Tabela de Identificadores CMinus - Resolução com FermaT

Miguel Regedor
miguelregedor@gmail.com

André Santos
pg15973@alunos.uminho.pt

Análise e Transformação de Software - Engenharia de Linguagens
Mestrado em Engenharia Informática
Universidade do Minho

Resumo A identificação, no momento de parsing, de variáveis usadas mas não declaradas no código-fonte de um programa, além de útil, é um bom exercício que permite aplicar conhecimentos nas áreas de Engenharia Gramatical e Análise e Transformação de Programas.
Neste relatório fazemos uma breve introdução ao FermaT, um sistema de análise e transformação de software, e descrevemos como este poderia ser usado para resolver o problema descrito.

1 Introdução

Este trabalho foi realizado no âmbito do módulo de Análise e Transformação de Software da UCE de Engenharia de Linguagens do Mestrado em Engenharia Informática da Universidade do Minho.

Foi-nos pedido que fizéssemos uma investigação sobre como resolver o anteriormente abordado problema da identificação de variáveis não declaradas em código CMinus e sua inclusão na tabela de identificadores a ser construída, usando um sistema de transformação de software à nossa escolha. Optámos por experimentar o FermaT, um sistema baseado na linguagem WSL.

Neste documento começamos por introduzir o FermaT e as suas principais características. Depois passamos a explicar as principais funcionalidades que suportam o nosso modo de resolução. Posteriormente descrevemos como resolver o problema referido, e no fim apresentamos algumas conclusões.

2 Apresentando o FermaT

O FermaT é descrito pelos seus autores como um poderoso sistema de transformação, com capacidade para realizar operações à escala industrial. Este sis-

tema baseia-se na linguagem WSL (abreviatura para *Wide Spectrum Language*), e está agora disponível segundo a licença GPL.

O FermaT tem sido usado com sucesso em projectos de migração de código máquina para código C e COBOL (alguns destes projectos eram de grande dimensão, envolvendo a conversão de milhões de linhas de código máquina escritas à mão para código C e COBOL mais eficiente e mantível).

Apesar das referências ao COBOL, pareceu-nos que este projecto se tem mantido activo, uma vez que as ultimas actualizações foram realizadas há menos de um ano.

2.1 Modo de funcionamento

O sistema FermaT actua de acordo com os seguintes passos:

- Tradução do código original para código WSL
- Alteração do código WSL de acordo com o pretendido
- Tradução do código WSL alterado para a linguagem destino

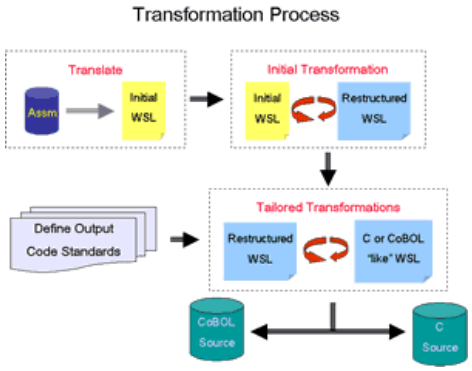

Figura 1. Processo de transformação com FermaT

3 WSL

A WSL é uma linguagem de espectro largo, o que signifca que suporta um vasto leque de estruturas possíveis para traduzir um programa ou algoritmo para WSL.

O "Espectro" em "Linguagem de Espectro Largo" refere-se ao leque de operações, desde operações de "baixo nível" como atribuições, instruções IF e *gotos* a operações de nível mais elevado, como instruções de especificação.

Além das estruturas e comandos de programação habituais, a WSL contém ainda comandos, funções e rotinas que operam sobre programas escritos em WSL, o que disponibiliza ao utilizador uma ferramenta para analizar, reescrever, e simplificar programas (e mesmo para escrever programas que analizam e reescrevem programas).

```
begin
  q := split(input, same_item);
  output :=header ++ process * q
              ++ ⟨⟨"NUMBER CHANGED = ", ℓ(q)⟩⟩
where
funct same_item(x, y) ≡ x.writem = y.writem.
funct process(L) ≡ ⟨L[1], +/change * L⟩.
funct change(s) ≡
  if s.wrtype ≠ "R" then −s.wrqty else s.wrqty fi.
end
```

Figura 2. Exemplo de uma especificação em WSL

A teoria de transformação da WSL baseia-se em *infinitary logic*, uma extensão da lógica de primeira ordem que admite fórmulas de comprimento infinito.

