Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

UTILIZANDO MORFOL PARA ANÁLISE DE CENAS
ESPECIFICAÇÃO E IMPLEMENTAÇÃO

Aluno: Caio Pimentel Seguin
Orientador: Edward Hermann Haeusler

Introdução

No projeto MORFOL[1] foi desenvolvida a ferramenta MORFOL 1.0, uma solução para o reconhecimento morfossintático de imagens via lógica. A partir de um modelo redigido na Linguagem de Descrição (linguagem lógica semelhante ao Prolog [2]), a ferramenta gera um código em Linguagem C (função de interpretação), que utilizando primitivas de segmentação do Juiz Virtual (TecGraf), é capaz de, dadas cenas e imagens digitais, reconhecer o padrão descrito pelo modelo.

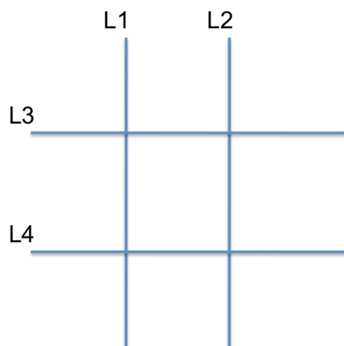
Objetivos

Implementar, em conjunto com o aluno Marco Antônio Teixeira (desenvolvedor do MORFOL 1.0), a ferramenta MORFOL 2.0. Esta nova versão apresenta mudanças visando otimizar o processo de geração de código e código gerado para reconhecimento de padrões, através da adição do operador de corte (!) à Linguagem de Descrição. O projeto objetiva também a integração da ferramenta ao Juiz Virtual, buscando padrões mais eficientes para o reconhecimento de situações em um jogo de futebol.

Familiarização

A primeira etapa do projeto se caracterizou pelo estudo do funcionamento e implementação do MORFOL 1.0, para que, após esse estágio inicial de familiarização, pudessem ser feitas as melhorias pretendidas.

A ferramenta MORFOL, como dito acima, recebe como uma de suas entradas um modelo lógico, que busca descrever um certo padrão bem definido em um conjunto de segmentos. Esse modelo é redigido na Linguagem de Descrição, uma linguagem lógica baseada em PROLOG. A seguir um exemplo de um modelo lógico de um "jogo da velha" feito na Linguagem de Descrição.



JogoDaVelha(x1,x2,x3,x4) = Par(x1,x2), Par(x3,x4), Ort(x1,x3)

Se fossemos expressar com linguagem natural as restrições necessárias para que um certo conjunto de segmentos {L1, L2, L3, L4} descreva um padrão do tipo jogo da velha, diríamos que L1 é paralela a L2, L3 é paralela a L4 e L1 é ortogonal a L3. Essa é exatamente a idéia da Linguagem de Descrição. As variáveis x1, x2, x3 e x4 representam os segmentos a serem analisados em busca do padrão e o operador "," indica conjunção (e). A entrada completa para especificar o padrão acima é:

```
Base:
Ort = Ortogonal
Par = Paralelo

Def:
JogoDaVelha(x1,x2,x3,x4) = Par(x1,x2), Par (x3,x4), Ort(x1,x3)

Cod:
JogoDaVelha(x1,x2,x3,x4)
```

A seção Base abriga as funções primitivas de segmentação utilizadas pela ferramenta. Essas funções são definidas no Juiz Virtual, como veremos mais adiante, e utilizadas para compor as definições locais. A parte Def armazena um número qualquer de definições lógicas, especificando todos os padrões da entrada, nesse exemplo, somente o de um jogo da velha. A área Cod indica qual das funções definidas em Def é a principal, ou seja, qual o padrão que deve ser reconhecido. No exemplo dado, como só temos uma única função local, ela é a função principal.

Uma situação em que teríamos mais de uma função local, seria, por exemplo, o reconhecimento de um campo de futebol. Poderíamos compor a definição lógica de Campo utilizando a de Área, Retângulo, dentre outras. Nesse caso, todas essas funções seriam especificadas em Def, porém somente Campo estaria em Cod.

```
Base:
Par = Paralelo
Ort = Ortogonal
Ent = Entre

Def:
Rec(x1,x2,x3,x4) = Par(x1,x3), Par(x2,x4), Ort(x1,x2)
Fundo(x1,x2,x3) = Par(x1,x3), Ort(x1,x2)
Area(x1,x2,x3,x4,x5,x6,x7) = Rec(x1,x2,x3,x4), Rec(x1,x5,x6,x7), Ent(x5,x2,x7), Ent(x2,x4,x7), Ent(x1,x3,x6)
Campo(x1,x2,x3,x4,x5,x6,x7,x8,x9) = Rec(x1,x5,x6,x7), Area(x1,x2,x3,x4,x5,x6,x7), Fundo(x8,x1,x9),
(Ent(x8,x5,x9),Ent(x8,x7,x9))

Cod:
Campo(x1,x2,x3,x4,x5,x6,x7,x8,x9)
```

