TDS-Repl

Mateus Azeredo Torres

Updated: 24 August 2020

1 A linguagem

1.1 Características da linguagem

A linguagem permitirá por meio de comandos simples, funções, e diretivas próprias, representar *TDS*s[3] para portas quaisquer de um circuito *Reo*[1], e então expressar a relação de dependência entre elas, manipular e visualizar seus dados antes de converter para o *nuXmv*[4]. A linguagem considera em sua gramática os comandos mais simples e comuns a outras linguagens de programação e a definição de procedimentos e funções. A linguagem conta com "diretivas temporais" e "diretivas TDS", auxiliando a definição de intervalos de observação e comportamentos com noções de dependência temporal.

As diretivas temporais: *I_TIME*, *C_TIME* e *F_TIME*, respectivamente, representam o instânte incial, atual e final de observação, e iniciam com valores *default* 0, 0 e 3, porém, as diretivas podem ser usadas e alteradas em diversos contextos. Vale ressaltar, que *C_TIME* é incrementada dentro do intervalo de observação a medida que o modelo é executado. Além disso, quando *C_TIME* é alterada ocorre uma mudança do "contexto temporal" da execução do modelo, isto é qualquer definição após essa alteração só será considerada quando o passo da execução, isto é um "instante de tempo" se igualar ao valor alterado na diretiva.

As diretivas *TDS*, associam o rótulo de uma porta qualquer aos comportamentos especificados que ditam presença de dados nessa. Os comportamentos das *TDS* podem ser gerados de diversas formas, seja utilizando procedimentos comuns a linguagens de programação, ou por uma função "matemática", ou uma lista de dados para cada instante de tempo, ou ainda por outra *TDS* já especificada. Onde no último caso, deve-se especificar o input para a nova tds, feito pela diretiva *linked*. Além disso, pode-se especificar "atraso" para as portas, por exemplo, para modelos assincronos como uma *FIFO* representada em *Reo*.

O interpretador da linguagem irá converter os comandos utilizados no escopo global para váriaveis e *constraints* no módulo *Main* do *nuXmv*. De maneira similar, os comandos associados a diretivas *TDS* são convertidos para um outro módulo *nuXmv* chamado *portsModule*, que representa o fluxo de dados e dependência entre as portas de um conector *Reo*. Onde esse módulo *portsModule* pode ser usado por outros módulos criados no *nuXmv*, assim fornecendo fluxo de dados para portas de um conector de um modelo qualquer utilizado pelo usuário. Atualmente o *portsModule* será utlizado por um módulo que representa um *Constraint Automaton*[3] criado no modelo *nuXmv* pelo trabalho de Daniel Toledo[2]

1.2 BNF e Caracteristicas

A BNF da linguagem considera a definição de comandos simples para expressar a lógica que dita o input e output de dados, contemplando os comandos mais simples e comuns a outras

linguagens de programação, e a definição de procedimentos e funções. O *design* dessa linguagem intermediária parte do principio que para representar o "fluxo" de uma *TDS* o usuário não é orbrigado saber criar exatamente um "programa", embora seja possível fazer isso com os comandos básicos oferecidos, assim a linguagem tem um foco em utilizar suas diretivas e representações simplificadas para *TDS* e funções.

⟨*prog*⟩ ::= ⟨*functiondefs*⟩ ⟨*cmds*⟩
  | ⟨*cmds*⟩

⟨*functiondefs*⟩ ::= ⟨*functiondefs*⟩ ⟨*functiondef*⟩
  | ⟨*functiondef*⟩

⟨*functiondef*⟩ ::= '`function`' **id** '`(`'⟨*params*⟩'`)`''`{`' ⟨*functionbody*⟩ '`}`'
  | '`function`' **id** '`(`''`)`''`{`' ⟨*functionbody*⟩ '`}`'

⟨*functionbody*⟩ ::= ⟨*cmds*⟩ ⟨*optionalreturn*⟩

⟨*optionalreturn*⟩ ::= '`return`' ⟨*expr*⟩
  | ⟨*empty*⟩

⟨*params*⟩ ::= id
  | ⟨*params*⟩ '`,`' id

⟨*cmds*⟩ ::= ⟨*cmds*⟩ ⟨*cmd*⟩
  | ⟨*cmd*⟩

⟨*cmd*⟩ ::= if'`(`'⟨*expr*⟩'`)`' '`{`'⟨*cmds*⟩'`}`'⟨*matchornot*⟩
  | ⟨*otherstmt*⟩

