UNIVERSIDADE FEDERAL DO PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TUPI

DOCUMENTO DE ESCOPO DE PROJETO

Filipe Marques Chaves de Arruda

Raisa Brito Costa

Rodrigo Barbosa Folha

{fmca , rbc5, rbf2}@cin.ufpe.br

RECIFE, ABRIL DE 2015.

INTRODUÇÃO

Máquinas de estados finitos (FSM) são abstrações que podem representar computações, as quais são implementadas através de transições entre os finitos estados, onde cada transição possui uma condição e ação associada. O comportamento de uma máquina de estados é observável em diversas áreas e tecnologias: desde o desenvolvimento de software para semáforos, elevadores e máquinas de venda [1], como também aplicados a pesquisas para geração de testes automáticos [2] e até para modelar o processo de mutação de genes [3].

Porém, FSMs são representações mais fracas que uma máquina de Turing. Então, é necessário estender o conceito de FSMs para: otimizar o espaço de estados utilizando módulos e hierarquia (evitando explosão de estados), permitir adoção de variáveis, introduzir ações de saída etc., constituindo-se numa EFSM (máquina de estados finitos estendida) [4]. Por definição, uma EFSM é uma tupla:

$$M = (Q, \Sigma 1, \Sigma 2, I, V, \Lambda)$$

Q -> Conjunto de estados finitos, simples ou compostos

$\Sigma 1$ -> Conjunto finito de eventos

$\Sigma 2$ -> Conjunto de ações

I -> Conjunto de estados iniciais $\subset Q$

V -> Conjunto de variáveis globais

Λ -> Conjunto de transições

OBJETIVO

O objetivo desse projeto é implementar um interpretador de uma linguagem que representa máquinas de estados estendidas (EFSMs), aplicando conceitos como herança e casamento de padrão.

Linguagem

O objetivo principal é representar cada propriedade de uma EFSM em código:

- ❏ Memória;
- ❏ Estados;
- ❏ Transições;
- ❏ Eventos;
- ❏ Guardas;
- ❏ Ações;

Além da adoção de conceitos e funcionalidades adicionais, tais como **herança** e **casamento de padrões**.

Interpretador

O interpretador será implementado em Java, acarretando dessa forma na execução na maioria das plataformas, inclusive em páginas da web. Além disso, será utilizada a ferramenta Xtext[1] para desenvolvimento da gramática, pois essa habilita a geração automática de uma IDE para suporte à linguagem.
Também será possivel também ativar a geração de arquivos ***.dot*** [5] para ilustrar qualquer EFSM em forma de grafo.

Exemplo de código

Cada evento é descrito por uma **{EVENT}** tag seguida por um ou mais casamentos de padrão da transição corresponde **[estados_origem_padrao] -> novoEstado**, uma **guarda** e uma **action** a ser executada.

[1] https://eclipse.org/Xtext/

```
machine Stack

   memory
      list, peek

   states
      empty, notempty

   events
      {START}
         [*] -> empty
            initialize

      {PUSH} [x]
         [*empty] -> notempty    | pushAllowed
            addElement x

      {POP}
         [notempty] -> empty     | !hasMore1Element
            deleteElement
         [notempty] -> notempty  | hasMore1Element
            deleteElement
   guards
      hasMore1Element = list.size > 1
      pushAllowed = true

   actions
      initialize
         list = start List

      addElement [x]
         list {add} x
         peek = list.last

      deleteElement
         list {delete} list.size-1
         peek = list.last
```

Também é possível estender um outra máquina:

```
machine SizeLimitedStack extends Stack

   memory
      limit

   states
      full

   events
      {PUSH} [x]
         [*empty] -> full | !pushAllowed
            addElement x

      {POP}
         [full] -> empty | !hasMore1Element
            deleteElement
         [full] -> notempty
            deleteElement

   guards
      pushAllowed = list.size < limit

   actions
      initialize [x]
         Stack.initialize
         limit = x
```

Referências

1. Kaveri, P., G. R. K. Prasad, and Fazal Noorbasha. "Router Design Using Cadence Encounter."
2. Kwang Ting Cheng and A. S. Krishnakumar. 1993. Automatic functional test generation using the extended finite state machine model. In *Proceedings of the 30th international Design Automation Conference* (DAC '93). ACM, New York, NY, USA, 86-91.
3. Gao, Rui, Wensong Hu, and Tzyh-Jong Tarn. "The Application of Finite State Machine in Modeling and Control of Gene Mutation Process." *NanoBioscience, IEEE Transactions on* 12.4 (2013): 265-274.
4. Alagar, V. S., and K. Periyasamy. "Extended finite state machine."*Specification of Software Systems*. Springer London, 2011. 105-128.
5. Koutsofios, Eleftherios, and Stephen North. *Drawing graphs with dot*. Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ, 1991.