &\doteq \forall A.\ A \\ P \wedge Q &\doteq \forall A.\ (P \to Q \to A) \to A \\ P \vee Q &\doteq \forall A.\ (P \to A) \to (Q \to A) \to A \end{aligned}$$

A validade destas igualdades permite à táctica `Elim`:

- Dada uma hipótese $\perp$, provar trivialmente qualquer objectivo.
- Dada uma hipótese da forma $P \wedge Q$, substituir qualquer objectivo A por um outro $P \to Q \to A$.
- Dada uma hipótese da forma $P \vee Q$, substituir qualquer objectivo A por dois outros, $P \to A$ e $Q \to A$.

A táctica `Elim` é de facto mais poderosa do que isto. Ela permite, por exemplo, dada uma hipótese $R \to P \wedge Q$, substituir, por eliminação dessa hipótese, qualquer objectivo A por dois outros, R e $P \to Q \to A$.

Comecemos por uma prova extremamente simples que envolve a eliminação de uma hipótese negada:

```
Coq < Variables P, Q, R, S : Prop.
Coq < Theorem absurd: S /\ ~S  -> Q.
absurd < Intro H.
absurd < Elim H.
absurd < Intros H0 H1.
absurd < Unfold not in H1.
absurd < Elim H1.
```

O último comando efectuou eliminação da hipótese $S \to \perp$. De facto, uma vez que do absurdo se pode inferir a validade de todas as fórmulas, qualquer objectivo de prova pode ser substituído por $\perp$, e este por S, dada a presença da hipótese referida. De notar que não teria sido necessário

[11] Note-se que à conectiva $\perp$, que também não é primitiva em **Coq**, não está associada qualquer regra de introdução.

efectuar a reescrita de $\neg S$ como $S \rightarrow \bot$. Recuemos dois passos na prova:

```
absurd < Undo 2.
absurd < Elim H1.
absurd < Assumption.
absurd < Save.
```

Vejamos agora uma prova de um teorema envolvendo conjunção e disjunção (observe-se a sintaxe concreta `/\` e `\/` para os operadores `and` e `or` respectivamente):

```
Coq < Theorem conj_disj : P/\Q -> P\/Q.
conj_disj < Intro H.
conj_disj < Elim H.
conj_disj < Clear H.
```

Frequentemente, uma hipótese, depois de eliminada, não mais volta a ser usada, pelo que pode ser abandonada. É o que se passa no caso corrente. O comando `Clear` permite apagar uma hipótese do contexto actual.

```
conj_disj < Intros H H0.
conj_disj < Left.
conj_disj < Assumption.
conj_disj < Save.
Coq < Print Proof conj_disj.
```

Note-se que em vez da táctica `Left` se poderia ter usado `Right`, uma vez que ambos os "lados" da disjunção existem como hipóteses.

O próximo exemplo utiliza a táctica `Split`:

```
Coq < Theorem pc : (P -> Q) /\ (R -> S) -> (P /\ R -> Q /\ S).
pc < Intros H H0.
pc < Elim H; Clear H; Elim H0; Clear H0.
pc < Intros H H0 H1 H2.
pc < Split.
pc < Apply H1; Assumption.
pc < Apply H2; Assumption.
pc < Save.
Coq < Check pc.
```

Questão 6A. Construa em **Coq** uma prova para o teorema

$$\neg(Q \vee R) \to (\neg Q \wedge \neg R)$$

Questão 6B. Construa em **Coq** uma prova para o teorema

$$(\neg Q \wedge \neg R) \to \neg(Q \vee R)$$

O Quantificador Existencial

Também o quantificador existencial está definido com recurso a um tipo indutivo, sendo no entanto possível a sua utilização de forma simples.

O tipo do quantificador é aquele com que é costume codificar-se qualquer quantificador: $(A \to *) \to *$ para um determinado tipo A. De facto, ele deve construir uma fórmula, a partir de um predicado. Se for por exemplo $P : A \to *$, então será $(\exists P) : *$, uma frase lógica, assim como $(\exists(\lambda x\colon A.Px)) : *$. Esta segunda representação será a que utilizaremos, e a razão para isso compreende-se muito facilmente: imagine-se um predicado binário $R\colon A \to A \to *$. A quantificação existencial em qualquer das variáveis argumentos do predicado pode ser feita muito facilmente (no contexto $\{A : *\}$) como $\exists(\lambda x\colon A.\ Rxy)$ e $\exists(\lambda y\colon A.\ Rxy)$, respectivamente.

```
Coq < Check ex.
```

Em **Coq**, o operador `ex` tem o tipo acima, mas quantificado ainda no tipo A. A representação de $\exists x.\ Cx$, com $C\colon U \to *$, será feita como se mostra:

```
Coq < Variable U : Set.
Coq < Variable C : U -> Prop.
Coq < Check (ex U [x:U](C x)).
```

Existe no entanto a sintaxe concreta `Ex`, que dispensa o primeiro argumento:

```
Coq < Check (Ex [x:U](C x)).
```

`ex` pode ser eliminado, como as restantes conectivas, pela táctica `Elim`, e introduzido no objectivo de prova com a táctica especial `Exists`.

