DESENVOLVIMENTO DE UMA TEORIA DE GRAMÁTICA DE GRAFOS PARA A PLATAFORMA RODIN

NATÁLIA TORALLES DARLEY[1]; LUCIANA FOSS[2]; SIMONE ANDRÉ DA COSTA CAVALHEIRO[3];

[1]*Universidade Federal de Pelotas – ntdarley@inf.ufpel.edu.br*
[2]*Universidade Federal de Pelotas – luci.foss@gmail.com*
[3]*Universidade Federal de Pelotas – simone.costa@gmail.com*

1. INTRODUÇÃO

As metodologias de especificação formal são utilizadas para descrever e analisar diversos tipos de sistemas computacionais (DAVIS, 2005). Com a especificação formal do sistema é possível desenvolver o sistema com uma base sólida, além de garantir confiabilidade do sistema através da verificação de suas propriedades (REIS, 2006). São diversas abordagens para a verificação formal de sistemas, uma destas é conhecida como prova de teoremas.

Na abordagem de prova de teoremas o sistema é modelado por um conjunto de definições lógico-matemáticas utilizando a inferência lógica para provar suas propriedades (CLARKE, 1996). Este é indicado para ser aplicado em sistemas com espaço de estados infinito e sistemas complexos. Isto é aplicado devido que a verificação não é feita pela exploração de dados, mas utilizando técnicas de indução estrutural que, por sua vez, permitem a prova sobre domínios infinitos. Existem provadores de teoremas que possibilitam ao desenvolvedor aplicar esta técnica, de forma que as as provas são feitas através da interação do usuário com o provador, um exemplo deste é a plataforma Rodin. Uma aplicação desta técnica pode ser encontrada em uma tradução de gramática de grafos para Event-B que foi proposta por DA COSTA (2010) em uma abordagem com estrutura relacional e linguagem lógica e com a indução matemática para verificar sistemas com espaço de estados infinito.

A Gramática de Grafos consiste em uma técnica da linguagem formal que é utilizada para analisar e descrever sistemas (EHRIG, 1997). Para compor uma gramática de grafos é feita a modelagem dos estados do sistema como grafos e a representação das possíveis mudanças de estados pelas regras que são descritas através do homomorfismo entre grafos. Após a tradução previamente proposta é possível fazer a verificação de um sistema especificado em gramática de grafos com a abordagem de prova de teoremas, através da plataforma Rodin. Para modelar uma gramática de grafos em Event-B deve-se especificar contexto e máquina. Em contexto é definida a parte estática do sistema, descrevendo as regras da gramática por meio de conjuntos, constantes e axiomas.E na máquina é definida a parte dinâmica do sistema através de eventos, detalhando o evento de inicialização como um estado inicial e as aplicações de regras como os restantes eventos do sistema.

Ao especificar uma gramática de grafos em Event-B um conjunto de obrigações de provas é gerado. Estas obrigações podem ser de bem-definição, de preservação de invariantes, entre outras. A prova das obrigações é normalmente realizada selecionando hipóteses, aplicando regras e táticas disponíveis nos provadores da ferramenta.

A proposta deste projeto é desenvolver uma teoria sobre gramática de grafos em Event-B para a plataforma Rodin que facilitará as especificações de sistemas que são modelados por gramática de grafos.

2. METODOLOGIA

Para desenvolver o projeto foi utilizado Event-B como linguagem para modelagem da gramática de grafos e Rodin como plataforma de entrada desta linguagem. Nesta ferramenta são feitas as verificações das propriedades do sistema através das provas de obrigações geradas após sua especificação.

Event-B é uma linguagem que permite fazer uma especificação formal de sistemas, sendo a linguagem de entrada da plataforma Rodin. Para a especificação é utilizada a notação lógico-matemática. Com isto, define-se o modelo por contexto e máquina, onde no contexto se definem os conjuntos, constantes e axiomas do sistema e em máquina especificam-se os eventos que podem ocorrer.

A plataforma Rodin foi proposta por Michael Butler e Stefan Hallerstede (BUTLER; HALLERSTEDE, 2007) para facilitar a modelagem formal em Event-B. A mesma foi desenvolvida para aplicar as técnicas de modelagem formal e de verificação dos sistemas de software. Ao se especificar um modelo, a ferramenta gera um conjunto de obrigações de prova, que devem garantir a consistência do sistema. A plataforma possui extensões que são disponibilizadas pela comunidade de pesquisadores e desenvolvedores, como por exemplo, o Theory Plugin. Esta é uma extensão que permitiu a criação de teorias para facilitar a especificação, validação e desenvolvimento da linguagem Event-B sem necessitar de modelo de contexto ou de máquina (BUTLER; MAAMRIA, 2013). Este incremento permite definir tipos de dados, operadores e axiomas, além de regras de inferência e regras de reescrita. Um novo tipo de dado deve definir um novo tipo construtor com uma função construtor e uma função destrutor para os elementos do novo tipo. Um operador define uma propriedade e pode gerar uma obrigação de prova, esta obrigação verifica e valida as condições da definição do operador. A reescrita de regra serve para facilitar o processo de prova, podendo ser automática ou interativa.

