CSet - Apresentação da linguagem

Daniella Angelos [1]

[1]Departamento de Ciência da Computação - Universidade de Brasília

1. Objetivo

Este trabalho visa apresentar a linguagem formal escolhida como linguagem fonte do compilador que será implementado.

2. Introdução

Atualmente, a ciência da computação é utilizada para auxiliar em pesquisas e resolução de problemas das mais diversas áreas do conhecimento. Neste sentido, linguagens de programação são criadas e adaptadas para atender esta demanda e facilitar a tradução de problemas do mundo real a um idioma que a máquina entenda. Uma das formas mais comuns de classificar estas linguagens de programação é com relação ao propósito para o qual foi projetada.

A principal vantagem de linguagens de programação de propósito geral é a extensa variedade de domínios nos quais podem ser aplicadas. Elas recebem esta definição por não possuírem construções que auxiliam programações voltadas a um domínio singular. No entanto, é possível utilizá-las como base para construir outras linguagens, com estruturas que diminuem a dificuldade de abstração de problemas específicos.

Este trabalho apresenta a linguagem *CSet*, uma modificação de uma versão reduzida da Linguagem C. CSet possui estruturas para lidar com uma abstração comum em problemas de lógica e matemática: conjuntos. Um conjunto é uma coleção de objetos, onde cada objeto é um elemento deste conjunto, e é um dos conceitos mais básicos da matemática.

As próximas seções estão organizadas de forma a apresentar a versão inicial da gramática da linguagem, bem como uma visão geral de sua sintaxe e semântica.

3. Descrição da linguagem

3.1. Gramática

Na literatura é possível encontrar algumas versões reduzidas da linguagem C, principalmente para fins didáticos. A gramática de CSet é baseada em uma dessas versões [1] com modificações que incluem alguns tipos primitivos e operadores não presentes nem na versão reduzida, nem na versão original da linguagem [2].

O principal tipo primitivo introduzido é o `set`, que corresponde à representação de conjuntos da linguagem. Para construção de conjuntos genéricos, a especificação de um subtipo segue a declaração de `set`, que restringe-se aos tipos primitivos da linguagem. Também se introduziu o tipo `pair`, necessário como subtipo dos conjuntos resultados de produtos cartesianos entre dois outros. Por fim, `bool` também foi considerado interessante como tipo primitivo por ser uma abstração relevante em lógica e matemática.

Além de operadores inerentes à linguagem C (+, −, `||` etc), foram adicionados os seguintes:

- `<?>` : pertinência
- `<*>` : produto cartesiano
- `$` : cardinalidade de conjunto

A sintaxe e a semântica desses operadores serão introduzidas nas seções 3.2 e 3.3, respectivamente.

A última modificação relevante diz respeito às inclusões de `{ FactorList }` e `( Pair )` à variável `Term`, tornando possível a inicialização de conjuntos e pares de variáveis. A estrutura sintática será explicada na seção 3.2.

Finalmente, segue a especificação da gramática da linguagem CSet, sabendo que variáveis são iniciadas por letras maiúsculas, terminais são símbolos ou palavras iniciadas por letras minúsculas e `Function` é a variável inicial. A ordem em que as operações aparecem corresponde à precedência de cada operador, sendo a primeira a aparecer a de menor precedência, e a última a de maior.

```
Function        -> Type identifier ( ArgList ) CompoundStmt

ArgList         -> Arg
                   | ArgList , Arg

Arg             -> Type identifier

Declaration     -> Type IdentList
                   | Type Attr

Type            -> int
                   | float
                   | char
                   | bool
                   | set < Type >
                   | pair < Type , Type >

IdentList       -> identifier , IdentList
                   | identifier

Stmt            -> WhileStmt
                   | Expr ;
                   | IfStmt
                   | CompoundStmt
                   | Declaration ;
                   | ReturnStmt ;
                   | ;

WhileStmt       -> while ( Expr ) Stmt

IfStmt          -> if ( Expr ) Stmt ElsePart
```

```
ElsePart        -> else Stmt
                   | ε

ReturnStmt      -> return Expr
                   | return

CompoundStmt    -> { StmtList }

StmtList        -> StmtList Stmt
                   | ε

Expr            -> Attr
                   | Rvalue

Attr            -> identifier = Expr

Rvalue          -> Rvalue Compare LogicalOr
                   | LogicalOr

Compare         -> ==
                   | <
                   | >
                   | <=
                   | >=
                   | !=

LogicalOr       -> LogicalAnd || LogicalOr
                   | LogicalAnd

LogicalAnd      -> Pertinence && LogicalAnd
                   | Pertinence

Pertinence      -> Pertinence <?> Cartesian
                   | Cartesian

Cartesian       -> Cartesian <*> Addition
                   | Addition

Addition        -> Addition + Multiplication
                   | Addition - Multiplication
                   | Multiplication

Multiplication -> Multiplication * Factor
                   | Multiplication / Factor
                   | Term
```

```
Term            -> ( Expr )
                   | - Term
                   | + Term
                   | $ Term
                   | ! Term
                   | { FactorList }
                   | ( Pair )
                   | Factor

Factor          -> identifier
                   | boolean
                   | number
                   | character

FactorList      -> FactorList , Factor
                   | Factor
                   | ε

Pair            -> Factor , Factor
```