Provemos então o teorema $\exists x.\ (A(x) \to B(x)) \to \forall x.\ A(x) \to \exists x.\ B(x)$:

```
Coq < Variable D : Set.
Coq < Variables A, B : D -> Prop.
Coq < Goal (Ex [x:D]((A x) -> (B x))) ->
Coq <         ((x:D)(A x)) -> (Ex [x:D](B x)).
Unnamed_thm < Intros H H0.
Unnamed_thm < Elim H; Clear H.
Unnamed_thm < Intros y H.
```

O próximo passo é o mais importante: temos como hipóteses $\forall x.A(x)$ e $A(y) \to B(y)$, estando a variável $y : D$ também presente no contexto actual. Então já podemos fornecer o *testemunho* do objectivo actual $\exists x.\ B(x)$: trata-se de y.

```
Unnamed_thm < Exists y
Unnamed_thm < Apply H.
Unnamed_thm < Apply H0.
Unnamed_thm < Abort.
```

Questão 10A. Prove o teorema

$$(\exists x.\neg A(x)) \to \neg(\forall x.A(x))$$

Questão 10B. Prove o teorema

$$\neg(\exists x.A(x)) \to (\forall x.\neg A(x))$$

Exercício: Averigue se a frase $\forall x.\exists y.P(x,y) \to \exists y.\forall x.P(x,y)$ é um teorema da Lógica de Primeira Ordem.

4.2 Comandos de Gestão da Prova

Temos vindo a utilizar alguns comandos indispensáveis no processo de construção interactiva de uma prova. Sistematizemos a função de cada um:

`Show` Este comando permite visualizar o estado de prova corrente, com todos os seus objectivos.

`Clear` Permite remover uma hipótese do contexto corrente. A hipótese não poderá ser usada no futuro.

`Restart` Permite o regresso ao estado de prova inicial, ou seja, anula o efeito de todos os passos de prova efectuados.

`Abort` Utilizado sem argumento, este comando permite cancelar a construção de prova em que se trabalha actualmente. Quando se edita várias provas em simultâneo, recebe um argumento que descrimina qual a que se pretende abortar.

`Undo` Cancela o último passo de prova efectuado. Com um argumento numérico `n`, repete `n` vezes o seu efeito.

`Suspend` / `Resume` O primeiro comando tem como efeito abandonar temporariamente a edição da prova corrente, regressando-se ao *toplevel* do **Coq**, e o segundo permite retomar aquela edição. Sem argumento permite regressar à última prova em que se trabalhou; o argumento opcional permite especificar qual a prova a que se deseja regressar.

`Focus` / `Unfocus` O primeiro comando tem como efeito focar a prova no primeiro sub-objectivo a provar; os restantes sub-objectivos não são apresentados no ecran (embora existam). O segundo comando desfaz o efeito de `Focus`.

4.3 Automatização da construção de provas, Lógica Clássica, Táctica de *corte*, e Combinadores de tácticas

Um sistema dedutivo para a Lógica Clássica pode ser facilmente obtido de um sistema dedutivo Intuicionista,[12] bastando para isso adicionar a este um axioma clássico, como seja $\neg\neg A \to A$, que não é mais do que outra forma de escrever o chamado *princípio da redução ao absurdo*, $(\neg A \to \bot) \to A$. Outro axioma equivalente será ainda o *princípio do terceiro excluído*, $A \vee \neg A$.

Comecemos por iniciar uma nova secção onde se considera válido o princípio de redução ao absurdo. Provemos que o princípio do terceiro excluído é um teorema:

```
Coq < Section Classical.
Coq < Hypothesis raa: (A:Prop) ~~A -> A.
Coq < Theorem tnd : (A:Prop) A \/ ~A.
tnd < Intro A0.
tnd < Apply raa.
tnd < Red.
tnd < Intro.
tnd < Elim H.
tnd < Left.
tnd < Apply raa.
tnd < Red.
tnd < Intro.
tnd < Elim H.
tnd < Right.
tnd < Assumption.
```

Dada a dificuldade de construção desta prova em retrocesso, devida à necessidade de se "adivinhar"