A partir do modelo é gerado um código C que busca em um conjunto de segmentos de entrada quais segmentos representam o padrão desejado. A tradução ou transformação do modelo em código C é implementada em TXL [3], uma linguagem de programação intencional (transformacional). Dessa forma, o aprendizado da linguagem TXL, além do entendimento da ferramenta MORFOL, foi um dos pontos cruciais da primeira etapa do projeto.

O propósito do modelo é servir como base para a busca do padrão desejado em um conjunto de segmentos. Tal conjunto é extraído de uma imagem pela ferramenta Juiz Virtual e passado para a ferramenta MORFOL.

O método de busca utilizado pelo código C é o algoritmo conhecido como backtracking, muito usado em Prolog. O procedimento realiza uma busca em profundidade na árvore composta pelos segmentos a serem analisados, percorrendo-a de cima para baixo e da esquerda para direita. Quando algum segmento falha em atender alguma restrição (proposta pelo modelo) ou quando atingimos uma folha (todos os segmentos foram relacionados aos segmentos do modelo) o método de backtraking é acionado, fazendo com que o sistema retorne pelo mesmo caminho percorrido com a finalidade de encontrar soluções alternativas.

Voltando ao exemplo do jogo da velha, ao rodarmos a tradução com o modelo apresentado acima obtemos a seguinte função principal (foram feitas algumas simplificações em relação a função gerada originalmente):

```
int JogoDaVelha (linha * x1, linha * x2, linha * x3, linha * x4)
{
int j = 0;
int i1, i2, i3, i4;

for (i1 = 0; i1 <= MAX; i1 ++) {
        if (i1 == MAX) x1 = NULL;
        else { x1 = & Ent [i1]; }

        for (i2 = 0; i2 <= MAX; i2 ++) {
                if (i2 == MAX) x2 = NULL;
                else {
                if ((i2 == i1)) continue;
                x2 = & Ent [i2];
                 }

                for (i3 = 0; i3 <= MAX; i3 ++) {
                        if (i3 == MAX) x3 = NULL;
                        else {
                        if ((i3 == i1) || (i3 == i2)) continue;
                        x3 = & Ent [i3];
                        }

                        for (i4 = 0; i4 <= MAX; i4 ++) {
                                 if (i4 == MAX) x4 = NULL;
                                else {
                                if ((i4 == i1) || (i4 == i2) || (i4 == i3)) continue;
                                x4 = & Ent [i4];
                                }

                                if (
                                        ( (x1 == NULL) || (x2 == NULL) || Paralelo (* x1, * x2) ) &&
                                        ( (x3 == NULL) || (x4 == NULL) || Paralelo (* x3, * x4) ) &&
                                         ( (x1 == NULL) || (x3 == NULL) || Ortogonal (* x1, * x3))
                                   )
                                   {
                                        Saida [j].x1 = i1;
                                        Saida [j].x2 = i2;
                                        Saida [j].x3 = i3;
                                        Saida [j].x4 = i4;
                                         j ++;
                                   };
                        }
                }
        }
}

return j;
};
```

O código acima é subdivido em cores que indicam certas partes do algoritmo de backtracking. Se uma das restrições em vermelho (evitam repetições) não for atendida, o seu respectivo for passará adiante para sua próxima iteração, implementando assim o processo do backtracking. Os trechos em verde armazenam temporariamente os segmentos que passam pelas restrições em vermelho, até que eles sejam submetidos ao último teste em azul (extraído diretamente do modelo), e caso passem, são guardados como uma solução no trecho em

laranja. Na maioria dos casos, apenas uma solução é esperada, ou seja, o valor retornado em j é 1.

Desenvolvimento

Concluída a fase inicial de estudos, iniciou-se a implementação da nova versão da ferramenta, visando otimizar o processo de geração de código e a eficiência do código gerado.

Após alguma analises, foi concluído que o código TXL precisava ser reestruturado em alguns pontos, o que permitiria a adição das novas funcionalidades desejadas e melhoraria a qualidade do código.