A partir da tradução de gramática de grafos para Event-B este trabalho propõe a especificação de uma teoria de gramática de grafos, de forma a definir tipos de dados e regras que facilitem a especificação de uma gramática de grafos em Event-B.

3. RESULTADOS E DISCUSSÃO

Na figura 1 tem a especificação dos tipos de dados grafo, morfismo de grafo e grafo tipado. Os parâmetros de tipo utilizados são: **vertSet**, conjunto de vértices e **edgeSet,** conjunto de arcos. Estão definidos três tipos de dados: o tipo de dado **Graph** (grafo), o tipo de dado **morphGraph** (morfismo de grafo) e o tipo de dado **typedGraph** (grafo tipado) .

O tipo de dado Graph possui o construtor **graph** e quatro destrutores que são: seus vértices (**vert**), seus arcos (**edge**), a função destino (**target**) e a função origem (**source**) que representam as relações dos arcos com vértices.

O tipo de dado morphGraph possui o construtor **morphGraph** e quatro destrutores que são: **sourceG** e **targetG** que são grafos e indicam os dois lados

do morfismo e **vertMap** e **edgeMap** que são, respectivamente, os mapeamentos dos vértices e arcos entre os grafos do morfismo.

O tipo de dado typedGraph possui como construtor o **typedGraph** e seus três destrutores são: **graphG** que é um grafo, **typedGraph** que é o grafo-tipo e **morphGraph** como morfismo.

```
THEORY
  graph
TYPE PARAMETERS
  vertSet
  edgeSet
DATATYPES
  Graph(vertSet, edgeSet)  ≙
   ▶ graph(vert:ℙ(vertSet), edge:ℙ(edgeSet), target:edgeSet ↔ vertSet, source:edgeSet ↔ vertSet)
  ■orphGraph(vertSet, edgeSet)  ≙
   ▶ ■orphgraph(sourceG:Graph(vertSet, edgeSet), targetG:Graph(vertSet, edgeSet), vertMap:vertSet↔vertSet , edgeMap:edgeSet↔edgeSet)
  typedGraph(vertSet, edgeSet)  ≙
   ▶ typegraph(graphG:Graph(vertSet, edgeSet), graphT:Graph(vertSet, edgeSet), typedVert:vertSet↔vertSet , typedEdge:edgeSet↔edgeSet)
```

Figura 1. Especificação da estrutura de grafo

```
OPERATORS
  •wdGraph  :  wdGraph(g : Graph(vertSet, edgeSet))  PREDICATE  PREFIX
  direct definition
   wdGraph(g : Graph(vertSet, edgeSet)) ≙ source(g) ∈ edgeSet ⟶ vertSet ∧ target(g) ∈ edgeSet ⟶ vertSet
  •finiteGraph  :  finiteGraph(g : Graph(edgeSet, vertSet))  PREDICATE  PREFIX
  well-definedness condition
   wdGraph(g)
  direct definition
   finiteGraph(g : Graph(edgeSet, vertSet)) ≙ fi nite(vert(g)) ∧ fi nite(edge(g))
  •welldefinedGraphMorph  :  welldefinedGraphMorph(gm : morphGraph(vertSet, edgeSet))  PREDICATE  PREFIX
  well-definedness condition
   wdGraph(sourceG(gm))
   wdGraph(targetG(gm))
   vertMap(gm) ∈ vert(sourceG(gm)) ⇸ vert(targetG(gm))
   edgeMap(gm) ∈ edge(sourceG(gm)) ⇸ edge(targetG(gm))
   ∀e·e∈ dom(edgeMap(gm)) ⇒ source(sourceG(gm)) (e) ∈ dom(vertMap(gm))
   ∀e·e∈ dom(edgeMap(gm)) ⇒ target(sourceG(gm)) (e) ∈ dom(vertMap(gm))
  direct definition
   welldefinedGraphMorph(gm : morphGraph(vertSet, edgeSet)) ≙ ∀e · e ∈ dom(edgeMap(gm)) ⇒ vertMap
   (gm) (source(sourceG(gm))(e)) = source(targetG(gm))(edgeMap(gm)(e)) ∧
    vertMap(gm) (target(sourceG(gm))(e)) = target(targetG(gm))(edgeMap(gm)(e))
  •totalMorph  :  totalMorph(gm : morphGraph(vertSet, edgeSet))  PREDICATE  PREFIX
  well-definedness condition
   welldefinedGraphMorph(gm)
  direct definition
   totalMorph(gm : morphGraph(vertSet, edgeSet)) ≙ dom(vertMap(gm))=vert(sourceG(gm)) ∧ dom(edgeMap(gm))
   =edge(targetG(gm))
  •typedGraph  :  typedGraph(gm : morphGraph(vertSet, edgeSet))  PREDICATE  PREFIX
  well-definedness condition
   welldefinedGraphMorph(gm)
  direct definition
   typedGraph(gm : morphGraph(vertSet, edgeSet)) ≙ dom(vertMap(gm))=vert(sourceG(gm)) ∧ dom(edgeMap(gm))
   =edge(targetG(gm))
```

