2 O λ-Cubo

As entidades bem formadas de uma Teoria de Tipos, são determinadas pelo conjunto de *universos* e pelo conjunto de *regras de inferência* que determinam quais os *juizos* $\Gamma \vdash t : T$ que são válidos[3]

Uma vez definidas estas duas classes de entidades, os termos generalizados (*g-termos*) bem formados ficam completamente determinados. Note-se que estes termos representam igualmente termos e tipos; portanto ficam determinados os termos e os tipos da Teoria de Tipos que estivermos a caracterizar.

Com um conjunto de dois universos $\{*, \Box\}$ e o conjunto de regras a seguir apresentadas, definem-se oito sistemas que podem ser organizados coerentemente nos vértices de um cubo.

Sistemas do λ-Cubo

Sejam $s, s_1, s_2 \in \{*, \Box\}$.

Regras Gerais

(axioma) $\quad \vdash * : \Box$

(assunção) $$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \;,\; x \notin \Gamma$$

(enfraquecimento) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \;,\; x \notin \Gamma$$

(aplicação) $$\frac{\Gamma \vdash F : (\Pi\, x{:}A.\, B) \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]}$$

(abstracção) $$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi\, x{:}A.\, B) : s}{\Gamma \vdash (\lambda x{:}A.b) : (\Pi\, x{:}A.\, B)}$$

(conversão) $$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_\beta B'}{\Gamma \vdash A : B'}$$

Regras específicas

regra $\langle s_1, s_2 \rangle$ $$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi\, x{:}A.\, B) : s_2}$$

<u>Nota</u>: $\quad (\Pi\, x{:}A.\, B) \;\equiv\; A \to B \;\;,\;\; \text{se} \;\; x \notin \mathcal{L}(B)$

A regra $\langle s_1, s_2 \rangle$ tem várias instâncias consoante s_1 e s_2 tomam o valor de $*$ ou $\Box$. Cada par de universos $\langle s_1, s_2 \rangle$ que determina uma regra de formação Π dá origem a uma *dependência*. Cada sistema é gerado pelas regras de inferência gerais e algumas regras específicas (de acordo com as dependências permitidas).

[3]Vendo o sistema de tipos como uma lógica, as suas proposições (fórmulas) são os juizos $\Gamma \vdash t : T$.

Os sistemas de tipos que iremos considerar contêm todos a dependência $\langle *, * \rangle$. Consoante contém ou não cada uma das restantes três dependências ($\langle *, \Box \rangle, \langle \Box, * \rangle$ e $\langle \Box, \Box \rangle$) identificam-se oito *Sistemas Abstractos de Tipos* diferentes, a que se dão nomes específicos (ver quadro).

Dependências

$\lambda_{\rightarrow}$	$\langle *, * \rangle$			
$\lambda 2$	$\langle *, * \rangle$	$\langle \Box, * \rangle$		
λP	$\langle *, * \rangle$		$\langle *, \Box \rangle$	
$\lambda P2$	$\langle *, * \rangle$	$\langle \Box, * \rangle$	$\langle *, \Box \rangle$	
$\lambda \underline{\omega}$	$\langle *, * \rangle$			$\langle \Box, \Box \rangle$
$\lambda \omega$	$\langle *, * \rangle$	$\langle \Box, * \rangle$		$\langle \Box, \Box \rangle$
$\lambda P \underline{\omega}$	$\langle *, * \rangle$		$\langle *, \Box \rangle$	$\langle \Box, \Box \rangle$
λC	$\langle *, * \rangle$	$\langle \Box, * \rangle$	$\langle *, \Box \rangle$	$\langle \Box, \Box \rangle$

Cada uma das "dependências extra" determina sistemas diferentes designados por: $\lambda 2$ (*termos dependentes de tipos*), λP (*tipos dependentes de termos*) e $\lambda \underline{\omega}$ (*tipos dependentes de tipos*). Estes três sistemas podem considerar-se como os três eixos de um sistema tridimensional, o que permite visualizar os oito sistemas como os vértices de um cubo – o λ-*Cubo*.

O sistema de tipos mais abrangente, onde todas as dependencias são válidas é o sistema λC chamado *Cálculo das Construções*. É este o sistema que o Coq implementa.

