Tabela de Identificadores CMinus - Resolução com DMS

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 DMS, 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 o software DMS.

Neste documento começamos por introduzir o DMS 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 DMS

O DMS é descrito por alguns dos seus autores como um sistema de análise e transformação de programas prático e comercial, cujo sucesso advém de um

conceito - design de manutenção - e da construção da infraestrutura que aparenta ser necessária para suportar esse conceito.

O DMS lida com a escalabilidade dos programas através de uma cuidadosa gestão de espaço, escalabilidade computacional através de paralelismo e escalabilidade de aquisição de conhecimento através de domínios.

De acordo com a visão do DMS, a evolução de software pode ser gerida através de ferramentas automáticas, baseada nestas ideias:

- Software system design can be captured as a formal artifact explaining what, how and why.
- Desired changes to software in specification, performance, and implementation technology can be explicitly captured as formal maintenance deltas.
- Maintenance deltas can be used to incrementally and mechanically update the design, realizing the desired change while keeping the design current.

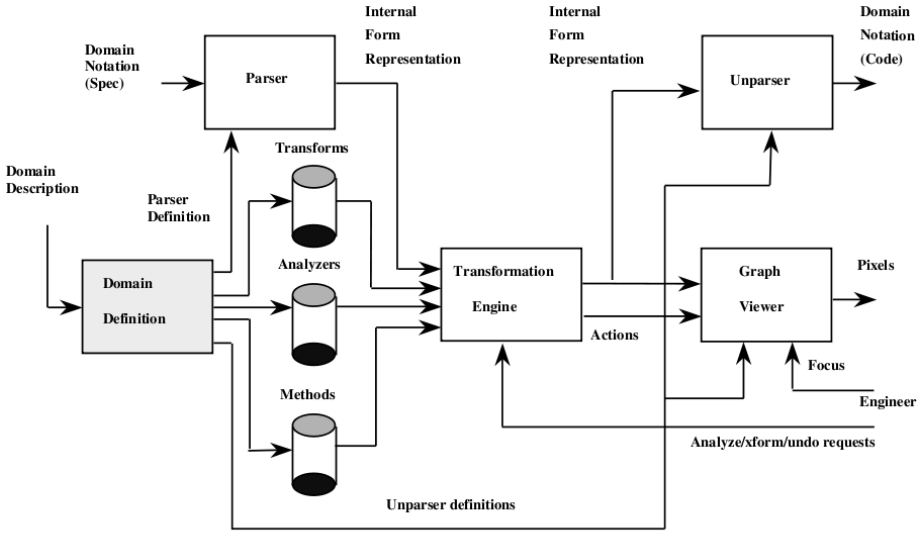

Figure 1: DMS Architecture

3 PARLANSE

De acordo com os autores do DMS, um dos principais problemas na análise e transformação de programas tem a ver com o tamanho de sistemas de software grandes, sendo o DMS capaz de ler dezenas de milhar de linhas de código numa única sessão, permitindo análise de sistemas inteiros.

Para ajudar a aliviar o problema, o DMS está implementado numa *fine grain parallel language* - PARLANSE. PARLANSE foi concebida para suportar paralelismo irregular para manipulação simbólica, induzida por grandes estruturas de dados irregulares tais como AST coom milhões de linhas, em multiprocessadores com memória partilhada.

Foi desenhada inspirada no C, mas com sintaxe similar a LISP. É estaticamente tipada, incluindo tipos de dados escalares, permitindo que, mesmo quando não é possível correr em paralelo, seja executado código muito eficiente.

O paralelismo de PARLANSE é baseado na ideia de *grãos computacionais*, que consistem em bocados de código relativamente pequenos que podem ser executados em paralelo.

4 Suporte para Tabelas de Símbolos

Apenas os sistemas notacionais muito simples são independentes do contexto. Sistemas usados para fins práticos (C++, XML, ...) têm todos regras para nomear entidades e sistemas complexos para gerir os *namespaces* enormes que ocorrem na prática.

O DMS disponibiliza um sistema geral de tabelas de símbolos com funcionalidades tais como definir e guardar informação relativa às associações nome/tipo nos *symbol spaces*, onde os nomes são strings Unicode arbitrárias e os tipos são estruturas PARLANSE arbitrárias (incluindo referências para outros *symbol spaces*).

