Compiladores – Ano lectivo 2009/10

Acerca da representação para a APT na Linguagem VSPL

Informação de versão: `Id: apt-vspl.tex,v 1.2 2010/03/26 07:35:57 spa Exp spa`

Especificação

Na construção da árvore abstracta para um programa em VSPL, é necessário ter em conta a estrutura concreta da gramática utilizada para construir o parser. No entanto, e como um dos objectivos da noção de sintaxe *abstracta* é o de "suavizar" a sintaxe concreta, é possível definir alguns critérios gerais para elaboração da sintaxe abstracta. Estes irão incorporar alguns aspectos da semântica pretendida para a linguagem.

Este documento pretende apresentar critérios para compôr a sintaxe abstracta para uma linguagem de programação. É aplicada ao VSPL mas pode ser adaptada a outras linguagens duma forma simples.

Na secção 1 são apresentados princípios orientadores para a resolução deste tipo de problema. Na secção 3 descreve-se abordagens à representação de literais. Na secção 4 as expressões (aritméticas, nomeadamente) são descritas e na secção 5 é abordada a problemática da representação das instruções ("statements").

1 Princípios Gerais

Embora se trate de especificar uma estrutura capaz de exprimir um programa, com uma sintaxe simplificada, muitos aspectos importantes para fases posteriores do processo de compilação serão omitidos ao nível da APT, são estes nomeadamente:

- A informação de tipo associada a qualquer expressão ou nome.
- A informação de âmbito ("scope") associada aos nomes.

Estes aspectos só deverão ser contemplados posteriormente, na fase de análise semântica (análise de nomes e de tipos).

Por ter sido estipulado que as fases de análise semântica e posteriores seriam implementadas numa linguagem de Programação em Lógica com Restrições (*Constraints*), torna-se importante adequar as características da APT ao modelo computacional subjacente. Assim, para beneficiar plenamente das capacidades dos mecanismos de Unificação e Constraints, importa especificar cuidadosamente a estrutura da APT por forma a criar oportunidades de factorizar a representação sempre que isto se mostre benéfico.

2 Tipos

Quando for preciso representar um tipo, utiliza-se um termo. Os termos associados aos tipos do VSPL restrito são:

`int` Um inteiro.

`bool` Um booleano.

`array(TYPE, DIM)` Um array (vector) do tipo `TYPE` com `DIM` (uma expressão inteira) elementos.

`pair(TYPE1, TYPE2)` Um par de tipos.

`tuple([T1,T2,..Ts])` Uma lista de tipos, representando um tuplo.

3 Literais

Os literais que ocorrem na linguagem VSPL restrita são:

- As constantes inteiras,
- As constantes booleanas,
- Os literais de tipos compostos que incluem os tuplos, os *arrays*,
- Os literais funcionais.

Para agrupar estas formas de literal, convém definir uma estrutura comum capaz de as representar todas. Uma possível representação seria:

```
lit(TYPE, VALUE)
```

Em que `TYPE` é um átomo com a representação do tipo de literal que está a ser representado e `VALUE` uma sua representação, com uma estrutura adequada ao tipo. Por exemplo, poderiamos ter:

`lit(int, 3)`
: Para a constante inteira 3.

`lit(pair(int, pair(bool, int)), V)`
: Em que **`V = pair(lit(int, 2), pair(lit(bool, true), lit(int, 4)))`**, para o literal de tuplo **`(2, true, 4)`**, que é entendido como **`(2, (true, 4))`**.

`lit(tuple([int, bool, int]), [lit(int, 2), lit(bool, true), lit(int, 4)])`
: Para o mesmo literal, mas neste caso a estrutura foi "linearizada" pois ficámos com um triplo expresso como uma lista Prolog e não uma árvore. Esta representação é alternativa à anterior e, embora possa ser mais trabalhosa de obter, é conveniente por ser pouco profunda.

`lit(map(ARGT, VALT), VALOR)`
: Para um literal funcional. Note-se que, neste caso, o tipo de literal é muito simplificado pois ainda não inclui informação explícita sobre o tipo concreto do literal: este está por agora no valor associado e possívelmente só poderá ser determinado completamente após a análise de nomes e de tipos. Só quando isto tiver sido feito é que `ARGT` e `VALT` poderão tomar valores.

  Também acerca deste exemplo concreto, não é dado mais detalhe sobre o valor associado pois este requer que tenham sido definidas as representações para expressões (secção 4) e para as instruções ou "statements" (secção 4).

`lit(array(T,NUM), VALOR)`
: Para um literal de array. Tal como no caso anterior, muita da informação sobre o literal (nomeadamente a conducente à determinação do tipo do literal) está contida no argumento `VALOR` e só depois de alguma análise (p/ex a travessia de `VALOR`) é que se podem concretizar `T` e `NUM`.

Atenção que o tipo referido no termo `lit` começa por ser uma especificação de tipo *incompleta*, no sentido dos tipos do VSPL. O papel de determinar o tipo exacto do literal (assim como das expressões) pertence a uma fase posterior da compilação, designada por *análise de tipos.*

4 Expressões

No tocante a expressões há muitas formas viáveis de as representar usando termos Prolog. Uma primeira escolha consiste em determinar se se agrupam todas as expressões sob uma representação comum ou se se deixa que cada expressão seja distinta das restantes, imediatamente ao nível superficial. Assim, estariam em confronto as seguintes[1] representações, por exemplo para a expressão `2+a`:

[1] Claro que esta enumeração não é exaustiva, pretende apenas sugerir formas diferentes de representar expressões focando alguns dos aspectos próprios a cada uma.