[12]Note-se que a analogia proposições como tipos coloca qualquer subsistema do Cálculo de Construções em correspondência com uma determinada lógica construtiva.

quando aplicar a regra RAA, e como obter o absurdo, justifica-se a apresentação da árvore de prova em Dedução Natural (mp e $\rightarrow$ são as regras de eliminação e introdução da implicação, e não se faz referência explícita às aplicações da definição da negação):

$$
\begin{prooftree}
\AxiomC{$[\neg A]$}
\RightLabel{$Right$}
\UnaryInfC{$A \vee \neg A$}
\AxiomC{$[\neg(A \vee \neg A)]$}
\RightLabel{mp}
\BinaryInfC{$\perp$}
\RightLabel{raa}
\UnaryInfC{A}
\RightLabel{$Left$}
\UnaryInfC{$A \vee \neg A$}
\AxiomC{$[\neg(A \vee \neg A)]$}
\RightLabel{mp}
\BinaryInfC{$\perp$}
\RightLabel{raa}
\UnaryInfC{$A \vee \neg A$}
\end{prooftree}
$$

Observe-se agora a seguinte árvore de prova alternativa:

$$
\begin{prooftree}
\AxiomC{$[A]$}
\RightLabel{$Left$}
\UnaryInfC{$A \vee \neg A$}
\AxiomC{$[\neg(A \vee \neg A)]$}
\RightLabel{mp}
\BinaryInfC{$\perp$}
\RightLabel{$\rightarrow$}
\UnaryInfC{$\neg A$}
\AxiomC{$[\neg A]$}
\RightLabel{$Right$}
\UnaryInfC{$A \vee \neg A$}
\AxiomC{$[\neg(A \vee \neg A)]$}
\RightLabel{mp}
\BinaryInfC{$\perp$}
\RightLabel{$\rightarrow$}
\UnaryInfC{$\neg\neg A$}
\RightLabel{mp}
\BinaryInfC{$\perp$}
\RightLabel{raa}
\UnaryInfC{$A \vee \neg A$}
\end{prooftree}
$$

Como reproduzir este raciocínio alternativo em **Coq**?

```
tnd < Restart.
tnd < Intro A0.
tnd < Apply raa.
tnd < Red.
tnd < Intro.
```

Esta parte inicial da prova é idêntica. Mas segue-se um passo não trivial: O objectivo $\perp$ deve agora ser substituído pelos dois objectivos $\neg A$ e $\neg A \rightarrow \perp$. A táctica `Cut` permite em geral substituir o objectivo V por $U \rightarrow V$ e U, mediante o comando `Cut U`:[13]

```
tnd < Cut ~A0.
tnd < Intro H0.
tnd < Elim H.
tnd < Right; Assumption.
tnd < Red.
tnd < Intro H0.
tnd < Elim H.
tnd < Left; Assumption.
```

Não vamos ainda abandonar esta prova. Antes disso, utilizemo-la para demonstrar a utilização da táctica `Auto` de automatização do processo de construção de prova. Esta táctica sistematiza

[13] Uma outra táctica útil é `Absurd`: O comando `Absurd U` substitui qualquer objectivo de prova pelos dois objectivos U e $\neg U$.

algumas sequências correntes, como sejam aplicações de `Intro` e `Assumption`, e ainda `Apply` de algumas hipóteses, nomeadamente as de introdução das conectivas, como `Left` ou `Split`. A táctica não faz qualquer aplicação de `Elim`, nem tenta adivinhar testemunhos para a táctica `Exists`.

```
tnd < Restart.
tnd < Intro A0.
tnd < Apply raa.
tnd < Red.
tnd < Intro.
tnd < Cut ~A0.
tnd < Auto.
tnd < Red.
tnd < Auto.
```

A prova acima pode ser resolvida com uma única táctica, construída com os combinadores `;` e `[ ... | ... | ... ]`. O primeiro permite sequenciar aplicações de tácticas, enquanto o segundo é útil na aplicação de tácticas que gerem múltiplos objectivos de prova. Assim, por exemplo, a táctica composta `t1; [t21 | t22 | t23]` consiste na aplicação de `t21` ao primeiro objectivo gerado por `t1`, `t21` ao segundo, e `t23` ao terceiro. A táctica composta `t1; t2` consiste na aplicação da táctica `t1` ao objectivo corrente e depois a táctica `t2` a todos os sub-objectivos gerados por `t1`.

```
tnd < Restart.
tnd < Intro A0; Apply raa; Red; Intro;
tnd <     Cut ~A0; [Auto | Red; Auto].
```

