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.[10] 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 $\bot$, 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:

$$\begin{array}{rcl} \bot & \doteq & \forall A.\ A \\ P \wedge Q & \doteq & \forall A.\ (P \to Q \to A) \to A \\ P \vee Q & \doteq & \forall A.\ (P \to A) \to (Q \to A) \to A \end{array}$$

A validade destas igualdades permite à táctica `Elim`:

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

[10] Note-se que à conectiva $\bot$, 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).
Praeclarum < Intros H H0.
Praeclarum < Elim H; Clear H; Elim H0; Clear H0.
Praeclarum < Intros H H0 H1 H2.
Praeclarum < Split.
Praeclarum < Apply H1; Assumption.
Praeclarum < Apply H2; Assumption.
Praeclarum < Save.
Coq < Check pc.
```

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 $(\exists(\lambda x{:}\, A.Px)) : *$. Esta segunda representação será a que utilizaremos, e a razão para isso compreende-se muito facilmente: imagine-se um predicado binário $R{:}\, A \to A \to *$. A quantificação existencial em qualquer das variáveis argumentos do predicado pode ser feita muito facilmente (no contexto $\{A : *\}$) como $\exists(\lambda x{:}\, A.\ Rxy)$ e $\exists(\lambda y{:}\, A.\ Rxy)$, 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 $C{:}\, U \to *$, 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.
```

Exercício: Averigue se a frase $\forall x.\exists y.P(x,y) \rightarrow \exists y.\forall x.P(x,y)$ 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.

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,[11] bastando para isso adicionar a este um axioma clássico, como seja $\neg\neg A \rightarrow A$, que não é mais do que outra forma de escrever o chamado *princípio da redução ao absurdo*, $(\neg A \rightarrow \bot) \rightarrow A$. 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:

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

```
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" 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):

$$\dfrac{\dfrac{\dfrac{[\neg A]}{A \vee \neg A}\ Right \qquad [\neg(A \vee \neg A)]}{\dfrac{\bot}{A}\ mp}\ raa}{\dfrac{A \vee \neg A}{\bot}\ Left \qquad\qquad [\neg(A \vee \neg A)]}\ mp}{A \vee \neg A}\ raa$$

Observe-se agora a seguinte árvore de prova alternativa:

$$\dfrac{\dfrac{\dfrac{[A]}{A \vee \neg A}\ Left \qquad [\neg(A \vee \neg A)]}{\bot}\ mp}{\neg A}\ \rightarrow \qquad \dfrac{\dfrac{\dfrac{[\neg A]}{A \vee \neg A}\ Right \qquad [\neg(A \vee \neg A)]}{\bot}\ mp}{\neg\neg A}\ \rightarrow}{\bot}\ mp}{A \vee \neg A}\ raa$$

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 $\bot$ deve agora ser substituído pelos dois objectivos $\neg A$ e $\neg A \to \bot$. A táctica `Cut` permite em geral substituir o objectivo V por $U \to V$ e U, mediante o comando `Cut U`:[12]

```
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 algumas sequências correntes, como sejam aplicações de `Intro` e `Assumption`, a 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.

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

---

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

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.
```

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