2.1 Termos Generalizados

Vamos começar pela seguinte definição

```
Coq < Definition id := [A:Set][x:A] x.
```

Acabamos de definir o g-termo: $id \doteq (\lambda A\colon *. \lambda x\colon A.x)$. Qual é o tipo de id ?

```
Coq < Check id.
```

Como o próprio Coq deve confirmar, id tem tipo $(\Pi A\colon *. A \rightarrow A)$. Podemos descrever o raciocínio que nos permite chegar a esta conclusão por

$$\begin{array}{rcl} x & : & A \\ (\lambda x\colon A.x) & : & A \rightarrow A \\ (\lambda A\colon *. \lambda x\colon A.x) & : & (\Pi A\colon *. A \rightarrow A) \end{array}$$

Vejamos como poderia ser feita a prova formal do juizo $\vdash (\lambda A\colon *. \lambda x\colon A.x) : \Pi A\colon *. A \rightarrow A$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash * : \Box}{A : * \vdash A : *}}{A : *, x : A \vdash x : A} \qquad \dfrac{\dfrac{\vdash * : \Box}{A : * \vdash A : *} \quad \dfrac{\vdash * : \Box}{A : * \vdash A : *}}{A : * \vdash A \rightarrow A : *}}{A : * \vdash (\lambda x\colon A.x) : A \rightarrow A} \qquad \dfrac{\vdash * : \Box \qquad \dfrac{\dfrac{\vdash * : \Box}{A : * \vdash A : *} \quad \dfrac{\vdash * : \Box}{A : * \vdash A : *}}{A : * \vdash A \rightarrow A : *}}{\vdash (\Pi\, A{:}*.\, A \rightarrow A) : *}}{\vdash (\lambda A\colon *. \lambda x\colon A.x) : (\Pi\, A{:}*.\, A \rightarrow A)}$$

Repare que para fazermos tal prova, o sistema deve dispôr das dependências $\langle *, * \rangle$ e $\langle *, * \rangle$. As provas formais da validade dos juizos são normalmente muito extensas.

Podemos agora fazer

```
Coq < Variable T : Set.
Coq < Variable t : T.
Coq < Compute (id T t).
```

Repare na cadeia de dedução que justifica este resultado:

$$(id\, T\, t) \equiv (\lambda A{:}\, *.\lambda x{:}\, A.x)\, T\, t \rightarrow (\lambda x{:}\, T.x)\, t \rightarrow t$$

Considere agora a seguinte definição

```
Coq < Definition f := [C:Set][D:Set][x:C][y:C->D](y x).
```

O g-termo acabado de definir é $(\lambda C{:}\, *.\lambda D{:}\, *.\lambda x{:}\, C.\lambda y{:}\, C \rightarrow D.yx)$. Qual o seu tipo ?

$$\begin{array}{rcl}
y & : & C \rightarrow D \\
x & : & C \\
yx & : & D \\
(\lambda y{:}\, C \rightarrow D.yx) & : & (C \rightarrow D) \rightarrow D \\
(\lambda x{:}\, C.\lambda y{:}\, C \rightarrow D.yx) & : & C \rightarrow (C \rightarrow D) \rightarrow D \\
(\lambda D{:}\, *.\lambda x{:}\, C.\lambda y{:}\, C \rightarrow D.yx) & : & \Pi D{:}\, *.C \rightarrow (C \rightarrow D) \rightarrow D \\
(\lambda C{:}\, *.\lambda D{:}\, *.\lambda x{:}\, C.\lambda y{:}\, C \rightarrow D.yx) & : & \Pi C{:}\, *.\Pi D{:}\, *.C \rightarrow (C \rightarrow D) \rightarrow D
\end{array}$$

```
Coq < Check f.
```

Faça agora

```
Coq < Variable P : Set.
Coq < Check P->P.
Coq < Compute (f T (P->P) t).
```

Vamos indicar uma cadeia de redução maximal para $(f\, T\, (P \rightarrow P)\, t)$

$$(f\, T\, (P \rightarrow P)\, t) \equiv (\lambda C{:}\, *.\lambda D{:}\, *.\lambda x{:}\, C.\lambda y{:}\, C \rightarrow D.yx)\, T\, (P \rightarrow P)\, t \rightarrow$$

$$\rightarrow (\lambda D{:}\, *.\lambda x{:}\, T.\lambda y{:}\, T \rightarrow D.yx)\, (P \rightarrow P)\, t \rightarrow (\lambda x{:}\, T.\lambda y{:}\, T \rightarrow P \rightarrow P.yx)\, t \rightarrow (\lambda y{:}\, T \rightarrow P \rightarrow P.yt)$$

Exercício: Avalie a seguinte sequência de comandos:

```
Coq < Section aula.
Coq < Variables A, B, C : Set.
Coq < Definition S := [x:A->B->C][y:A->B][a:A](x a (y a)).
Coq < Print S.
Coq < End aula.
Coq < Print S.
```

Comente o efeito e a resposta do sistema Coq à sequência de comandos anterior.

