Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

PROJETO MORFOL
UMA FERRAMENTA PARA ANÁLISE LÓGICA DE CENAS

Aluno: Marco Antônio Barbosa Teixeira
Orientador(es): Edward Hermann Haeusler e
Geiza Maria Hamazaki da Silva

Introdução

Este projeto é uma continuidade do Projeto MORFOL[1], cujo objetivo foi o desenvolvimento de uma ferramenta que possibilita a análise morfológica de uma linguagem, através da descrição de um modelo, com base em uma descrição lingüística em linguagem lógica gerando código na linguagem C. Na ferramenta MORFOL, dado um modelo descrito na Linguagem de Descrição é gerado código em Linguagem C para o mesmo modelo, o qual utiliza primitivas de segmentação do Juiz Virtual(JV) como primeiro estudo de caso. A linguagem de descrição tem como base uma descrição em linguagem lógica semelhante ao Prolog.

Objetivos

Este projeto objetiva dar continuidade no desenvolvimento da ferramenta MORFOL, e concluir o protótipo inicial da ferramenta, onde será realizada a validação, através de testes sobre os códigos gerados, além da integração deste código com o JV. Serão realizadas comparações entre o método utilizado pelo JV, e o método proposto em MORFOL. É esperado obter subsídios para uma análise mais conclusiva do uso de princípios morfológicos em detrimento de reconhecimento estatístico de cenas no domínio do Juiz Virtual. A linguagem utilizada será amplicada com adição do operador de corte (!), que deverá tornar o processamento mais eficiente.

Metodologia

Na primeira etapa foi finalizada a ferramenta de geração de código, sendo aplicados testes, onde o código gerado foi executado para a verificação dos resultados de suas interpretações. Inicialmente, o código foi testado utilizando valores definidos manualmente. Após esse passo o mesmo código foi integrado ao sistema do JV, utilizando segmentos "reais" reconhecidos pelo programa. Com isto, o código foi testado em situações reais, e foi verificado os ajustes à ferramenta para uma fácil integração do código com outros programas.

Após os testes de integração para a validação do código gerado, foram estudadas e acrescentadas novas "funcionalidades" à MORFOL, como a inclusão do símbolo de corte à linguagem, para tratamento e otimização do algoritmo.

Para o desenvolvimento da ferramenta MORFOL, foi utilizada a linguagem de transformação TXL, que é usada para a Transformação/tradução entre o código de especificação de MORFOL e C. E para a integração do código gerado com o do JV, foi utilizado o compilador GCC, o ambiente de programação Visual Studio 2005/2008, com utilização de ferramentas gráficas, como Glut/ openGL, iup, im, e outras.

Desenvolvimento

Após a implementação da ferramenta de tradução, foi inicializado a fase de teste do código gerado. Esta foi realizada em dois momentos: em um primeiro momento, testamos o código gerado em busca de falhas no processo de interpretação realizado pelo código e no seguinte foi realizado a integração do código ao programa do JV.

Nos primeiros testes, executamos o código com uma série de entradas, geradas manualmente, para a verificação das soluções encontradas.

Segue abaixo, um exemplo de um dos testes efetuados, com o código gerado sobre um modelo de um campo de futebol, como o mostrado abaixo.

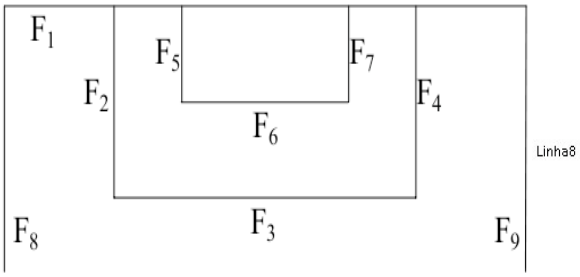

Figura 1- Modelo especificado

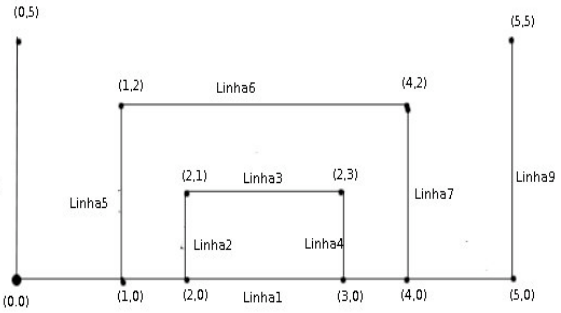

Figura 2 – Segmentos fornecidos

Segmentos fornecidos:

