Linguagens e Ambientes de Programação (2018/2019)

Teórica 10 (08/abr/2019)

Ligações, ambientes, âmbitos. Regras de escopo.

Ligações (bindings)

De forma geral, uma ligação consiste numa associação entre duas entidades. Essas **entidades** podem ser nomes, localizações de memória, tipos, objetos, etc. Por exemplo, uma variável imperativa do C pode ser formalizada como uma ligação entre um nome e uma localização de memória.

Uma ligação é sempre unidirecional e geralmente associa uma entidade mais simples a outra entidade mais complexa. O caso das duas entidades terem complexidade semelhante também ocorre por vezes. As ligações facilitam o trabalho do programador pois ele pode usar a entidade mais simples para representar a entidade mais complexa.

Por exemplo:

- Ligação dum nome a um valor fixo, que pode ver um inteiro, uma função, etc. (constante).
- Ligação dum nome a uma localização de memória (variável mutável do C).
- Ligação duma localização de memória a outra localização de memória (apontador).
- Ligação dum nome a um valor, onde o valor pode mudar dinamicamente (variável mutável do JavaScript).

Diferentes ligações duma entidade

Para uma mesma entidade, digamos um nome, podem estar definidas multiplas ligações. Por exemplo, em C o nome duma variável global tem as seguintes ligações:

- Ligação a um tipo fixo (estabelecida em tempo de compilação).
- Ligação a uma localização de memória fixa (estabelecida em tempo de carregamento).
- Ligação a um valor (estabelecida em tempo de execução).

Semântica e ligações

A semântica de qualquer linguagem de programação é determinada de forma essencial a partir:

- O conjunto de formas de ligação que se podem estabelecer;
- O momento em que essas ligações se estabelecem.

Momento da ligação

Eis diversos momentos em que uma ligação pode ser estabelecida:

- Tempo de conceção da linguagem. Exemplo em OCaml: operador "+".
- Tempo de implementação da linguagem Exemplo em OCaml: constante "max_int".
- Tempo de compilação. Exemplo em C: constante "#define A 2"
- Tempo de ligação. Exemplo em C: função externa "extern int f(void);".
- Tempo de carregamento. Exemplo em C: variável global "int x;".
- Tempo de execução. Exemplo em OCaml: constante "let a = 2 + x in ...".

Em geral, quanto mais tarde se estabelecem as ligações mais flexível é a linguagem.

Em geral, quanto mais cedo se estabelecem as ligações, mais rápida é a linguagem.

Por exemplo, é mais eficiente invocar um procedimento em C (ligação em tempo de ligação) do que enviar uma mensagem para um objeto em C++ (ligação em tempo de execução). Em todo o caso, justifica-se o preço a pagar em C++: é o facto da ligação entre a mensagem e o método ser estabelecida muito tarde (late binding) que permite fazer funcionar o paradigma orientado pelos objetos.

Classificação das ligações

Classificação das ligações em função do momento de ligação:

- **Estáticas**: efetuadas antes da execução do programa (portanto até ao momento do carregamento, inclusive).
- **Semidinâmicas**: efetuadas em tempo de execução mas determinadas em grande parte antes de o programa começar a correr.
- **Dinâmicas**: efetuadas completamente em tempo de execução.

Exemplos de ligações estáticas:

- nome->valor: constantes "#define" em C;
- nome->tipo: tipos, variáveis e constantes em ML e C; (ML->inferência de tipos)
- nome->localização: variáveis globais em C.

Exemplos de ligações semidinâmicas:

- nome->localização: variáveis locais em C e constantes locais em ML;
- nome->método: mensagens em C++ e Java.

Discussão do caso das variáveis locais em C, um caso de ligação semidinâmica: Essas variáveis residem na chamada *pilha de execução* do C, e as respetivas localizações de memória são determinadas em tempo de execução. No entanto, é em tempo de compilação que se determina o *offset* dentro do registo de cativação da função onde a variável reside. As constantes locais do ML "let a = 2 + x in ..." também são implementadas da mesma forma.

Exemplos de ligações dinâmicas:

- localização->valor: variáveis mutáveis em C;
- localização->localização: apontadores em C;
- nome->valor: variáveis mutáveis em qualquer linguagem;
- nome->tipo: variáveis em Smalltalk e Ruby.

Discussão do caso das variáveis em Smalltalk e Ruby: Estas variáveis não têm tipos estáticos associados. Aceitam valores de qualquer tipo e portanto, sempre que há uma atribuição muda o valor e o tipo dessas variáveis.

