Verigraph: uma ferramenta para a reescrita de grafos

Thiago Rafael Becker
Universidade Federal do Rio grande do Sul - Instituto de Informática
trbecker@inf.ufrgs.br

Introdução

O projeto Verites busca desenvolver técnicas de verificação, validação e testes para o desenvolvimento de sistemas computacionais, e desenvolver ferramentas de apoio para as técnicas desenvolvidas. Gramática de grafos é um modelo de computação, em que estados de um sistema são descritos por grafos, e computações são descritas por um conjunto de regras de reescrita, misturando a intuitividade da representação gráfica com uma semântica de execução precisa. Uma abordagem semelhante para representar sistemas e suas computações são os diagramas de classes e sequência de UML, uma linguagem de especificação gráfica sem semântica precisa. Trabalhos foram propostos para aproximar os dois modelos[2], sendo assim possível executar modelos de verificação formal em modelos UML. Dentro deste cenário, podemos considerar a evolução de software (modificações que são feitas no software durante seu desenvolvimento e manutenção) através do modelo de gramáticas de grafos de segunda ordem[3]. Assim, é possível verificar como modificações em uma parte do software afetam o sistema inteiro.

Em gramáticas de grafos, a sequência de estados criados pela aplicação de reescritas forma um *espaço de estados*, no qual é possível aplicar métodos de verificação formal, tais como diversos modelos de lógica modal, sendo os mais comuns CTL (*computational tree logic*) e LTL (*linear temporal logic*). Estas técnicas pertencem a classe de verificações dinâmicas, pois verificam a execução do modelo. Em contraste, a classe de verificações estáticas verificam a estrutura do modelo sem executá-lo antes. As ferramentas de tranformação de grafos disponíveis atualmente podem verificar usando apenas uma das destas duas classes: AGG[5] faz verificações estáticas, e Groove[4] executa verificações dinâmicas. Ambas executam apenas transformações de primeira ordem.

O objetivo deste trabalho é desenvolver uma ferramenta de transformação de grafos que possa simular a execução de uma especificação e executar verificações formais estáticas e dinâmicas sobre os espaços de estados resultantes destas transformções.

Transformação de grafos

Para implementar a transformação de grafos, foi escolhido o modelo *double push-out*[1], pois este foi usado originalmente para desenvolver a teoria de transformações de segunda ordem. O modelo usa um grafo de estado e um conjunto de regras de reescrita que adicionam ou removem elementos do grafo gerado um novo estado do sistema. Na Figura 1 é dada um gramática de exemplo, que ordena uma sequência de caracteres; o estado inicial é dado po $s0$ e a regra de ordenação é feita pela regra $r0$. Na regra, o grafo L é o grafo que precisa ser encontrado no estado atual. Os elementos não apontados pelo morfismo $l : K \to L$ são os elementos que são eliminados pela regra, enqunto os elementos não apontados pelo morfismo $r : K \to R$ são os elementos adicionados no novo estado. A sequência de aplicações destas regras forma um espaço de estados, como o demonstrado na Figura 2.

Verificação de propriedades

É interessante poder verificar propriedades do espaço de estados de um sistema. Existem modelos de verificação propostos, o os mais usados estão as lógicas temporais, LTL, e as lógicas de árvore de computações, CTL. Ambos os modelos usam os operadores da lógica clássica, e acrescentam operadores que navegam entre os estados. Em LTL, os operadores navegam em profundidade no espaço de estados (a partir de um estado, verifica se uma propriedade é válida nos próximos estados), e em CTL, além dos operadores em profundidade, são adicionados quantificadores de caminho (a partir de um estado, verifica se os caminhos gerados aderem a determinada propriedade). Na lógica para descrição das propriedades do modelo, as fórmulas atômicas correspondem ao nome das regras aplicadas ao grafo-estado. Uma fórmula atômica é verdadeira se no grafo-estado exite um match para ela e false caso contrário. No futuro, planejamos a possibilidade de verificar propriedades do próprio estado, como a existência de um determinado elemento no grafo. No exemplo (Figura 2, os estados $s0$ a $s8$ possuem a propriedade $r0$, enquanto o estado $s9$ não possui propriedade nenhuma. A verificação de uma propriedade é feita em todos os estados do espaço de estados.