Exercício: Considere as seguintes definições Coq:

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

Declare agora

```
Coq < Variable a :A.
Coq < Variable b :B.
Coq < Variable T : Set.
Coq < Variable t : T.
```

e analise o resultado dos comandos que se seguem fazendo, no papel, as respectivas cadeias de redução maximais.

```
Coq < Compute (exp1 a b A ([x:A][y:B] x) ).
Coq < Compute (exp2 T t).
```

Qual é tipos das expressões acima descritas ?

Exercício: 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) $soma\,(SZ)(S(SZ))$

Questão 4A. Abra uma secção e declare / defina em Coq

$$\begin{array}{l} A\colon * \\ B\colon * \\ exp \doteq \lambda p\colon (\Pi C\colon *.(A \to B \to C) \to C).p\,A\,(\lambda x\colon A.\lambda y\colon B.x) \end{array}$$

i) Quais as variáveis e ligadas de `exp` ?

ii) Deduza p tipo de `exp`. Use o Coq para verificar a sua resposta.

iii) Feche agora a secção que abriu. Qual o efeito do fecho desta secção.

iv) Qual o λ-termo definido agora em `exp` ?

v) Declare agora

$$\begin{array}{l} T\colon * \\ t\colon T \end{array}$$

Construa uma cadeia de redução maximal de raiz $(exp\;T\;T\;(\lambda C\colon *.\lambda x\colon T \to T \to C.x\,t\,t))$.

vi) Qual o tipo de $(exp\;T\;T\;(\lambda C\colon *.\lambda x\colon T \to T \to C.x\,t\,t))$?
Use o Coq para verificar a sua resposta.

Questão 4B. Abra uma secção e declare / defina em Coq

$$\begin{array}{l} A\colon * \\ B\colon * \\ ex \doteq \lambda p\colon B \to A.\lambda q(\Pi C\colon *.(A \to C).\lambda b\colon B.q\,A\,(p\,b) \end{array}$$

i) Quais as variáveis e ligadas de `exp` ?

ii) Deduza p tipo de `exp`. Use o Coq para verificar a sua resposta.

iii) Feche agora a secção que abriu. Qual o efeito do fecho desta secção.

iv) Qual o λ-termo definido agora em `exp` ?

v) Declare agora

$$\begin{array}{l} U\colon * \\ V\colon * \\ x\colon U \\ y\colon V \end{array}$$

Construa uma cadeia de redução maximal de raiz $(exp\;U\;V\;(\lambda y\colon V.x))$.

vi) Qual o tipo de $(exp\;U\;V\;(\lambda y\colon V.x))$?
Use o Coq para verificar a sua resposta.

2.2 Alguns Vértices do λ-Cubo

Utilizaremos agora o sistema Coq para efectuar a manipulação de termos do λ-calculus com tipos, com a presença das várias dependências possíveis: termos dependentes de tipos (introduzidos em $\lambda 2$), tipos dependentes de tipos (introduzidos em $\lambda\underline{\omega}$), e tipos dependentes de termos (introduzidos em λP). O sistema onde todas estas dependências são possíveis tem o nome de **Cálculo das Construções**, ou λC.

2.2.1 Quantificação impredicativa em $\boldsymbol{\lambda 2/F}$ – termos polimórficos (dependentes de tipos)

A dependência $\langle\Box, *\rangle$ num Sistema Abstracto de Tipos com a hierarquia linear de universos $* \in \Box$ permite a presença de abstracções que operam uniformemente sobre termos de todos os tipos. Comecemos por definir a função identidade que recebe um tipo (proposicional) seguido de um elemento desse tipo, que depois devolve:

```
Coq < Section lambda2.
Coq < Definition id := [alpha:Set][x:alpha] x.
Coq < Check id.
```

Note-se que:

(i) `id` tem tipo proposicional.

(ii) A aplicação de `id` a um tipo arbitrário resulta na função-identidade desse tipo.