Tempo de vida duma ligação

O tempo de vida duma ligação é o período de tempo da execução dum programa durante a qual essa ligação persiste. As ligações estáticas persistem durante a execução de todo o programa. As ligações semidinâmicas

e dinâmicas persistem geralmente apenas durante parte da execução do programa.

Ambiente (conjunto de ligações para nomes)

O conceito de ambiente aplica-se apenas a um tipo particular de ligações: as ligações de nomes.

Chama-se **ambiente** a um conjunto de ligações que associam nomes (identificadores) a entidades. Matematicamente um ambiente é uma função de nomes para entidades, ou seja uma função com o seguinte tipo:

```
Nomes -> Entidades
```

O seguinte pequeno programa em ML define quatro ambientes diferentes, consoante o ponto do programa que for considerado:

```
let f x = x + 1;;
let rec g x = f x + 1;;
```

1. O ambiente antes da função f inclui apenas as ligações dos nomes predefinidos na linguagem ML. São exemplos desses nomes: max, "^", "+", int, float.

2. O ambiente no interior da função f inclui as ligações dos nomes predefinidos, mais a a ligação do nome "x" que representa o argumento de f. Repare que o nome "f" não tem ligação dentro da função f, porque esta função não é recursiva.

3. O ambiente no interior da função g inclui as ligações dos nomes predefinidos, mais a ligação do nome "x" que representa o argumento de g, mais os nomes "f" e "g" que representam funções.

4. O ambiente após a função g inclui as ligações dos nomes predefinidos, mais os nomes "f" e "g" que representam funções.

Âmbito (escopo) duma ligação

Âmbito (escopo) duma ligação é a região do programa na qual esse nome tem os atributos estabelecidos pela declaração que introduz a ligação.

Na maior parte das linguagens de programação, o âmbito duma ligação é determinado pela estrutura sintática do programa (ver "Escopo estático", mais abaixo).

Há exemplos de âmbitos na secção "Blocos", um pouco mais abaixo.

Construções definidoras de âmbitos

A generalidade das linguagens de programação possuem construções que têm implicações nos âmbitos das ligações que se estabelecem nos programas. Eis alguns exemplos dessas construções:

- Blocos (C, C++, Java, Pascal, Ada)
- Let-in (OCaml)
- Módulos (Módula-2, OCaml)
- Classes (C++, Java, C#)
- Packages (Java)
- Namespaces (C++, C#)
- Espaço global (duma forma ou de outra, todas as linguagens dispõem dum espaço global de nomes)

Blocos

As linguagens que descendem do antigo Algol-60 possuem uma construção sintática chamada **bloco**. O Java, C e muitas outras linguagens suportam blocos. Um bloco tem duas utilidades:

- Serve para introduzir um novo ambiente no qual todas as novas ligações têm aproximadamente o mesmo âmbito. (Esta é a parte que nos interessa aqui.)
- Serve para agregar uma sequência de comandos num único comando, que se diz "composto".

O seguinte bloco, em C, determina ligações para os nomes `i`, `j` e `k`. Todas essas ligações têm como âmbito aproximadamente todo o interior do bloco. "Aproximadamente", porque realmente o âmbito de `j` é ligeiramente mais pequeno do que o âmbito de `i`, e o âmbito de `k` é ligeiramente mais pequeno do que o âmbito de `j`. Onde é que começa exatamente o âmbito de cada uma das três ligações?

```
{
    int i = 0
    int j = i + 2
    int k = i + j;
    printf("%d %d %d\n", i, j, k);
}
```

O seguinte exemplo, também em C, é mais interessante e ilustra um **bloco aninhado** dentro de outro bloco.

```
{
    int i;
    int b = 5;
    i = a + b;
    {
        int i = 0;
        int j = i + 2;
        int k = i + j;
        printf("%d %d %d\n", i, j, k);
    }
    printf("%d %d\n", i, b);
}
```

Repare que o âmbito da variável `b`, introduzida no bloco exterior, abrange aproximadamente todo o bloco externo, o que inclui o bloco interno. No entanto o âmbito da variável `i` introduzida no bloco exterior abrange o bloco exterior *menos* o bloco interior, porque a variável `i` é redefinida no bloco interior.

Este exemplo mostra que numa linguagem onde as construções definidores de âmbitos podem ser aninhadas, o âmbito duma ligação pode não corresponder a uma zona contígua de programa. Por outras palavras, **pode haver "buracos" âmbito duma ligação**!

