Geração de Modelos a partir de código C

Clei A. de Souza Junior
Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brasil
casjunior@inf.ufrgs.br

Introdução

Modelos que descrevem o software são úteis para o entendimento de seu funcionamento e podem ser usados para a validação e verificação de propriedades em ferramentas. Apesar de sua importância, tais modelos podem não ser triviais de se construir e, no caso de um sistema já implementado, pode não haver um modelo ou ele pode estar desatualizado. Neste caso, tais modelos podem ser obtidos de um código existente através de um processo de extração de modelos [Holzmann99]. Uma abordagem existente [Duarte06] permite a geração de um modelo LTS (*Labelled Transition Systems*) [Keller76] a partir de código Java, através da ferramenta LTSE (LTS Extractor) [Duarte06]. Tal ferramenta utiliza rastros de execução produzidos por uma versão instrumentada do código para a geração do modelo LTS, o qual pode ser visualizado e analisado na ferramenta LTSA [Magee06]. Para o desenvolvimento deste trabalho, foi usado uma base inicial de regras já existente para a linguagem C, que foi expandida e validada.

Objetivo

O objetivo deste trabalho é expandir as regras já existentes para a linguagem C, abrangindo assim uma maior quantidade de estruturas da linguagem, assim como validar as regras criadas e a ferramenta desenvolvida para a geração de modelos LTS.

Metodologia

Partiu-se do trabalho existente, que consistia em uma base inicial de regras para a linguagem C, desenvolvidas baseando-se em uma abordagem que gera modelos para a linguagem Java. Assim como na abordagem citada, utilizou-se a linguagem TXL (Turing eXtender Language) [Cordy02] para a expansão das regras de transformação de código, já que as regras iniciais já tinham sido desenvolvidas nesta linguagem, com o objetivo de instrumentar o código C para a geração de traces mais completos que pudessem ser utilizados pela LTSE para construir modelos LTS.

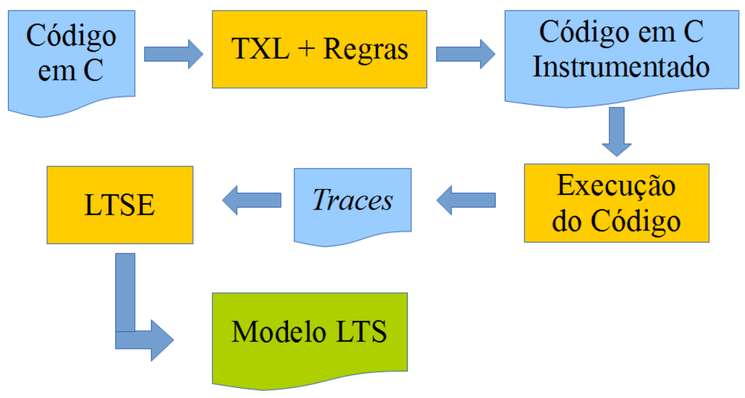

Fig. 1: Processo de geração de modelo

Instrumentação: Para o desenvolvimento deste trabalho, foram ampliadas as regras escritas de transformação baseadas nas regras para o Java usando a linguagem TXL. As novas regras foram escritas de forma modular, mas testadas em conjunto com as outras regras já existentes. Concluída a expansão das regras, as mesmas foram utilizadas para instrumentar código fonte de programas mais complexos que os previamente suportados, escritos na linguagem C.

```
while (i < j) {
    if (i < 5) {
        c = metC();
    } else {
        c = metA();
    }
    i++;
}

while (j > 0) {
    j = metDecInt(j);
    f = metIncFloat(f);
}
```

Fig. 2: Exemplo de código

Metodologia (cont.)

```
{
    while (j > 0) {
        fprintf (trace, "REP_ENTER:(j > 0)#true#Main=%d#{", getpid ());

        fprintf (trace, "}#6;");
        {
            {
                fprintf (trace, "INT_CALL_ENTER:metDecInt#Main=%d#{", getpid ());

                fprintf (trace, "}#2;");
                j = metDecInt (j);

                fprintf (trace, "INT_CALL_END:metDecInt#Main=%d#{", getpid ());

                fprintf (trace, "}#2;");
            }

            {
                fprintf (trace, "INT_CALL_ENTER:metIncFloat#Main=%d#{", getpid ());

                fprintf (trace, "}#3;");
                f = metIncFloat (f);

                fprintf (trace, "INT_CALL_END:metIncFloat#Main=%d#{", getpid ());

                fprintf (trace, "}#3;");
            }

        } fprintf (trace, "REP_END:(j > 0)#Main=%d#6;", getpid ());
    }

    fprintf (trace, "REP_ENTER:(j > 0)#false#Main=%d#{", getpid ());

    fprintf (trace, "}#6;");
    fprintf (trace, "REP_END:(j > 0)#Main=%d#6;", getpid ());
}
```