A prova pode ainda ser um pouco mais automatizada do que se mostrou acima. De facto, se se começar por remover todas as ocorrências da negação, a táctica `auto` consegue atacar a prova mais cedo:

```
tnd < Restart.
tnd < Unfold not; Unfold not in raa; Intro; Apply raa; Auto.
tnd < Save.
Coq < Check tnd.
```

Provamos pois que o princípio do terceiro excluído é derivado na Lógica Clássica, mas observe-se agora o efeito do seguinte:

```
Coq < End Classical.
Coq < Check tnd.
```

Ao fechar a secção onde se assumiu como válido o princípio da redução ao absurdo, o sistema abstraiu o termo `tnd` numa prova daquele princípio, como seria de esperar. O teorema provado por este termo é intuicionista! Este efeito poderia ter sido evitado se se tivesse declarado globalmente `raa` com `Axiom` em vez de localmente com `Hypothesis`.

Efectuemos agora a prova em sentido contrário: do princípio do terceiro excluído podemos inferir a redução ao absurdo:

```
Coq < Section Classical2.
Coq < Hypothesis te : (X:Prop) X \/ ~X.
Coq < Theorem raa : (Y:Prop) ~~Y -> Y.
raa < Intros Y H.
raa < Elim te.
```

Obtivemos uma mensagem de erro ao tentar eliminar a disjunção presente em `te`. De facto, a táctica `Elim` necessita de conhecer a instância específica daquela hipótese que se pretende eliminar:

```
raa < Elim te with Y.
```

Surgem dois objectivos, dos quais o primeiro pode ser imediatamente provado, mas o segundo necessita de alguma ajuda da nossa parte:

```
raa < Auto.
raa < Intro H0.
raa < Elim H.
raa < Assumption.
```

Ou numa só táctica:

```
raa < Restart.
raa < Intros Y H; Elim te with Y;
raa <      [Auto | Intro H0; Elim H; Assumption].
raa < Save.
```

Vejamos finalmente um exemplo de uma prova clássica em Lógica de Predicados. Efectuaremos a prova na secção corrente, tendo como hipótese o princípio do terceiro excluído. O teorema que desejamos provar é $(\forall x.\neg A(x)) \vee (\exists x.A(x))$.

```
Coq < Theorem dq : ((x:D) ~(A x)) \/ (Ex [x:D] (A x)).
```

Esta construção de prova seguirá o padrão típico de “análise de casos”, cuja utilização só é possível com o princípio do terceiro excluído. Começamos por eliminar a instância deste axioma relativa à validade de $\exists x.A(x)$ ou da sua negação:

```
dq < Elim (te (Ex [x:D] (A x))).
```

O sistema gera dois objectivos de prova. O primeiro pode ser facilmente atingido, uma vez que a hipótese introduzida coincide com o lado direito da disjunção. A táctica `auto` consegue tratar este primeiro objectivo. O segundo objectivo terá de ser atingido via o lado esquerdo da disjunção: provaremos para um x arbitrário $\neg A(x)$, a partir de $\neg\exists x.A(x)$:

```
dq < Auto.
dq < Intro H.
dq < Left.
dq < Intro x.
```

Os passos seguintes consistem, depois da reescrita do objectivo da prova (que tem a forma de uma negação) na eliminação da negação acima representada pela hipótese `H`, e na nomeação do testemunho que verifica $\exists x.A(x)$:

```
dq < Red; Intro.
dq < Elim H.
dq < Exists x.
dq < Exact H0.
```

Reproduzimos a prova usando uma táctica composta:

```
dq < Restart.
dq < Elim (te (Ex [x:D] (A x)));
dq <   [Auto | Intro H; Left; Intro x; Red; Intro; Elim H;
dq <           Exists x; Exact H0].
dq < Save.
Coq < End Classical2.
```

Exercício: Construa provas dos seguintes teoremas clássicos , usando (alternadamente) os axiomas de redução ao absurdo e do terceiro excluído.

1. $(A \to B) \to (\neg A \vee B)$.
2. $\neg\exists x.\neg A(x) \to \forall x.A(x)$.

4.4 Tácticais

Tácticais são operadores sobre tácticas que dão a possibilidade de descrever estratégias de prova de maneira mais sintética.

Para além dos operadores ; e [...|...|...] o Coq disponibiliza mais alguns tácticais. Passemos a apresentar, em resumo, os tácticais existentes:

`Idtac` Esta táctica não faz nada (o que pode ser util, como veremos).

`Do` *num tac* Repete *num* vezes a aplicação da tática *tac*.

tac_1 `Orelse` tac_2 Tenta aplicar a táctica tac_1 e caso esta falhe aplica tac_2.