Resolução de nomes

Chama-se **resolução de nomes** ao processo de descoberta do significado (ou seja, da ligação) de alguns nomes num ponto do programa onde esses nomes são usados.

Escopo estático

Escopo estático é o nome da regra de resolução de nomes usada na maioria das linguagem modernas, incluindo o OCaml, C, C++ e Java.

A regra é muito simples e diz apenas o seguinte:

- Um uso dum nome refere-se sempre à ligação sintaticamente envolvente, mais próxima.

Portanto, para saber o que o significado dum nome num dado ponto do programa, basta olhar para o código "à volta" (de acordo com as regras da sintaxe da linguagem), e procurar aí a declaração mais próxima desse nome. Um caso particular: se o nome estiver declarado localmente, então é essa declaração local que vale para o nome em causa.

Quem aprendeu a programar numa linguagem moderna, está tão habituado a esta regra que geralmente nem se apercebe dele. Bem, nos exemplos da secção "Blocos", atrás, nós já usámos esta regra "sem dar por isso"...

Mais um exemplo, agora em ML:

```
let z = 5 in
    let f x = x + z in
        let z = 6 in
            f 0
```

Neste exemplo aparece uma utilização duma variável `z`, dentro da função `f`, que pode criar dúvidas. Será que uso do nome `z` se refere à declaração de `z` exterior, ou à declaração de `z` interior?

Como a linguagens ML usa a regra de escopo estático, a resposta correta é: o uso do nome `z` refere-se à declaração de `z` exterior (ou seja, envolvente).

Pergunta: Quando o valor da expressão do exemplo, 5 ou 6?
Resposta: 5.

Escopo dinâmico

Escopo dinâmico é o nome da regra de resolução de nomes atualmente em desuso, mas que importa conhecer para se estabelecerem contrastes com a regra de escopo estático. É usado em algumas versões da linguagem Lisp e também na linguagem APL, por exemplo.

A regra também é muito simples e diz apenas o seguinte:

- Um uso dum nome refere-se sempre à ligação mais recentemente estabelecida para esse nome, temporalmente, durante a execução do programa.

Regressemos ao exemplo:

```
let z = 5 in
    let f x = x + z in
        let z = 6 in
            f 0
```

Pergunta: Usando a regra de escopo dinâmico, qual o valor da expressão do exemplo, 5 ou 6?
Resposta: 6.

A resposta é 6 porque, repare, quando a função `f` é chamada, a ligação mais recente para `z` é que foi estabelecida na declaração de `z` interior.

Efeitos das regras

Em muitas situações, como por exemplo quando estão em causa nomes declarados localmente, as duas regras de escopo acabam por dar os mesmos resultados, ou seja, resolvem os nomes da mesma forma. Só perante situações semelhantes à do exemplo anterior é que os efeitos são diferentes.

Em rigor, os efeitos das regras só diferem quando estão em causa acessos a nomes não-locais a partir do interior de funções. Por isso as duas regras de escopo podem ser apresentadas da seguinte forma alternativa:

- **Escopo estático** - As funções são chamadas no ambiente da sua definição.
- **Escopo dinâmico** - As funções são chamadas no ambiente de quem os invoca.

Comparação

O escopo estático é usado em praticamente em todas as linguagens modernas pois faz com que a estrutura estática de um programa se aproxime do seu comportamento dinâmico. Isto simplifica imenso a compreensão dos programas.

A regra de escopo dinâmico tem ainda mais estas desvantagens:

- Não permite fazer verificação de tipos estática;
- O comportamento dum programa fica sensível aos nomes escolhidos para as suas variáveis e procedimentos.

Estado (conjunto de ligações para localizações)

O conceito de estado tem a ver com um tipo particular de ligações: as ligações de localizações a outras entidades.

Chama-se **estado** a um conjunto de ligações que associam localizações de memória a entidades. Matematicamente um estado é uma função de localizações para entidades, ou seja uma função com o seguinte tipo:

```
Localizações -> Entidades
```

As linguagens funcionais puras não possuem estado. Este facto tem a desvantagem de reduzir a variedade de ligações que se podem estabelecer. Mas tem a vantagem de simplificar a linguagem; outra vantagem é o facto da linguagem ficar mais segura pois sabe-se que a maioria dos dos bugs dos programas estão relacionados com variáveis mutáveis ou com apontadores.

#100