Por exemplo, a regra $r0 \Rightarrow E[F(r0)]$ verifica se a partir de um estado, se regra $r0$ pode ser aplicada, existe um caminho (E) em que em algum estado futuro (F) a regra $r0$ também pode ser aplicada. A fórmula anterior é válida em todos os estados do exemplo na Figura 2. Um exemplo de uma verificação que falha no espaço de estados apresentado é a fórmula $r0 \wedge E[F(r0)]$, pois no estado $s8$ é possível aplicar a regra $r0$, porém no estado futuro $s9$ não é possível aplicar esta regra.

Situação atual

O software está em desenvolvimento, já sendo capaz de executar as transformações dos grafos pelo modelo DPO usando homomorfismos e isomorfismos como matches, e geração do espaço de estados. Para seu desenvolvimento, escolhemos a linguagem Haskell, devido a sua expressividade. O código do projeto está sendo revisado no momento, a fim de melhorar alguns aspectos não funcionais do sistema, e prepará-lo para executar reescritas de primeira e segunda ordem usando um mesmo *framework* de transformações. Ao mesmo tempo, estamos investigando como implementar CTL para podermos executar verificações formais nos modelos gerados.

Próximos passos

Para o futuro, planejamos implementar gramáticas de segunda ordem e verificações formais, tanto dinâmicas como estáticas.

References

[1] Hartmut Ehrig, Michael Pfender, and Hans Jürgen Schneider. Graph-grammars: An algebraic approach. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 167–180. IEEE, 1973.

[2] Frank Hermann, Hartmut Ehrig, and Gabriele Taentzer. A typed attributed graph grammar with inheritance for the abstract syntax of uml class and sequence diagrams. *Electronic Notes in Theoretical Computer Science*, 211:261–269, 2008.

[3] Rodrigo Machado. *Higher-Order Graph Rewriting Systems*. PhD thesis, UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2012.

[4] Arend Rensink. The groove simulator: A tool for state space generation. In *Applications of Graph Transformations with Industrial Relevance*, pages 479–485. Springer, 2004.

[5] Gabriele Taentzer. Agg: A graph transformation environment for modeling and validation of software. In *Applications of Graph Transformations with Industrial Relevance*, pages 446–453. Springer, 2004.

Agradecimentos

Gostaria de agradecer a FAPERGS e ao professor Rodrigo Machado pela oportunidade de participar deste projeto.

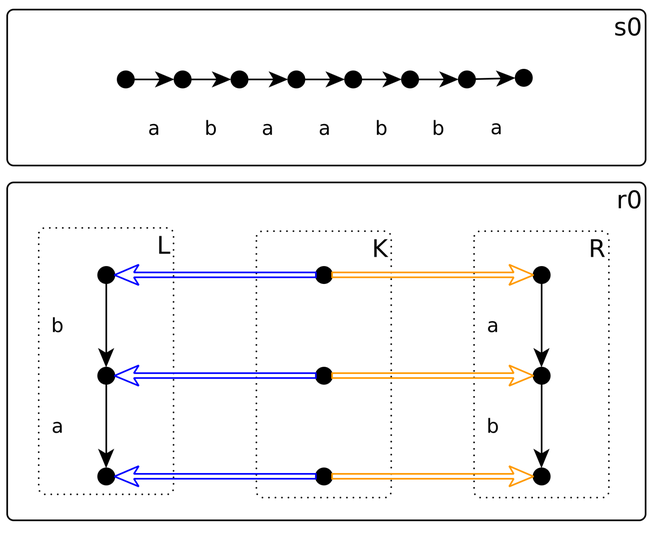

Figure 1: Exemplo de uma gramática que ordena as arestas lexicograficamente

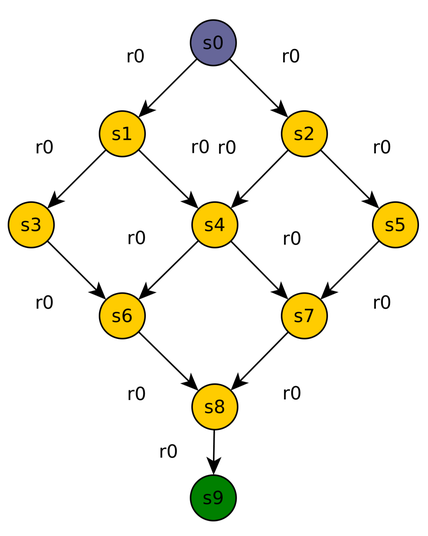

Figure 2: Espaço de estados gerado pela gramática na Figura 1