Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada

Disciplina: DIM0437 — Linguagens de Programação: Conceitos e Paradigmas
Docente: Umberto Souza da Costa
Discentes: Dogival Ferreira da Silva Junior
Felipe Cortez de Sá
Gabriel Sebastian von Conta
Phellipe Albert Volkmer
Vinícius Araújo Petch

Subproblema 10 — Adaptação ao paradigma lógico

1 Problema

1.1 Produto do problema

O produto desta fase deve ser um documento especificando uma versão lógica da linguagem imperativa definida pelo grupo. O documento deve deixar evidentes as adaptações ncessárias para que a linguagem migre de paradigma.

1.2 Questões

1. Quais são as características principais do paradigma lógico? Analise e compreenda todos os conceitos deste paradigma antes de responder às questões seguintes.
2. Quais diferenças existem entre as noções de variáveis dos paradigmas imperativo e lógico? Qual é o impacto dessas diferenças no projeto de uma linguagem? Qual é o impacto dessas diferenças na pragmática da linguagem?
3. Quais características da linguagem imperativa originalmente projetada poderão permanecer quando da mudança para o paradigma lógico?
4. Quais características da linguagem imperativa originalmente projetada que precisarão ser retiradas quando da mudança para o paradigma lógico?
5. Quais novas características poderão ser integradas à linguagem?
6. Dê exemplos de programas escritos na versão lógica de sua linguagem.

2 Resoluções

1. Programação lógica é um paradigma de programação baseado em lógica formal, utilizando inferência lógica para produzir resultados. Linguagens utilizadas para programação lógica são chamadas linguagens declarativas, pois programas escritos nelas consistem de declarações ao invés de tarefas e estruturas de controle de fluxo. As

declarações tomam forma de proposições em lógica simbólica. A base das linguagens lógicas é o cálculo de predicados.

O cálculo de predicados parte da análise de proposições, usando de uma lógica simbólica. Nesse caso, proposições são objetos de nossa linguagem (constantes ou variáveis) aplicadas a funções que relacionam esses objetos. Por exemplo, em `Homem(Vinicius)`, `Homem` é uma função que relaciona o objeto `Vinicius` definindo se ele é homem ou não. Caso ele seja (fica ao encargo de quem está escrevendo o programa) essa proposição é verdadeira, caso contrário, é falsa. É importante notar que proposições não tem semântica intrínseca, somos nós que decidimos, ou seja, elas significam o que nós quisermos que elas signifiquem.

Uma das características principais das linguagens lógicas são suas semânticas declarativas. A ideia básica é que existe uma maneira simples de determinar o significado de cada expressão, e não depende de como a expressão pode ser usada para resolver um problema. Essas semânticas são consideradas mais simples que as semânticas imperativas.

Programação em uma linguagem lógica é não-procedural, onde programas em tal linguagem não especificam exatamente quais são as etapas para se chegar a um resultado, mas que forma o resultado tem.

2. No paradigma imperativo, variáveis são espaços de memória que alocam um certo tipo de valor. Porém, no paradigma lógico, variáveis são vinculadas a apenas um valor por meio da unificação, funcionando de forma semelhante às variáveis no paradigma funcional. A diferença é que, em caso de backtracking, variáveis podem ter seus valores desvinculados.

   Sem a ideia imperativa de variável, valores não podem ser guardados da forma tradicional. Porém, operações onde valores são guardados e repassados ainda podem ser feitas, a partir do cálculo de predicados. E como o paradigma lógico se baseia nesse cálculo, a falta de variáveis "imperativas"não possui impacto na linguagem.

3. Tipos de variável, criação de tipos, strings (valores que não se encaixam nos outros tipos de variável) serão demarcadas por aspas. Verificação de tipo. Identificadores de variável manterão a regra da linguagem imperativa. Operadores lógicos não são utilizados, porém os operadores aritméticos serão mantidos. Curto-circuito.

4. Como o paradigma lógico não utiliza estruturas de controle (condicionais e laços são trabalhados com cláusulas e recursões), nenhuma delas será mantida. Por causa do formato utilizado nas linguagens lógicas, operadores lógicos não serão utilizados, onde as cláusulas fazem tal papel. Já os operadores aritméticos serão mantidos. Não teremos atribuição de valores a variáveis, ou declaração de variáveis. Não teremos funções, já que predicados são suficientes para suprir suas necessidades.

5. Várias características serão semelhantes ou comparáveis ao Prolog. De início, a linguagem sofrerá várias alterações para poder ser considerada lógica, e com isso entra o uso de cálculo de predicados e o backtracking, características fundamentais mencionadas anteriormente. Tuplas e listas substituirão registros e arranjos, para facilitar no trabalho com conjuntos e grupos de variáveis. Variáveis anônimas serão representadas por _. Ao invés de utilizarmos :-, que define as condições necessárias para que o predicado do lado esquerdo seja verdadeira, nós utilizaremos o símbolo :. No lugar do ,,

utilizado no Prolog para delimitar partes do corpo de uma fórmula, nós utilizaremos a palavra-chave `and`.

Negação será representada pela palavra-chave `not`, substituindo o `!` da linguagem original. Um programa será dividido em domínios, predicados, cláusulas, e a consulta. Essas etapas serão delimitadas pelas palavras-chave domains, predicates, clauses e goal, respectivamente, com utilização de chaves (`{` e `}`). Os domínios serão as variáveis criadas para a execução do programa. Na área de predicados serão declarados protótipos dos procedimentos utilizados no programa. Em cláusulas, serão declaradas os fatos e regras do programa, e na consulta procedimentos serão examinados a partir dos fatos (valores absolutos) e regras (condicionais) das cláusulas. A palavra-chave break terá a função de cut do símbolo `!`, utilizado no Prolog. Para podermos representar laços, serão utilizadas recursões, com auxílio do break ou de uma regra que não continue a recursão (ou comece outra). Os parâmetros de um procedimento servem tanto como parâmetros de saída quanto como parâmetros de entrada.

6. O exemplo seguinte retorna a diferença entre duas listas:

```
domains {
    lint = integer *;
}

predicates {
    diferenca(lint, lint, lint)
    pertence(int,lint).
    remove(int,lint,lint).
}

clauses {
    pertence(X,[X|_]): break;
    pertence(X,[_|L]): pertence(X,L);
    remove(X,[X|L],L): break;
    remove(X,[Y|L1],[Y|L2]): remove(X,L1,L2);
    diferenca([],_,[]): break;
    diferenca(L,[],L): break;
    diferenca(L1,[X2|L2],L3): pertence(X2,L1) and
                              remove(X2,L1,LAUX) and
                              diferenca(LAUX,L2,L3) and
                              break;
    diferenca(L1,[null|L2],L3): diferenca(L1,L2,L3);
}

goal {
    diferenca([1,2,7,3],[1,2,4,3],L);
}
```