Diversos módulos e funções foram modificados, alguns criados e outros excluídos. Durante tais alterações não foram esquecidos aspectos relativos a modularidade do código, sendo que, sempre que possível, as novas funções eram projetadas visando o máximo reuso, apesar de que, em considerável parte dos casos, a linguagem TXL tendesse a dificultar esse trabalho. A documentação dos módulos também foi um ponto importante, sendo feita detalhadamente nos módulos criados e atualizada em alguns dos já existentes.

Excluindo a reestruturação do código para fins de moduralidade, a principal alteração promovida foi a inclusão do operador de corte (!) na Linguagem de Descrição, juntamente com a implementação do seu tratamento. Esse operador proporciona uma nova funcionalidade ao backtracking, um mecanismo capaz de limitar o espaço de busca durante o processo de reconhecimento de modelos, "podando" determinados ramos da árvore de busca, tornando algoritmo mais eficiente em relação ao tempo de execução e espaço de memória. A seguir um exemplo de um modelo abstrato contendo o operador de corte.

Base:
Ort = Ortogonal
Par = Paralelo

Def:
JogoDaVelha(x1,x2,x3,x4) = Par(x1,x2), Par (x3,x4),!, Ort(x1,x3)

Cod:
JogoDaVelha(x1,x2,x3,x4)

O operador de corte indica quais funções já satisfeitas não precisam ser reconsideras com novos segmentos. Sendo assim, ele fixa todos os segmentos já avaliados até o momento, ou seja, ele interrompe o processo de backtracking.

Na prática, o corte pode ser utilizado para descrever modelos mais eficientes, já que ao diminuirmos o espaço de busca, diminuímos também o tempo de execução do programa. Alem disso, uma aplicação interessante é a sua utilização quando buscamos somente uma solução, ou seja, soluções alternativas são descartadas. Nesse caso, o operador de corte é o último objeto na descrição da função principal.

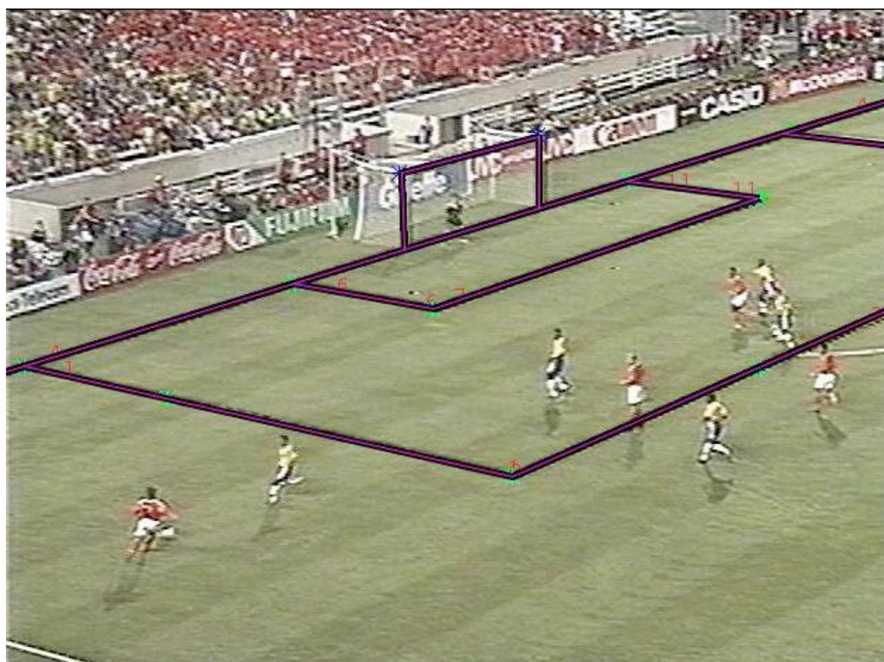
A seguir o código C gerado pela tradução do modelo acima.

```
int JogoDaVelha (linha * x1, linha * x2, linha * x3, linha * x4)
{
int j = 0;
int i1, i2, i3, i4;

for (i1 = 0; i1 <= MAX; i1 ++) {
        if (i1 == MAX) x1 = NULL;
        else { x1 = & Ent [i1]; }

        for (i2 = 0; i2 <= MAX; i2 ++) {
                if (i2 == MAX) x2 = NULL;
                else {
                if ((i2 == i1)) continue;
                x2 = & Ent [i2];
                }

                for (i3 = 0; i3 <= MAX; i3 ++) {
                        if (i3 == MAX) x3 = NULL;
                        else {
                        if ((i3 == i1) || (i3 == i2)) continue;
                        x3 = & Ent [i3];
                        }

                        for (i4 = 0; i4 <= MAX; i4 ++) {
                                if (i4 == MAX) x4 = NULL;
                                else {
                                if ((i4 == i1) || (i4 == i2) || (i4 == i3)) continue;
                                x4 = & Ent [i4];
                                }

                                if (! (Paralelo (* x1, * x2) && Paralelo (* x3, * x4))) continue;

                                if (
                                        ( (x1 == NULL) || (x2 == NULL) || Paralelo (* x1, * x2) ) &&
                                        ( (x3 == NULL) || (x4 == NULL) || Paralelo (* x3, * x4) ) &&
                                        ( (x1 == NULL) || (x3 == NULL) || Ortogonal (* x1, * x3))
                                  )
                                  {
                                        Saida [j].x1 = i1;
                                        Saida [j].x2 = i2;
                                        Saida [j].x3 = i3;
                                        Saida [j].x4 = i4;
                                        j ++;
                                  };

                                break;
                        }
                }
        }
}

return j;
};
```

