O Sistema Coq

O Coq é um sistema de prova assistida para lógicas de ordem superior, que permite o desenvolvimento de programas de modo consistente com a sua especificação (programas certificados). Este sistema baseia-se no Cálculo de Construções Indutivas.

Para entrar no sistema Coq[1] faça:

```
coqtop
```

Vamos, portanto, trabalhar em λ-Calculus com tipos. Vejamos como se representam as expressões λ em Coq:

λ-Calculus	**Coq**
$\lambda x : A.t$	$[x : A]t$
$t\, s$	$(t\, s)$
$t\, s_1\, s_2 \ldots s_n$	$(t\, s_1\, s_2 \ldots s_n)$
$A \to B$	$A \to B$
$\Pi x : A.B$	$(x : A)B$

O Coq divide o universo proposicional, `*`, em duas categorias `Set` e `Prop` , consoante os seus habitantes são vistos como conjuntos concretos ou como proposições. O universo $\Box$ é representado em Coq por `Type`.

1 Declarações, Definições, Inferência de Tipos Simples e Redução de Termos.

1.1 Declarações locais e globais. O mecanismo de secções

Comecemos por efectuar algumas declarações globais. Cada uma declara um nome para um g-termo, associando-lhe um tipo. Declaremos, antes de mais, dois tipos proposicionais:

```
Coq < Variable A : Set.
Coq < Variable B : Set.
```

As declarações feitas com `Variable` são *locais*, ou seja, têm validade num contexto cujo início e fim o utilizador define. Em Coq, estes contextos tomam o nome de **secções**. Como não especificámos ainda nenhuma secção para a validade das declarações acima, elas são válidas *globalmente*, ou seja, em todos os contextos.

[1] Certifique-se que `/usr1/ML/ocaml/COQ/bin` se encontram na sua `PATH`. Caso tal não se verifique inclua o comando `export PATH=${PATH}:/usr1/ML/ocaml/COQ/bin` no fim do seu ficheiro `~/.profile` ou `~/.bashrc`.

Iniciemos agora um novo contexto, e declaremos nesse contexto um termo de um dos tipos acima.

```
Coq < Section s1.
Coq < Variable t : A.
```

Note que o tipo `A` é conhecido dentro da secção `s1`.

Apesar de estarmos dentro do contexto `s1` podemos efectuar declarações globais em qualquer momento, com o comando `Parameter`:

```
Coq < Parameter g : B.
```

1.2 Inferência de Tipos

Voltaremos brevemente ao mecanismo de secções, mas antes disso vejamos como efectuar em Coq a síntese do tipo de um termo. O comando `Check` tem precisamente essa função:

```
Coq < Check [x:A] x.
Coq < Check [x:A] [y:A->B] (y x).
```

Os dois termos cujo tipo pedimos ao sistema para inferir correspondem, na notação tradicional do λ-calculus, a $(\lambda x\colon A.x)$ e $(\lambda x\colon A.\lambda y\colon A \to B.yx)$. Note que se trata de termos do λ-calculus com tipos simples (designado $\lambda_{\to}$), e observe a forma como o Coq infere e representa os seus tipos. De facto, tratando-se de termos anotados com tipos, este processo é simples.

De notar ainda que as variáveis `x` e `y` usadas nas abstracções dos termos acima representados, sendo locais a essas abstracções, não necessitam de ser declaradas no contexto corrente. Se tentarmos:

```
Coq < Check x.
```

o sistema imprimirá uma mensagem de erro.

1.3 Definições locais e globais

Em seguida, vamos efectuar algumas *definições*. Nestas, podemos associar a um qualquer g-termo um nome, novamente de forma local ou global, respectivamente com os comandos `Local` e `Definition`.

```
Coq < Definition id_A := [x:A] x.
Coq < Local K_AB := [x:A] [y:B] x.
```

Sobre estes termos podemos efectuar a mesma operação de extracção do tipo, ou alternativamente utilizar o comando `Print` para visualizar o termo associado a um identificador:

```
Coq < Check id_A.
Coq < Print K_AB.
```

O comando `Print All.` permite visualizar informação sobre todos os identificadores declarados e definidos com visibilidade no contexto actual. O comando `Inspect n.` permite visualizar os `n` últimos identificadores declarados ou definidos. Naturalmente, no caso dos identificadores declarados, apenas o seu tipo está disponível.

```
Coq < Inspect 7.
```

1.4 Secções Revisitadas

Procedamos agora ao *fecho* da secção corrente:

```
Coq < End s1.
```

Questão 1.1. Qual o efeito do fecho da secção `s1`?

