Definição de uma Linguagem para a Programação do Comportamento de Robôs Dentro do Contexto da RoboCup

Jéssica A. Bonini[1], Giovani R. Librelotto[1]

[1]Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM)
Caixa Postal 5082 – 97195-000 – Santa Maria – RS – Brasil

{jbonini, librelotto}@inf.ufsm.br

Abstract. *The RoboCup Soccer is one of the leagues of the World Robots Cup. It tests the autonomous humanoid robots play football ability. In the year of 2015, Taura Bots, robot soccer team from the Federal University of Santa Maria, first participated in the competition. Thus arose the need to create a language for the robot behavior programming that offers users a high level of abstraction, portability and a simple and intuitive syntax. The project objective was achieved, considering TauraLang, language built in this work, can now be used in simple behaviors programming and some have run in the simulator.*

Resumo. *A RoboCup Soccer é uma das ligas da Copa Mundial de Robôs. Nela testa-se a capacidade de robôs humanóides autônomos jogarem futebol. No ano de 2015 a Taura Bots, equipe de futebol de robôs da Universidade Federal de Santa Maria, participou pela primeira vez da competição. Assim, surgiu a necessidade de criação de uma linguagem para a programação do comportamento dos robôs que ofereça aos usuários um alto nível de abstração, portabilidade e uma sintaxe simples e intuitiva. O objetivo do projeto foi alcançado, levando em conta que a TauraLang, linguagem construída neste trabalho, já pode ser usada na programação de comportamentos simples e alguns já executam no simulador.*

1. Introdução

Cada vez mais as pesquisas mostram que é possível desenvolver robôs móveis autônomos comandados por uma inteligência programada. Nessa linha, a *RoboCup*, copa do mundo de robôs, busca desenvolver pesquisas nas áreas de inteligência artificial e robótica, com objetivo de tornar possível um time de robôs vencer um time de humanos campeão da Copa do Mundo FIFA [RoboCup 2012].

A UFSM participou de sua primeira RoboCup em 2015, levando para a China a equipe Taura Bots em parceria com uma equipe alemã. Para a programação do comportamento do robô, a equipe contou com um simulador 2D próprio [Montenegro 2015]. Esse simulador permite a realização de testes sem a necessidade do robô físico e, além disso, que o código programado no simulador seja portado diretamente para o robô real. Atualmente, as duas linguagens usadas pela equipe para a programação do robô e das estratégias no simulador são, respectivamente, C++ e *Python*. Assim, a programação em uma linguagem de propósito geral como C++ exige que cada programador conheça todo o sistema de software e hardware por trás das ações do robô.

Uma das dificuldades encontradas na programação do robô é a sintaxe das linguagens utilizadas. O programador muitas vezes sabe exatamente o que deve ser

feito, porém não sabe como programar essas ações, pois não tem muito conhecimento de como os conceitos são representados. Dessa forma, perde-se tempo com erros que não ocorreriam se o programador pudesse usar uma linguagem mais intuitiva e de sintaxe simples. Outro problema é que por vezes, o usuário que está programando o comportamento, não tem muito conhecimento das regras do futebol e acaba não sabendo quais ações devem ser realizadas pelo robô diante certa situação. Esse problema seria minimizado se pessoas que entendem como funciona um jogo de futebol, mas não sabem muito sobre programação e as sintaxes das linguagens, pudessem programar o comportamento. Uma linguagem de fácil compreensão e programação, junto a um tradutor para traduzi-la para uma linguagem com nível mais baixo e compreendida pelo sistema atual, diminuiria o tempo e otimizaria a programação das ações e táticas dos agentes.

O trabalho em questão tem como proposta a definição e criação de uma linguagem que visa proporcionar uma maneira mais intuitiva e simples de programar as ações e estratégias da inteligência artificial utilizada pelo robô/simulador. Assim, substituindo a forma de programação convencional que vem sendo usada e beneficiando o programador com a abstração da parte operacional do sistema, permitindo que este preocupe-se somente com as definições das estratégias dos agentes durante o jogo.