De facto, a execução dos seguintes comandos permite confirmar estas observações:

```
Coq < Check (alpha:Set)alpha->alpha.
Coq < Variable D:Set.
Coq < Compute (id D).
Coq < Variable d:D.
Coq < Compute (id D d).
```

Exercício: Interprete a seguinte sequência de comandos. Diga se o termo `strange` é um termo de $\lambda 2$ antes e depois de ser fechado, e porquê.

```
Coq < Section l21.
Coq < Variable beta:Set.
Coq < Definition strange := [a: (alpha:Set)alpha] (a beta).
Coq < Check strange.
Coq < End l21.
Coq < Check strange.
Coq < Print strange.
```

2.2.2 Tipos dependentes de tipos em $\lambda\underline{\omega}$

A dependência $\langle\Box,\Box\rangle$ permite extender o conjunto de *géneros* dos SATs (gerados pela hierarquia $* \in \Box$) pela seguinte instância da regra do produto:

$$\frac{\Gamma \vdash A : \Box \quad \Gamma \vdash B : \Box}{\Gamma \vdash A \to B : \Box}$$

Juntamente com o axioma $\Gamma \vdash * : \Box$, esta regra permite a existência de géneros como $* \to *$ ou $(* \to *) \to (* \to * \to *)$. comecemos por ver como por exemplo o combinador **K** pode ser escrito como operando sobre tipos proposicionais:

```
Coq < End lambda2.
Coq < Section lambdaWweak.
Coq < Check [A:Set][B:Set]A.
Coq < Check Set->Set->Set.
```

(Note-se que `Type` corresponde, em **Coq**, ao universo $\Box$.)

Estudemos em seguida um pequeno exemplo: consideremos o construtor de tipos `Par`, que a partir de dois tipos A e B constroi o tipo dos pares ordenados com a primeira componente em A e a segunda em B:

```
Coq < Variable Par : Set -> Set -> Set.
Coq < Variable A, B : Set.
```

`Par` é um g-termo classificado pelo género $* \to * \to *$, pelo que não é um tipo proposicional, nem em geral um tipo concreto.

Declaremos agora termos representando o constructor essencial dos pares (que agrega dois elementos num par), bem como os seus dois destructores (que desagregam um par nas suas duas componentes):

```
Coq < Parameter mkPair : A->B->(Par A B).
Coq < Parameter proj1 : (Par A B) -> A.
Coq < Parameter proj2 : (Par A B) -> B.
Coq < Check ( [x:(Par A B)] [y:B] (mkPair (proj1 x) y) ).
```

2.2.3 O sistema $\lambda\omega$

Se pretendermos agora declarar termos equivalentes aos constructores e destructores acima declarados, mas que sejam polimórficos, ou seja, capazes de lidar com pares de elementos de quaisquer tipos, poderemos fazer algo como:

```
Coq < Parameter MkPair : (A,B:Set) A->B->(Par A B).
Coq < Check (A,B:Set) A->B->(Par A B).
Coq < Parameter Proj1 : (A,B:Set) (Par A B) -> A.
Coq < Parameter Proj2 : (A,B:Set) (Par A B) -> B.
```

Note-se que os tipos de tais termos são ainda proposicionais (ou seja de género $*$), mas só têm existência na presença das duas dependências $\langle\Box, *\rangle$ e $\langle\Box, \Box\rangle$. O sistema onde ambas estão presentes toma o nome de $\lambda\omega$.

Exercício: Como definiria, recorrendo aos termos de que dispõe neste momento, a função polimórfica que recebe um par ordenado com ambas as componentes do mesmo tipo, e devolve esse par com as componentes invertidas? Qual o tipo de tal função?

(ex: $\langle 2, 4\rangle \mapsto \langle 4, 2\rangle$)

A coexistência das duas dependências referidas tem efeitos mais complexos do que os que se observam nos termos acima referidos. De facto, veja-se qual o efeito de se fechar a secção actual:

```
Coq < End lambdaWweak.
Coq < Check mkPair.
```

O tipo do termo `mkPair` foi alterado para reflectir o fecho desse termo em relação a identificadores declarados localmente na secção em que trabalhávamos. Como efeito desse fecho, o termo foi tornado polimórfico no sentido sugerido em 2.2.3 (efeito da abstracção nos tipos `A` e `B`), mas foi feita também abstracção no próprio constructor de tipos `Par`.

Note-se que o tipo deste termo `mkPair` é construído pela regra do produto com a dependência $\langle\Box, *\rangle$, mas tendo agora em conta a nova forma possível para os géneros (g-termos classificados por $\Box$), introduzida pela dependência $\langle\Box, \Box\rangle$. O efeito é o de ser agora possível o polimorfismo não só nos tipos mas também nos seus constructores.