Fig. 3: Código da fig. 2 após aplicação das regras

Geração de *traces*: Os programas instrumentados pela aplicação das regras foram executados, obtendo-se um conjunto de traces de cada um.

```
REP_ENTER:(j > 0)#true#Main=8060#{}#6;
INT_CALL_ENTER:metDecInt#Main=8060#{}#2;
MET_ENTER:metDecInt#Main=8060#{}#9;
MET_END:metDecInt#Main=8060#{}#10;
INT_CALL_END:metDecInt#Main=8060#{}#2;
INT_CALL_ENTER:metIncFloat#Main=8060#{}#3;
MET_ENTER:metIncFloat#Main=8060#{}#10;
MET_END:metIncFloat#Main=8060#{}#11;
INT_CALL_END:metIncFloat#Main=8060#{}#3;
REP_END:(j > 0)#Main=8060#6;
```

Fig. 4: Treixo do *trace* gerado pela execução do código da fig. 3

Extração de modelos: A partir dos traces, utilizou-se a ferramenta LTSE para gerar os modelos de comportamento dos códigos analisados.

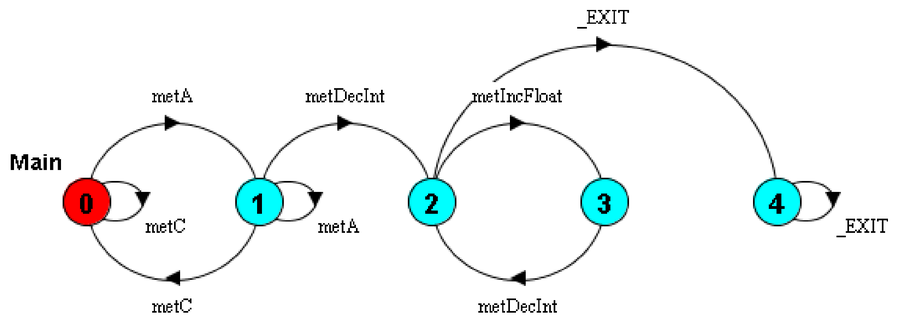

Fig. 5: Modelo LTS gerado a partir do código da Fig. 2

Validação dos modelos: Depois de garantida a funcionalidade básica do sistema, por meio de testes em programas de um só módulo, foi passado a fase de validação das funcionalidades em sistemas maiores, com mais de um módulo. O algoritmo alvo escolhido foi um modelo de produtor/consumidor, onde após a intervenção manual para a separação dos módulos nas anotações, foi obtidos modelos que condizem com o esperado e descrevem o comportamento esperado deste sistema.

Resultados

Foi possível expandir o conjunto de regras já existente para um subconjunto consideravelmente maior da linguagem C, assim permitindo a aplicação do processo em sistemas reais de vários módulos, com pequena interferência manual. Agora, da linguagem C, apenas não são cobertos vetores.
Também foi possível uma maior validação dos modelos obtidos, que acabaram por confirmar os resultados dos testes modulares de que os modelos condizem com o comportamento do sistema modelado.

Conclusões

A ferramenta agora cobre a linguagem C de forma geral, assim como os modelos gerados condizem com o esperado, pretendendo-se então a aplicação desta ferramenta em sistemas de larga escala. Espera-se também explorar o uso dessa abordagem na geração de outros tipos de modelos de comportamento (e.g., Gramáticas de Grafos) e o uso dos modelos para validação e verificação.

Referências

[Holzmann99] Holzmann, G.J., Smith, M.H. "A Practical Method for Verifying Event-Driven Software", ICSE, 1999, pp. 597-607.

[Duarte06] Duarte, L.M., Kramer, J., Uchitel, S. "Model Extraction Using Context Information", LNCS 4199, pp. 380-394.

[Keller76] Keller, R.M. "Formal Verification of Parallel Programs", CACM, v.19, n.7, 1976, pp. 371-384.

[Magee06] Magee, J., Kramer, J. "Concurrency: State Models and Java Programming", 2nd edition, Wiley & Sons, 2006.

[Cordy02] Cordy, J. R. et al. "Source Transformation in Software Engineering Using the TXL Transformation System", JIST, v.44, n.13, 2002, pp. 827-837.