Linha1 = {{ 0.0 , 0.0 } , { 5.0 , 0.0 }},
Linha2 = {{ 2.0 , 0.0 } , { 2.0 , 1.0 }},
Linha3 = {{ 2.0 , 1.0 } , { 3.0 , 1.0 }},
Linha4 = {{ 3.0 , 0.0 } , { 3.0 , 1.0 }},
Linha5 = {{ 1.0 , 0.0 } , { 1.0 , 2.0 }},
Linha6 = {{ 1.0 , 2.0 } , { 4.0 , 2.0 }},
Linha7 = {{ 4.0 , 0.0 } , { 4.0 , 2.0 }},
Linha8 = {{ 0.0 , 0.0 } , { 0.0 , 5.0 }},
Linha9 = {{ 5.0 , 0.0 } , { 5.0 , 5.0 }};

É esperado encontrar o seguinte resultado:

L1:F1 L2:F5 L3:F6 L4:F7 L5:F2 L6:F3 L7:F4 L8:F8 L9:F9

onde Lx : Fy significa que a linha Lx representa a linha Fy do modelo real.

Os resultados obtidos foram:

	L1	L2	L3	L4	L5	L6	L7	L8	L9
0	F1	F5	F6	F7	F4	F3	F2	F9	F8
1	F1	F7	F6	F5	F4	F3	F2	F9	F8
2	F1	F5	F6	F7	F2	F3	F4	F9	F8
3	F1	F7	F6	F5	F2	F3	F4	F9	F8
4	F3	F5	F6	F7	F4	F1	F2	F9	F8
5	F3	F7	F6	F5	F4	F1	F2	F9	F8
6	F3	F5	F6	F7	F2	F1	F4	F9	F8
7	F3	F7	F6	F5	F2	F1	F4	F9	F8
8	F1	F5	F6	F7	F4	F3	F2	F8	F9
9	F1	F7	F6	F5	F4	F3	F2	F8	F9
10	F1	F5	F6	F7	F2	F3	F4	F8	F9
11	F1	F7	F6	F5	F2	F3	F4	F8	F9
12	F3	F5	F6	F7	F4	F1	F2	F8	F9
13	F3	F7	F6	F5	F4	F1	F2	F8	F9
14	F3	F5	F6	F7	F2	F1	F4	F8	F9
15	F3	F7	F6	F5	F2	F1	F4	F8	F9

Figura 3- Resultado dos testes com poucas restrições

Obtivemos 16 possíveis respostas para o modelo. Pode-se observar que o resultado 10 é o esperado. Os outros resultados estão corretos, de acordo com a lógica aplicada no modelo da imagem. Concluimos que é necessário incluir mais restrições para que se obtenha um conjunto de soluções satisfatórias.

Foi gerado um novo modelo de imagem, com mais restrições sobre os segmentos. Os resultados obtidos foram:

	L1	L2	L3	L4	L5	L6	L7	L8	L9
0	F1	F7	F6	F5	F4	F3	F2	F8	F9
1	F1	F5	F6	F7	F2	F3	F4	F8	F9

Figura 4- Resultado dos testes com mais restrições

Considerando que o modelo é simetrico, o resultado foi considerado satisfatório, chegando a um resultado próximo da solução esperada.

Realizada a verificação do código, passamos para a fase de integração do código gerado com o programa Juiz Virtual. Esta fase visa testar o código em situações reais, e verificar os ajustes à ferramenta necessários para uma rápida integração de código com outros programas.