Resta fazer o seguinte comentário: o exemplo que utilizámos, do tipo estruturado polimórfico "Par Ordenado", tem interesse reduzido, dada a ausência de um mecanismo pelo qual asseguremos as equivalências semânticas próprias desse tipo, nomeadamente,

$$\begin{array}{rcl} mkPair\ (proj1\ p)\ (proj2\ p) & = & p, \\ proj1\ (mkPair\ x\ y) & = & x, \\ proj2\ (mkPair\ x\ y) & = & y. \end{array}$$

Veremos, quando estudarmos os tipos indutivos, como tal deficiência pode ser facilmente ultrapassada.

2.2.4 Quantificação Predicativa em λP – Tipos Dependentes de Termos

A última dependência que é possível introduzir na família de SATs que temos vindo a considerar é $\langle *, \Box\rangle$, responsável, quando acrescentada a $\lambda\rightarrow$, pelo aparecimento de géneros da forma $A_1 \rightarrow$

$A_2 \ldots \to A_i \ldots \to *$, com A_i tipos proposicionais. A seguinte instância da regra do produto, juntamente com o axioma $\Gamma \vdash * : \Box$, gera todos estes géneros:

$$\frac{\Gamma \vdash A : * \quad \Gamma \vdash B : \Box}{\Gamma \vdash A \to B : \Box}$$

O sistema resultante toma o nome de λP.

```
Coq < Restore State Initial.
Coq < Section lP.
Coq < Variable A : Set.
Coq < Check A->Set.
Coq < Variable P : A->Set.
Coq < Variable a:A.
Coq < Check (P a).
Coq < Check ((P a) -> Set).
Coq < Check ((P a) -> A -> Set).
```

A um termo como P acima declarado chama-se normalmente, em Teoria de Tipos, um predicado sobre o tipo A.[4]

A regra de formação de tipos-produto, que na sua instância de $\lambda_{\to}$ dava origem à formação dos tipos funcionais da forma $A \to B$, deve em λP ser devidamente reinterpretada. De facto, a razão pela qual em $\lambda_{\to}$ se podia escrever o tipo $\Pi x\colon A.B$ simplesmente como $A \to B$, prendia-se com a impossibilidade de o tipo B depender de alguma forma da variável x de tipo A. Ora, essa dependência torna-se possível em λP: basta observar, com $P : A \to *$, o tipo-produto $\Pi x\colon A.(Px)$.

Estes tipos, cuja existência só em λP se torna possível, dizem-se formados por quantificação predicativa, já que se quantifica em variáveis que percorrem um tipo (A) que é classificado da mesma forma que o tipo quantificado (pelo universo $*$, neste caso).

```
Coq < Check ((a:A)(P a)).
```

Exercício: Será o g-termo $\Pi a\colon A.(Pa) \to *$ bem formado em λP (na secção corrente)? Utilize o **Coq** para confirmar a sua resposta.

Exercício: Escreva um termo cujo tipo seja $\Pi a\colon A.(Pa \to Pa)$, na mesma secção. Utilize o **Coq** para provar que o termo que apresentou habita realmente este tipo.

2.2.5 Cálculo de Construções (λC)

Acrescentando-se a dependência $\langle *, \Box \rangle$ ao sistema $\lambda\omega$ obtém-se o sistema λC.[5]

A coexistência das várias instâncias da regra do produto produz diversos efeitos. Antes de mais, a interacção entre as dependências $\langle *, \Box \rangle$ e $\langle \Box, \Box \rangle$ produz novas formas possíveis para os géneros:

[4] Em geral, um predicado será qualquer termo de tipo $A \to U$, com U um universo e $A : U$.

[5] Os outros sistemas do λ-cubo são os que resultam de se acrescentar esta dependência a $\lambda\underline{\omega}$, resultando em $\lambda P\underline{\omega}$, e a $\lambda 2$, resultando em $\lambda P2$. O sistema λC contém estes e todos os outros sistemas que temos vindo a discutir.

```
Coq < Check ((A -> A -> Set) -> (A -> Set)).
Coq < Check ((P:A->Set)(a:A) (P a) -> Set).
```

Exercício: Apresente g-termos habitantes dos géneros anteriores e verifique os seus tipos em Coq.

Por outro lado, a instância da mesma regra gerada pela dependência $\langle \Box, * \rangle$ vai permitir agora a quantificação impredicativa em variáveis de todos os novos géneros. Observe-se os seguintes tipos proposicionais:

```
Coq < Print A.
Coq < Print P.
Coq < Check ((f:(A -> Set) -> (A -> Set)) (f P a)).
Coq < Check ((P:A->Set)(a:A) (P a) -> (P a)).
```