2 O Cálculo das Construções

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.

2.1 Quantificação impredicativa em $\lambda 2/F$ – 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 Tipos dependentes de tipos em $\boldsymbol{\lambda\underline{\omega}}$

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) ).
```

2.3 O sistema $\boldsymbol{\lambda\omega}$

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 $\lambda\omega$.

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 2.3 (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.

2.4 Quantificação Predicativa em $\boldsymbol{\lambda P}$ – 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 $A_1 \rightarrow A_2 \ldots \rightarrow A_i \ldots \rightarrow *$, com A_i tipos proposicionais. A seguinte instância da regra do produto, juntamente com o axioma $\Gamma \vdash * : \Box$, gera todos estes géneros:

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

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.[3]

A regra de formação de tipos-produto, que na sua instância de $\lambda \rightarrow$ dava origem à formação dos tipos funcionais da forma $A \rightarrow B$, deve em λP ser devidamente reinterpretada. De facto, a razão pela qual em $\lambda \rightarrow$ se podia escrever o tipo $\Pi x\colon A.B$ simplesmente como $A \rightarrow B$, 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 $P : A \rightarrow *$, 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: Escreva um termo cujo tipo seja $\Pi a\colon A.(Pa \rightarrow Pa)$, na mesma secção. Utilize o **Coq** para provar que o termo que apresentou habita realmente este tipo.

2.5 Cálculo de Construções (λC)

Acrescentando-se a dependência $\langle *, \Box \rangle$ ao sistema $\lambda\underline{\omega}$ obtém-se o sistema λC.[4]

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:

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

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

[4]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.

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)).
```