O trecho em vermelho garante que as funções antes do corte sejam avaliadas somente uma vez, fixando assim os primeiros segmentos que vierem a satisfazê-las.

A integração da ferramenta MORFOL com o Juiz Virtual já havia sido abordada no Projeto MORFOL. Na ocasião, observou-se que, ao trabalhar em conjuntos com aplicações reais, o modelo para descrever o padrão lógico desejado necessitaria de muito mais restrições do que o considerado na fase de implementação. Isso devido a algumas características do conjunto de segmentos fornecido, como por exemplo a possibilidade de certa imagem não

conter todos os segmentos de reta descritos presentes no modelo. Com o novo operador de corte buscamos a criação de padrões lógicos mais eficientes e menos ingênuos, para melhorar a descrição dos modelos.



Segmentos extraídos pelo Juiz Virtual

A integração das duas ferramentas consiste basicamente em automatizar o vínculo entre a geração do código de busca realizada pelo MORFOL, a extração dos segmentos da imagem de entrada pelo Juiz Virtual e a execução do código sobre o conjunto de segmentos.

O Juiz Virtual possui uma função própria que para determinar quais segmentos pertencem ao modelo, porém, ela é baseada em computação gráfica ao em vez de lógica. Assim, para realizar a integração é preciso substituir esse função pela gerada pelo MORFOL. Porém, para isso são precisas alterações no código do Juiz Virtual, que se mostraram bastante trabalhosas.

Conclusões

As alterações promovidas na implementação da ferramenta MORFOL se mostraram positivas. A reestruturação do código TXL deixou o programa mais modularizado e compreensível, o que é muito importante para trabalhos futuros que venham a utilizar a ferramenta.

A inclusão do operador de corte à Linguagem de Descrição tornou-a mais poderosa, capaz de descrever modelos mais eficientes. O novo operador possibilita a diminuição do tempo de execução do reconhecimento, por meio da redução do espaço de busca do algoritmo. Além disso, foi observado que o corte ainda auxilia na exclusão de soluções incorretas.

Foi iniciada a busca por padrões lógicos mais eficientes quando aplicados a aplicações reais, por meio da utilização do operador de corte. Os resultados obtidos até então foram superficiais, mas realmente indicam que a eficiência dos modelos pode ser aumentada pelo corte.

Referências

1 - TEIXEIRA, M. A. B. **Projeto MORFOL: Uma Ferramenta para Análise Lógica de Cenas**. 2009. Projeto de Iniciação Científica, Departamento de Informática - PUC-Rio.

2 - PALAZZO, L. A. M. **Introdução à Programação Prolog**. 1997. EDUCAT, Pelotas.

3 - CORDY, J. R., CARMICHAEL I. H., HALLIDAY, R. **The TXL Programming Language**. Version 10.5, November 2007. **Net**, Disponível em: Acesso em: 04 julho 2010.

4 - SZENBERG, F. **Acompanhamento de Cenas com Calibração Automática de Câmeras**. 2001. Tese de Doutorado - Departamento de Informática, PUC-Rio.

5 - FELIX, M. F. **LET: Uma Linguagem para Especificar Traduções e seu Compilador**. 1998. Dissertação de Mestrado - Departamento de Informática, PUC-Rio.