Tipicamente uma tabela de símbolos é instanciada por um avaliador de atributos, com as referências aos *symbol spaces* percorrendo a árvore de parsing, e os "efeitos colaterais"(acções que podem ser definidas para serem executadas durante uma procura por um símbolo) inserindo novas entradas nos *symbol spaces*.

Os sistemas de reescrita puros (TXL, XT, ...) tentam evitar suporte para tabelas de símbolos, obrigando os utilizadores a codificar manualmente essas funções.

5 Regras de reescrita

O DMS permite regras de reescrita *source-to-source*. Uma regra de reescrita típica tem a seguinte forma:

$$LHS \rightarrow RHS \textbf{ if } condition$$

onde tanto o LHS como o RHS representam padrões da linguagem fonte, com variáveis para representar substrings da linguagem, bem formadas e de comprimento arbitrário.

A condição **if** é uma frase opcional que se refere às variáveis contidas no padrão LHS. Estas regras são interpretadas como, "quando uma parte de um programa coincide com o LHS, substui por RHS se a condição for verdadeira". A condição pode ser implementada como restrições de *matching* adicionais, ou como uma chamada num procedimento de decisão implementado em PARLANSE.

Segues-se um exemplo de uma regra de reescrita:

```
default domain C.
rule auto_inc(v:lvalue):
  statement->statement =
  "\v = \v+1;" rewrites to "\v++;"
  if no_side_effects(v).
```

6 A nossa solução

Sem um conhecimento mais aprofundado tanto do DMS como da linguagem PARLANSE não nos é possível implementar correctamente uma solução. Ainda assim, baseados nos artigos que encontramos acerca do modo de funcionamento do DMS e da linguagem PARLANSE concebemos uma solução para o problema da inserção na tabela de símbolos de variáveis não declaradas.

Esta solução assenta nos tópicos cobertos nas secções anteriores: o suporte para tabelas de símbolos e as regras de reescrita.

A nossa solução passa pela criação de uma regra de reescrita a aplicar à produção que simboliza um identificador, para que, sempre que é feito o parsing de um novo identificador, este seja inserido na tabela de símbolos, com a condição de ainda não se encontrar lá. Nesta solução ficaria a faltar a dedução do tipo de cada identificador, e teria que ser feito um controlo dos vários *scopes*, à semelhança da solução que apresentámos anteriormente, nas aulas de EL, escrita em Antlr.

Uma alternativa seria usar o suporte para tabelas de identificadores que vem integrado com o próprio DMS, inserindo os símbolos à medida que fossem encontradas expressões que fizessem referência a símbolos não declarados. Nos artigos que lemos pareceu-nos que o DMS facilita a gestão de *scopes* (denominados *symbol spaces*). Na solução anterior também poderia ser usada estas tabelas de indentificadores internas.

7 Conclusão

O DMS é um sistema bastante poderoso e que, de acordo com os seus autores, é um dos poucos capazes de lidar com análise e transformação de sistemas complexos.

Parte dessa capacidade de lidar com exemplos bastante volumosos advém da linguagem PARLANSE, desenvolvida de raíz para este propósito, e que tira partido de paralelismo e gestão eficiente de memória para desempenhar a sua função.

Esta ferramenta possui várias funcionalidades interessantes, entre as quais o suporte para a criação e manutenção de tabelas de identificadores e a possibilidade de definir regras de reescrita.

Como pontos negativos, o DMS apenas corre em sistemas Windows, e não é *open source*. Aparentemente, não existe sequer um compilador para PARLANSE tirando o fornecido pela empresa Semantic Designs Inc.

Referências

1. Baxter, I.D. and Mehlich, M.: Preprocessor conditional removal by simple partial evaluation. Proceedings of the Working Conference on Reverse Engineering. (2004) 281–290
2. Baxter, I.D. and Pidgeon, C. and Mehlich, M.: DMS®: Program transformations for practical scalable software evolution: Proceedings of the 26th International Conference on Software Engineering IEEE Computer Society Washington, DC, USA. (2004) 99–116