Figura 2. Operadores definidos.

Na Figura 2 estão os operadores que foram definidos para os tipos de dados previamente especificados. São eles:

1. O operador **wdGraph** que estabelece um grafo bem definido, ou seja, as relações source e target devem ser funções totais, garantindo que cada arco deve ter um vértice origem e um vértice destino.
2. O operador **finiteGraph** é especificado quando o grafo é finito, ou seja, quando possui os conjuntos de arcos e vértices finitos.
3. O operador **welldefinedGraphMorph** define se o morfismo de grafo está bem definido. Para isso o grafo origem (*sourceG*) e destino (*targetG*) devem ser bem-definidos e o mapeamento de arcos deve preservar o mapeamento de seus vértices origem e destino.
4. O operador **totalMorph** estabelece que um morfismo é total quando o mapeamento de vértices e arcos são funções totais.
5. O operador **typedGraph** define que um grafo tipado deve ser definido por morfismo total entre grafos. Portanto, pode-se usar a operação de *morphGraph* para estabelecer a condição de grafo tipado. Neste operador,

os vértices do grafo são mapeados para o grafo-tipo e o mesmo acontece com os arcos que devem ser mapeados para os arcos do grafo-tipo.

Neste projeto ainda está em desenvolvimento a definição de tipo de dado, regra e gramática de grafos.

Tendo estes elementos definidos, pode-se descrever então uma gramática de grafos, que consiste em um tupla *GG = (T, G0, R)*, onde *T* define o grafo-tipo, *G0* indica o grafo inicial (um grafo tipo em *T)* e *R* é o conjunto de regras que essa gramática possui.

4. CONCLUSÕES

A fase de especificação formal ao desenvolver um software tem como objetivo descrever o comportamento do sistema, e com isto, prever ações que ele pode tomar através da realização de testes. Muitas vezes realizar testes é uma tarefa demorada e difícil, sendo assim custosa, e nem sempre se consegue abrangir todos os casos de testes. A metodologia formal prevê facilitar este processo e faz isso oferecendo fundamentações teóricas e técnicas formais, para que o produto que será implementado tenha a garantia de que não sofrerá equívocos durante sua execução. Ao fazer o uso das técnicas de especificação formal, além de garantir a seguraça do sistema, é possível simplificar e diminuir o custo em tempo do desenvolvimento.

A meta deste trabalho que é desenvolver uma teoria sobre gramática de grafos na plataforma Rodin, visa facilitar a especificação de modelos nesta ferramenta.

5. REFERÊNCIAS BIBLIOGRÁFICAS

ABRIAL, J.R. **Modeling in Event-B: system and software engineering**. Cambridge University Press, 2010.

BUTLER, M; MAAMRIA, I. Practical theory extension in Event-B. In: **Theories of Programming and Formal Methods**. Springer Berlin Heidelberg, 2013. p. 67-81.

BUTLER, M.; HALLERSTEDE, S. The Rodin formal modelling tool. In: ELECTRONICWORKSHOP IN COMPUTING, 2007. Anais. . . [S.l.: s.n.], 2007.

DAVIS, J. F. The affordable application of formal methods to software engineering. In: **ACM SIGAda Ada Letters**. ACM, p. 57-62, 2005.

DA COSTA, S. A. **Relational approach of graph grammars**. 2010. Tese de Doutorado. PhD thesis, Universidade Federal do Rio Grande do Sul.

EHRIG, H. Algebraic approaches to graph transformation: Part II: Single pushout approach and comparison with double pushout approach. In:**Handbook of Graph Grammars**. 1997. p. 247-312.

REIS, C.A.L. Métodos Formais para Especificação de Software. 2006.