Como primeiro trabalho nesta etapa, foi realizado uma análise dos requisitos necessários para a integração através de uma verificação de todos os itens essenciais para a execução da interpretação, como a disponibilização do conjunto de segmentos para a função de interpretação gerada e a forma de retorno das soluções encontradas de preferência automática, de código com programas em contextos diferentes do aplicado.

Para a verificação de resultados, e o prosseguimento da implementação e da análise das possíveis modificações, foi implementada um conjunto de funções para ser utilizado temporariamente na interligação entre JV e a função de interpretação.

Um problema previsto no código C gerado é que, quando este faz a interpretação dos segmentos, espera que todos os segmentos a serem encontrados estejam presentes na base, i.e., presentes no conjunto de segmentos fornecidos. Porém, é comum o detector de segmentos, que é executado antes do interpretador, encontrar apenas parte dos segmentos pertencentes à imagem tratada, no caso o campo de futebol.

Para que o interpretador pudesse tratar este problema, é necessário que o interpretador seja capaz de encontrar o conjunto parcial dos segmentos da imagem. Desta forma, o interpretador deverá assumir que um segmento a ser encontrado não esteja na base, e ignorá-lo do modelo. Com isso, será necessário ignorar da verificação todas as restrições onde um dos parâmetros é um segmento "ignorado".

A partir das novas hipóteses, era esperado um aumento na quantidade de possiveis soluções ao modelo, pois seriam aplicadas menos funções que restringem algumas combinações de segmentos. Além disso, era esperado que casos "triviais", como nenhum segmento encontrado, que é sempre verdadeiro, aparecesse como resultado das interpretações. Tais problemas foram confirmados nos testes. Foram encontradas nas interpretações cerca de 70000 possiveis soluções para o modelo.

Diante destes resultados se fez necessário pesquisar novas restrições às soluções obtidas no reconhecimento. Foram aplicadas restrições, que limitavam o número mínimo de segmentos existentes na solução, reduzindo para aproximadamente 10000 o número de soluções encontradas.

Após um estudo sobre o JV, algumas restrições observadas foram inseridas no processo: restrições de existência de segmentos considerados essênciais para a obtenção do modelo em 3D foram adicionadas ao teste para filtrar mais opções incorretas. Foram considerados essenciais todos os segmentos paralelo à linha de fundo, sendo necessários mais de 2 segmentos perpendiculares.

Além destas, também foram inseridas restrições relacionadas à direção de um conjunto de segmentos em relação à outro. Por exemplo, todos os segmentos do modelo devem estar em um mesmo semiplano em relação à linha de fundo. Ou, nenhum dos segmentos podem se cruzar. Aplicadas as novas restrições, o resultado obtido foi da ordem de 300 soluções.

Atualmente, está sendo realizado pesquisas sobre outras formas de filtrar os segmentos ainda incorretos. O trabalho está sendo realizado junto à equipe de desenvolvimento do JV para a melhoria das funções de base disponibilizadas e outras formas de restrição.

Além disto, é importante citar a inclusão do operador de corte (!). O operador de corte é um recurso comum à linguagens como Prolog, evitando o processamento de certos ramos da árvore de busca.

O operador de Corte atua indicando ao interpretador que todos os termos anteriores ao simbolo sejam sucedidos antes de continuar o processo de interpretação. O operador atua também de forma que o resultado anterior não seja alterado, isto é, todas os segmentos definidos anteriormente deverão se manter inalterados. Este processo corta do processamento os termos anteriores ao operador, tornando este mais eficiente, tanto em relação a espaço de memória como tempo de processamento, pois evita o reprocessamento dos termos anteriores a este.

Abaixo, podemos observar um exemplo simplificado da transformação efetuada. Observe que os termos que aparecem antes do operador são verificados no comando *if(!...)continue;* antes de continuar o processamento, e o corte é feito posteriormente pelo comando *return ERRO ,* após todo o processamento posterior.