Avaliemos agora a seguinte sequência de comandos:

```
Coq < Section s2.
Coq < Variable x : A.
Coq < Local tlocal := [y:B] x.
Coq < Definition tglobal := [y:B] x.
Coq < Definition ttglobal := (tlocal g).
Coq < Print tlocal.
Coq < Print tglobal.
Coq < Print ttglobal.
Coq < End s2.
Coq < Print tlocal.
Coq < Print tglobal.
Coq < Print ttglobal.
```

A interpretação do papel das secções como contextos de juízos do sistemas de tipos é imediata: cada secção corresponde à sequência de declarações que forma o lado esquerdo de cada juízo de tipagem de um termo. Assim, o juízo

$$t_1{:}\,\sigma_1, t_2{:}\,\sigma_2, \ldots, t_i{:}\,\sigma_n \vdash t : \sigma.$$

será válido na teoria de tipos do Coq se numa secção onde sejam válidas apenas as declarações presentes no lado esquerdo do sinal $\vdash$, o comando `Check` aplicado ao termo t inferir o tipo σ para aquele termo.

O *fecho* de uma secção em Coq efectuará assim o fecho do λ-termo t, de acordo com a regra de abstracção dos sistemas de tipos.

Para concluir o estudo das secções, relembre a definição dos combinadores básicos K e S, e avalie a seguinte sequência de comandos

```
Coq < Section s3.
Coq < Variables sigma, tau, delta : Set.
Coq < Definition K_SigmaTau := [x:sigma][y:tau] x.
Coq < Definition S_SigmaTauDelta :=
                    [x:sigma->tau->delta][y:sigma->tau][a:sigma](x a (y a)).
Coq < Print K_SigmaTau.
Coq < Print S_SigmaTauDelta.
Coq < End s3.
Coq < Print K_SigmaTau.
Coq < Print S_SigmaTauDelta.
```

Questão 1.2. Comente o efeito, e a resposta do sistema, à sequência anterior.

1.5 Redução de Termos

Vejamos agora como poderemos efectuar em Coq a redução-β de termos do λ-calculus:

```
Coq < Variable b : A.
Coq < Definition f := (([a:A] a) b).
Coq < Eval f.
Coq < Eval (([a:A] a) b).
Coq < Compute f.

Coq < Variable a : A.
Coq < Eval (id_A a).
Coq < Compute (id_A a).

Coq < Eval ([C:Set]C A).
Coq < Definition ff := [C:Set][D:Set][x:C][y:C->D](y  x).
Coq < Check ff.
Coq < Compute (ff A B b).
```

O comando `Eval` efectua redução-β até à forma normal, sem no entanto proceder à substituição de identificadores pelos termos que eles representam. Para forçar essa substituição podemos utilizar o comando `Compute`.

1.6 Outros Exemplos

Observe que as seguintes tentativas de extracção de tipo e de redução falham, dada a má formação dos termos $(\lambda x\colon A.xx)$ e $(\lambda x\colon A.x)(\lambda x\colon A.x)$.[2]

```
Coq < Check ([x:A](x x)).
Coq < Check (([x:A]x)([x:A]x)).
Coq < Eval ([x:A]x [x:A]x).
```

Naturalmente, um termo para o qual não existe tipo na teoria do Coq é um termo mal formado, pelo que não pode ser reduzido.

A sequência de comandos que se segue também falha devido à mal formação dos termos envolvidos.

```
Coq < Check [x:A] t.
Coq < Check [y:A->C] (y a).
Coq < Eval (([y:A->C] y) a).
```

Note que os identificadores `t` e `C` não estão definidos no contexto.

[2]No λ-calculus sem tipos os termos resultantes destes depois de eleminadas as anotações de tipos são bem formados.

Questão 1.3. Considere a seguinte função do λ-calculus com tipos simples, que efectua a aplicação de uma função composta consigo mesma a um argumento:

$$double \doteq \lambda f\colon A \to A.\lambda x\colon A.f(fx)$$

Efectue em Coq a definição de um g-termo que represente esta função *double*, verifique qual é o seu tipo, e aplique-a com a função identidade (do tipo A) a um elemento de tipo A. Qual é o resultado?

Questão 1.4. Considere as seguintes definições Coq:

```
Definition exp1 := [x:A][y:B][C:Set][p:A->B->C] (p x y).
Definition exp2 := [A:Set][x:A][P:A->Set][y:(P x)] y.
Definition exp3 := [p:A->(C:Set)C][C:Set][a:A] (p a C).
```

Indique quais os g-termos que estão a ser definidos. Deduza o tipo de cada uma das expressões atrás definidas. Use o Coq para verificar a correcção das suas respostas.

Questão 1.5. Use o sistema Coq para o auxilar na resolução do seguinte problema.
Considere as seguites definições:

$$N \doteq \Pi C : *.C \to (C \to C) \to C$$

$$Z \doteq \lambda A : *.\lambda z : A.\lambda s : A \to A.z$$

$$S \doteq \lambda a : N.\lambda A : *.\lambda z : A.\lambda s : A \to A.s(aAzs)$$

$$soma \doteq \lambda b : N.\lambda a : N.\lambda A : *.\lambda z : A.\lambda s : A \to A.(bA)(aAzs)s$$

Deduza o seu tipo, e reduza à sua forma normal as expressões que se seguem:

i) SZ

ii) $S(SZ)$

iii) $soma\,(SZ)(S(SZ))$

Para sair do Coq faça:

```
Coq < Quit.
```