`expr(add, expr(lit(int, 2)), expr(name(a)))`
: Nesta representação, uma expressão é dada por um termo `expr/3`, em que o primeiro argumento indica a operação e os restantes os seus operandos, sendo estes forçosamente coagidos a serem representados como uma expressão.

`add(lit(int, 2), name(a))`
: Esta abordagem simplifica em larga medida a anterior, não tornando explícito que se trata duma expressão mas transmitindo essa informação implícitamente pelo contexto em que se insere.

`binop(add, lit(int, 2), name(a))`
: Esta representação é bastante próxima da anterior mas foca o facto da expressão em causa ser a aplicação dum operador binário (daí o functor principal ser `binop/3`), deixando para o papel de argumento a especificação da operação concreta a usar.

`op(add, lit(int, 2), name(a))`
: É uma representação que coincide com a anterior mas que se adequa mais à filosofia do Prolog no sentido em que um termo composto é caracterizado pelo par (functor, aridade) e não só pelo functor, o que permite que se distingam fácilmente por exemplo `op/3` para operadores binários de `op/2` para operadores unários.

Qual a melhor? Esta pergunta não tem resposta muito clara, pelo que a decisão ficará à escolha de quem tiver de implementar procedimentos que actuem sobre estas estruturas de dados. Dado que procuramos ter alguma interoperabilidade entre trabalhos de diversos grupos, temos de assentar numa representação: será a última.

Note-se que a distinção entre `primary` e `expr` desaparece nesta APT, pois a função desta distinção (e da introdução do não-terminal `primary`) era exclusivamente de restringir sintacticamente o conjunto de programas aceitáveis.

5 Instruções

Tal como para as expressões, a multiplicidade de abordagens possíveis é vasta. As questões que se colocam são as mesmas, na sua essência, pelo que nos limitamos a apresentar algumas possibilidades sem preferir nenhuma às restantes. Os exemplos são excertos de código VSPL, nos quais se focam os aspectos relativos ao uso de diferentes instruções (“statements”).

5.1 Exemplo

Supondo que estamos a representar a instrução em VSPL dada pelo segmento de programa expresso na figura 1, poderemos recorrer às seguintes representações:

```
[ a := 2;
  * [ a < 10 -> a := a+1 ] ]
```

Figura 1: Exemplo de código para as instruções (“statements”)

Existem várias formas de representar este fragmento de programa, aquela que optamos por usar é esta:

`[assign(name(a), lit(int, 2)), while(CLAUSES)]`
: Para representar a sequência de instruções usou-se uma *lista* nativa do Prolog. Para as restantes instruções usaram-se convenções “minimalistas” semelhantes à usada para os literais, como os que ocorrem neste exemplo.

  Por exemplo, o elemento `CLAUSES` que aparece poderá ser representado por:

`CLAUSES=[clause(op(lt, name(a), lit(int, 10)), STMT)]`
Note-se que `CLAUSES` é uma lista, embora neste caso só com um elemento. Também o termo `clause/2` tem como primeiro argumento uma expressão (a guarda) que deverá ser booleana e como segundo argumento uma instrução (statement):

`STMT=assign(name(a), op(add, name(a), lit(int, 1)))`
Finalmente, o segundo subtermo de `CLAUSES` é descrito: tem a estrutura duma instrução (statement.)

5.2 Enumeração de statements

Concretamente, os tipos de nó de APT para VSPL que iremos utilizar ao longo da disciplina são estes:

`decl(NAME, TYPE, VALUE)`
Em que NAME é da forma `name(NOME)` e NOME é um átomo, `TYPE` é uma expressão de tipo (como na secção 2).

`assign(EXPR, EXPR)`
Uma afectação, em que a primeira expressão deverá ser um *primário* e a segunda não terá restrições. Note-se que neste caso deixa de haver distinção estrutural entre o primeiro e segundo argumentos: sabemos (pela análise sintactica) que o primeiro elemento tem um domínio muito mais restrito (o dos primários), embora isso não se reflicta aqui.

`funcall(EXPR, EXPR)`
Uma chamada de função, como expressão. As mesmas observações que para a afectação são aqui aplicáveis.

`return(EXPR)`
Retorna um valor.

`break`, `skip`, `retry`
Instruções de modulação de fluxo de controle local. Do ponto de vista da APT são semelhantes.

`cond(CLAUSES)`
Grupo de clausulas condicionais (o "if-then-else" da linguagem). `CLAUSES` será posteriormente detalhado.

`while(CLAUSES)`
Ciclo while: estes ciclos são ligeiramente mais poderosos que os tradicionais ciclos while de outras linguagens de programação, pois estes permitem ter múltiplas condições, entendo-se por condição de terminação do ciclo a não-satisfacção de nenhuma dessas condições. A estrutura é a mesma do `cond`.

A lista de clausulas tem a forma duma lista Prolog em que os elementos podem ser:

`clause(EXPR, STAT)` Para uma clausula "normal", com uma guarda (a expressão booleana) e uma instrução (o statement).

`clause(STAT)` Para a clausula "else", em que a instrução é incondicionalmente avaliada.

`[STAT | STATS]`
Lista de statements (ou statement composto.) Neste caso foi simplesmente usada o termo lista do Prolog, por conveniência.

6 Representação noutras linguagens

Caso pretendamos representar a APT numa linguagem que não Prolog, basta criar tipos de dados que contenham a informação descrita anteriormente. Caso estejamos a lidar com uma linguagem orientada a objectos, poderemos procurar modelar a taxonomia das entidades da linguagem como uma hierarquia de classes, com herança.