O núcleo (*kernel*) da linguagem WSL é composto por quatro instruções básicas e 3 instruções compostas. De entre as instruções básicas, aquela que nos vai interessar é a instrução para adicionar variáveis: **add**(x).

4 Adicionar Variáveis

A instrução **add**(x) adiciona as variáveis em x ao *state space* a atribui-lhes um valor arbitrário. Se as variáveis já estivessem no *state space*, ainda assim recebem valores arbitrários.

A principal questão que se levanta com este modo de funcionamento é: como implementar uma atribuição tradicional quando a única forma de alterar o valor de uma variável é atribuir-lhe um valor arbitrário. Este efeito pode ser alcançado com o auxílio de instruções de guarda que restringem a não determinância anterior. Por exemplo, a atribuição $x = 1$ pode ser implementada pela sequência, em WSL, **add**(x)**;** $[x = \mathbf{1}]$.

5 A nossa solução

Sem um conhecimento mais aprofundado tanto do FermaT como da linguagem WSL não nos é possível implementar correctamente uma solução. Ainda assim,

baseados nos artigos que encontramos acerca do modo de funcionamento de ambos concebemos uma solução para o problema da inserção na tabela de símbolos de variáveis não declaradas.

Esta solução assenta na funcionalidade descrita anteriormente: a capacidade de adicionar variáveis ao *state space*.

A nossa solução passa então pela conversão do código CMinus para código WSL. Durante esta conversão, todas as ocorrências de atribuições em C deverão ser substituídas pelas instruções respectivas em WSL, usando a instrução **add** recorrendo às guardas para definição do valor a atribuir.

Depois de gerado o código WSL, volta a converter-se para CMinus, tendo o cuidado de, dentro de cada *scope*, inserir as declarações de todas as variáveis contidas no respectivo *state space*.

A maior falha no que respeita a esta solução provém do facto de, apesar de várias vezes mencionado, não estar disponível em lado algum o conversor, em WSL, entre C e WSL (ou vice-versa). Isto significa que, para implementar esta solução, teria primeiro que se escrever os conversores C-WSL e WSL-C.

6 Conclusão

O FermaT aparenta ser um sistema robusto, capaz de realizar os mais diversos tipos de transformações em larga escala.

Parte desta capacidade de lidar com exemplos bastante volumosos provém da linguagem WSL, uma linguagem de espectro largo (*wide spectrum language, WSL*), que possui simultaneamente instruções de baixo nível e de alto nível.

O FermaT apresenta a vantagem de ser uma aplicação *open source* e actualizada regularmente.

Referências

1. M. Ward, *Proving program refinements and transformations*, University of Oxford, 1989.
2. _____, *Derivation of a sorting algorithm*, Tech. report, Durham University, Technical Report, 1990, 1990.
3. M. Ward and T. Hardcastle, *WSL Programmer's Reference Manual*, 2003.
4. MP Ward, *Assembler to C migration using the FermaT transformation system*, IEEE International Conference on Software Maintenance, 1999.(ICSM'99) Proceedings, 1999, pp. 67–76.
5. _____, *Pigs from sausages? Reengineering from assembler to C via FermaT transformations*, Science of Computer Programming **52** (2004), no. 1-3, 213–255.
6. MP Ward and KH Bennett, *A practical program transformation system for reverse engineering*, Reverse Engineering, 1993., Proceedings of Working Conference on, 1993, pp. 212–221.

7. MP Ward, H. Zedan, and T. Hardcastle, *Conditioned semantic slicing via abstraction and refinement in fermat*, Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on, 2005, pp. 178–187.