3.2. Sintaxe

O código abaixo é um exemplo de uma função em CSet. A função `foo` recebe dois conjuntos de inteiros, `A` e `B`, e um valor inteiro `c` e retorna um conjunto de pares de inteiros. Este conjunto corresponde ao produto cartesiano de `A` e `B` caso `c` pertença a ambos os conjuntos, e ao conjunto vazio, caso contrário.

```
1 set<pair<int, int>> foo (set<int> A, set<int> B, int c)
2 {
3   set<pair<int,int>> result = { };
4
5   if (c <?> A * B)
6     result = A <*> B;
7
8   return result;
9 }
```

Note que a precedência dos operadores garante que a interseção seja realizada antes da verificação de pertinência.

Como é possível perceber, a sintaxe da linguagem é muito semelhante à da linguagem C.

A tabela a seguir contém a especificação dos novos operadores introduzidos:

operador	afixo	cardinalidade
<?>	infixo	binário
<*>	infixo	binário
$	prefixo	unário

Conjuntos podem ser inicializados como vazios, conforme o exemplo acima, ou contendo elementos. Já os pares devem ter seus componentes identificados ao serem inicializados, conforme exemplo a seguir. Ambos podem, também, ser utilizados em operações sem necessariamente estarem associados a uma variável.

```
1 set<int> nat = {1, 2, 3};
2 pair<int, bool> bah = (2, false);
3 bool pert = bah <?> nat <*> {true, false};
```

No exemplo, a variável `pert` receberá o valor `true`.

3.3. Semântica

Esta seção apresentará os principais aspectos semânticos das modificações introduzidas por CSet.

Alguns operadores foram mantidos da linguagem original, mas seu significado foi alterado para o tipo `set`. Sejam `A` e `B` conjuntos do mesmo subtipo, o conjunto resultante corresponderá:

- `A + B` : à união dos dois conjuntos
- `A - B` : a todos os elementos de `A` que não estão em `B`
- `A * B` : à interseção dos dois conjuntos
- `/` : não está definido para o tipo `set`

Podemos também comparar dois conjuntos utilizando o operador == que resultará em `true` se ambos os conjuntos possuírem exatamente os mesmos elementos.

Ocorrerá um erro semântico ao tentar utilizar os operadores acima com dois conjuntos que não possuam o mesmo subtipo.

O significado dos operadores incluídos seguem abaixo. Sejam `A` e `B` conjuntos, x e y seus respectivos subtipos e `c` uma variável do tipo x:

- cardinalidade `$A` : número inteiro que representa o número de elementos do conjunto
- produto cartesiano `A <*> B` : conjunto cujos elementos são pares ordenados entre os elementos dos dois conjuntos operados, possui subtipo `pair<x,y>` e sua cardinalidade será `($A)*($B)`
- pertinência `c <?> A` : booleano correspondente ao valor `true` caso `c` pertença a `A`, e `false`, caso contrário

Os novos operadores estão definidos somente para o tipo `set`.

4. Considerações finais

Projetar a gramática da linguagem é uma etapa indispensável, permitiu antecipar alguns dos problemas que serão enfretados ao construir o compilador para a mesma, além de identificar algumas limitações, como por exemplo, a impossibilidade de se abstrair a ideia de conjuntos infinitos.

5. Referências Bibliográficas

[1] http://www2.ufersa.edu.br/portal/view/uploads/setores/184/AppendixA.pdf [1]

[2] http://www.lysator.liu.se/(nobg)/c/ANSI-C-grammar-y.html [1]

[1] Consultados no dia 16 de agosto de 2015.