Exemplo de Especificação em MORFOL(aplicando corte)	*Saída obtida pela transformação*

Base:
b = B
c = C
d = C
d = D
e = E
f = F
g = G
 	
Def:
 	
A(xa,xb,xc,xd,xf,xi,xj)=b(xa,xb,xc),c(xb,xa,xf) , ! , d(xf,xd,xi) , \
e(xi,xj,xb) , ! , f(xg,xa,xd) , g(xa,xc).
 	
Cod:
A(xa,xb,xc,xd,xf,xi,xj)
	A(xa,xb,xc,xd,xf,xi,xj){
	for(xa=0; xa < MAXLINHAS; xa ++){
	for(xb=0; xb < MAXLINHAS; xb ++){
	for(xc=0; xc < MAXLINHAS; xc ++){
	for(xf=0; xf < MAXLINHAS; xf ++){
	if(! B(xa,xb,xc)) continue ;
	if(! C(xb,xa,xf)) continue ;
	for(xd=0; xd < MAXLINHAS; xd ++){
	for(xi=0; xi < MAXLINHAS; xi ++){
	for(xj=0; xj< MAXLINHAS; xj++){
	if(!D(xf,xd,xi)); if(!E(xi,xj,xb));
	for(xg=0; xg< MAXLINHAS;xg++){
	if(F(xg,xa,xd)&&G(xa,xc)){
	return OK; }}
	return ERRO;
	}}}
	return ERRO;
	}
	}}}return ERRO;}

Figura 5- Exemplo dos padrões de entrada e saída

Foram realizadas, além disso, correções e inclusões na ferramenta de tradução MORFOL, como, a geração automática da função que realiza a intermediação entre a função gerada e a aplicação, sendo necessário a inclusão de outras diretivas, para especificação da função intermediária.

Na linguagem de descrição, poderão ser inseridas informações importantes ao modelo, como nome da função a ser gerada, segmentos essênciais do modelo, e outras restrições do modelo.

Além dos mecanismos para filtragem de soluções incorretas, estão sendo estudados formas de melhorar o tempo de resposta através de análise de complexidade e mecanismos de otimização.

Conclusões

A utilização de uma linguagem de descrição para reconhecer padrões estruturais da imagem que estejam formalmente relacionados apresentou resultados interessantes como a facilidade na descrição do protótipo ou modelo (não se limitando somente à área de PDI), a rápida implementação do mesmo, além da geração de um código legível. A integração do código gerado com outra aplicação levantou novos problemas que muitas vezes não são visualizados na etapa de projeto da aplicação. É interessante citar a inclusão de comandos de descrição para auxilio na geração automática de funções intermediárias, e para estabelecer critérios de seleção de soluções "preferenciais", mesmo que o desenvolvimento e o projeto desta linguagem sejam parciais e incompletos. E o desenvolvimento de uma linguagem de fácil descrição para determinação dos critérios válidos para a seleção da solução. A inclusão do operador de corte à linguagem tornou o processo de reconhecimento mais eficiente, além de auxiliar na eliminação de soluções incorretas

Como trabalho futuro podemos citar o aperfeiçoamento da ferramenta MORFOL, através de alterações na ferramenta para otimização tanto no processo de geração de código como no código gerado. Podemos citar também o estudo sobre a viabilidade da utilização da ferramenta em outras aplicações.

Referências

[1] TEIXEIRA, M.A.B. PROJETO MORFOL: Uma Ferramenta para Análise Lógica de Cenas – Projeto de Iniciação Científica - Departamento de Informática - PUC-Rio. 2008.
[2] FELIX, M. F. LET: Uma Linguagem para Especificar Traduções e seu Compilador. Dissertação de Mestrado - Departamento de Informática - PUC-Rio. 1998.
[3] Szenberg, F. Acompanhamento de Cenas com Calibração Automática de Câmeras. Tese de Doutorado - Departamento de Informática - PUC-Rio. 2001.
[4] Palazzo,L. A. M. Introdução à Programação Prolog. EDUCAT, Pelotas, 1997.
[5] http://www.w3.org/Style/XSL/
[6] http://www.txl.ca