⟨*matchornot*⟩ := else'`{`'⟨*cmds*⟩'`}`'
  | ⟨*empty*⟩

⟨*timedirective*⟩ ::= I_TIME
  | C_TIME
  | F_TIME

⟨*otherstmt*⟩ ::= '`for`' **id** '=' ⟨*expr*⟩ '`to`' ⟨*expr*⟩ '`do`' '`:`' ⟨*cmds*⟩
  | ⟨*functioncall*⟩
  | **id** ⟨*extraaccesses*⟩ '=' ⟨*expr*⟩
  | ⟨*timedirective*⟩ '=' ⟨*expr*⟩

⟨*extraaccesses*⟩ ::= [' ⟨*expr*⟩ '`]`' ⟨*variableprop*⟩
  | ⟨*variableprop*⟩

⟨*variableprop*⟩ ::= '`.`' ⟨*tdsprop*⟩
  | ⟨*empty*⟩

⟨*tdsprop*⟩ ::= ⟨*functioncall*⟩
  | '`portname`'
  | '`data-time`'
  | '`linked`'
  | '`delayed`'

⟨*functioncall*⟩ ::= id‘(’⟨*params*⟩‘)’
| id ‘(’‘)’

⟨*aritplusminus*⟩ ::= ‘−’
| ‘+’

⟨*expr*⟩ ::= ‘−’ ⟨*expr*⟩
| ⟨*expr*⟩ ⟨*aritplusminus*⟩ ⟨*multiexp*⟩
| ⟨*multiexp*⟩

⟨*multdividemodule*⟩ ::= ‘*’
| ‘/’
| ‘%’

⟨*multiexp*⟩ ::= ⟨*multiexp*⟩ ⟨*multdividemodule*⟩ ⟨*ineqexp*⟩
| ⟨*ineqexp*⟩

⟨*compare*⟩ ::= ‘>’
| ‘>=’
| ‘<’
| ‘<=’
| ‘==’
| ‘!=’

⟨*ineqexp*⟩ ::= ⟨*ineqexp*⟩ ⟨*compare*⟩ ⟨*logical*⟩
| ⟨*logical*⟩

⟨*logicop*⟩ ::= ‘and’
| ‘or’

⟨*logical*⟩ ::= ‘not’ ⟨*data*⟩
| ⟨*logical*⟩ ⟨*logicop*⟩ ⟨*data*⟩
| ⟨*data*⟩

⟨*data*⟩ ::= **id**
| RAWNUMBERDATA
| BOOLEAN
| NULL
| I_TIME
| C_TIME
| F_TIME
| LABEL
| **id** ⟨*extraaccesses*⟩
| ⟨*variabledata*⟩

⟨*variabledata*⟩ ::= ‘{’ ‘portname’ ‘:’ label ‘,’ ‘data-time’ ‘:’ ‘{’ ⟨*dataflow*⟩‘}’ ⟨*extras*⟩‘}’
| ⟨*functioncall*⟩

⟨*extras*⟩ ::= ‘,’ ‘linked’ ‘:’ ‘{’⟨*params*⟩‘}’
| ‘delayed’ ‘:’ BOOLEAN
| ⟨*empty*⟩

⟨*dataflow*⟩ ::= ⟨*domain*⟩

⟨*domain*⟩ ::= ⟨*timelist*⟩
| ‘`function-domain`’ ‘`:`’ ⟨*functionCall*⟩

⟨*timecomponent*⟩ ::= RAWNUMBERDATA ‘`:`’ ⟨*expr*⟩

⟨*timelist*⟩ ::= ⟨*timecomponent*⟩
| ⟨*timelist*⟩ ‘`,`’ ⟨*timecomponent*⟩

Referências

[1] Farhad Arbab. *“Reo: a channel-based coordination model for component composition”*. Math. Struct. in Comp. Science, vol.14, pp.329–36. Cambridge University, 2004.

[2] Daniel Arena. *“Um compilador de circuitos Reo para modelos nuXmv”*. Trabalho de conclusão de curso - Instituto de Computação, Universidade Federal Fluminense, 2019.

[3] Arbabd F Baier C, Sirjanib M and J Ruttend. *“Modeling component connectors in Reo by constraint automata”*. Science of Comp. Programming, vol.61, pp.75–113. Elsevier, 2006.

[4] et al. Cavada R. *The nuXmv Symbolic Model Checker. In:Biere A., Bloem R.(eds)*. CAV 2014. Lecture Notes in Computer Science, vol 8559. Springer, Cham, 2014.