`Repeat` *tac* Repete a aplicação da táctica *tac*, enquanto ela não falha.

tac_1 **;** tac_2 Sequêncializa da aplicação de tácticas. Aplica tac_1 e em seguida aplica tac_2 a todos os objectivos erados pela táctica tac_1.

tac **;** [$tac_1|...|tac_n$] Paraleliza a aplicação de tácticas. Aplica tac_i ao i-ezimo sub-objectivo gerado por *tac*.

`Try` *tac* Tenta aplicar a táctica *tac*, se não conseguir não dá erro. Portanto nunca falha.

Assim por exemplo:

- `Do 2 Intro.` é equivalente a `Intro. Intro.`
- `Intros.` tem o mesmo efeito de `Repeat Intro.`
- `Try` *tac.* é equivalente a *tac* `Orelse Idtac.`

4.5 Algumas notas sobre a *igualdade* em Coq

A existência de um predicado universal[14] de igualdade em **Coq** leva-nos a reflectir um pouco sobre quais devem ser as propriedades de tal predicado. Além de ter de se tratar obrigatoriamente de uma relação de equivalência (reflexiva, simétrica e transitiva), tem de verificar ainda duas propriedades:

1. a de substituição, que afirma que se dois termos t e t' são iguais, e σ é uma substituição das suas variáveis, então tem de ser ainda $\sigma(t) = \sigma(t')$. Esta propriedade é automaticamente verificada em ferramentas de prova, pela utilização da unificação.
2. a de congruência, em relação a todos os operadores definidos na assinatura de trabalho.

[14] no sentido em que é válida para todos os `Sets`.

Em **Coq**, em vez de se encontrarem axiomatizados todos estes princípios, a igualdade é definida por um axioma de ordem superior:

$$x = y \to \forall P.\ P(x) \to P(y)$$

Este axioma permite derivar todos os outros, e leva a que a forma preferencial de se lidar com a igualdade em **Coq** seja por utilização de tácticas especiais de **reescrita**. Assim, se o objectivo actual de prova for $P(y)$ e existir uma hipótese $H : y = x$, podemos invocar a táctica `Rewrite` com argumento H para substituir aquele objectivo por $P(x)$. Se a hipótese fosse $H : x = y$, a táctica a utilizar deveria ser `Rewrite <-`, que permite que a equação seja utilizada da direita para a esquerda.

Como ilustração deste facto, demonstremos a propriedade de transitividade:

```
Coq < Theorem trans : (A:Set)(x,y,z:A)x=y->y=z->x=z.
trans < Intros A x y z H H0.
```

Podemos agora efectuar duas reescritas diferentes, correspondentes a cada uma das hipóteses (equações). Note-se que cada uma delas deve ser aplicada num sentido diferente:

```
trans < Rewrite H.
trans < Undo.
trans < Rewrite <- H0.
trans < Auto.
trans < Abort.
```

Os três axiomas definidores das propriedades da igualdade como relação de equivalência, apesar de poderem ser deduzidos desta forma, encontram-se também disponíveis através de três tácticas, `Reflexivity`, `Symmetry`, e `Transitivity`. Esta igualdade, conhecida por **Igualdade de Leibniz** é definid em Coq com recurso a um tipo indutivo[15]

O sistema Coq quando arranca carrega, por defeito, uma serie de definições, entre as quais a dos números naturais. Observe, atentamente o resultado dos seguintes comandos:

```
Coq < Check nat.
Coq < Check O.
Coq < Check S.

Coq < Check (S (S O)).
```

Vejamos mais uns exemplos. Seja `f` uma função sobre naturais que aplicada a zero dá como resultado zero:

[15] De facto, toda a concepção do sistema Coq se encontra intimamente ligada aos tipos indutivos.

```
Coq < Variable f : nat -> nat.
Coq < Hypothesis foo : (f O) = O.
```

Provemos então que $\forall n.\ n = O \rightarrow f(n) = n$

```
Coq < Lemma L1 : (n:nat) n=O -> (f n)=n.
L1 < Intros.
L1 < Rewrite H.
L1 < Apply foo.
L1 < Qed.
```

Provemos ainda que $f(f(O)) = O$

```
Coq < Lemma L2 : (f (f O))=O.
L2 < Rewrite foo.
L2 < Rewrite foo.
L2 < Reflexivity.
```

Em seguida, apresentamos mais duas maneiras alternativas de de provar este mesmo lema:

```
L2 < Restart.
L2 < Rewrite foo.
L2 < Apply foo.

L2 < Restart.
L2 < Replace (f O) with O.
L2 < Apply foo.
L2 < Symmetry.
L2 < Apply foo.

L2 < Save.
```