A primeira etapa para a execução do trabalho foi a definição de requisitos de entrada e saída, definindo todos os objetos que o robô poderia reconhecer e todas as ações que ele poderia executar. Após a definição destes requisitos, definiu-se a sintaxe da linguagem através da gramática e suas regras de produção. Nessa fase, notou-se que a programação de robôs que jogam futebol é similar a programação de sistemas síncronos, o que levou ao uso da gramática da linguagem RS, linguagem voltada para a especificação e implementação de núcleos de sistemas reativos síncronos [Toscani 1993], como base da TauraLang. Após essas etapas, foi construído, com ajuda do *ANTLR*, um parser e um tradutor para *Python*.

Este trabalho está estruturado da forma apresentada a seguir: a Seção 2 apresenta a revisão bibliográfica, dando destaque aos tópicos *RoboCup*, Simulador 2D da equipe Taura Bots e Linguagem RS, necessários para a compreensão do trabalho. A Seção 3 expõe a metodologia, apresentando a definição da gramática, implementação do parser e tradutor. A Seção 4 apresenta um caso de estudo para validação da linguagem. Por fim, a Seção 5 traz trabalhos relacionados.

2. Fundamentação Teórica

Esta seção destina-se à definição de conceitos teóricos necessários para a compreensão deste trabalho.

2.1. RoboCup

A *RoboCup* é uma competição a nível mundial onde testa-se a capacidade de robôs desenvolvidos com objetivo de desempenhar funções normalmente realizadas por seres humanos. A liga principal subdivide-se em ligas menores como a *Robocup@Home*, onde robôs desenvolvem funções do dia a dia; *RobocupRescue* que visa a competição de robôs em ambientes de catástrofe e a *RoboCup Soccer* que traz para a robótica o contexto de robôs que jogam futebol. Essa competição tem como objetivo principal o desenvolvimento e realização de pesquisas nas áreas de robótica e inteligência artificial por meio da sugestão de plataformas estimulantes fundamentadas em problemas do

mundo real [RoboCup 2012].

A edição da *RoboCup Soccer* de 2015 contou com a participação da Taura Bots, equipe formada por estudantes da UFSM, em parceria com a equipe alemã WF-Wolves da Universidade de Ostfalia. Para o treinar as estratégias da equipe, foi desenvolvido um simulador 2D.

2.2. Simulador 2D da equipe Taura Bots

A Taura Bots é dividida em subequipes responsáveis por partes diferentes do humanoide como a subequipe do comportamento responsável por programar estratégias que devem ser realizadas pelo agente. Dessa forma, como a maioria das subequipes necessitava o uso constante do robô para testar o que havia sido desenvolvido notou-se que esse era um recurso escasso e que era necessário a construção de um simulador capaz de prover o mundo, contexto onde o robô está inserido, e o próprio robô.

O simulador 2D foi desenvolvido com objetivo de permitir a programação do comportamento do robô sem a necessidade de tê-lo fisicamente. Além disso, foi projetado para suprir a falta de um simulador que oferecesse o nível desejado de abstração e permitisse que o código desenvolvido fosse diretamente portado para o robô. No simulador existe uma parte responsável por simular o mundo, ou seja, todo o contexto em que o robô está inserido (campo, bola, poste, etc) e a parte que simula o robô, responsável por reproduzir as ações que devem ser tomadas [Montenegro 2015].

A abstração oferecida pelo simulador permite que o programador não precise se preocupar com a forma como os dados do ambiente são coletados e ainda como os movimentos do agente são programados, levando o programador a preocupar-se especificamente com o desenvolvimento das estratégias do comportamento. A comunicação de dados foi realizada usando o padrão *JSON*, fornecido junto ao *Python*, que oferece simplicidade e robustez, permitindo que ao mesmo tempo seja criado um alto nível de abstração para a simulação das estratégias e a manutenção de uma camada de compatibilidade com o sistema real.

O robô então recebe estímulos do mundo, objetos percebidos pela visão, e reage de acordo com eles, realiza uma ação como caminhar. Dessa forma, buscou-se uma linguagem já existente que pudesse auxiliar na criação da linguagem proposta.

2.3. Linguagem RS

A linguagem RS é voltada para a especificação e implementação de núcleos de sistemas reativos síncronos [Toscani 1993]. Esses são sistemas dirigidos por estímulos provenientes de um ambiente externo. A hipótese de sincronismo considera sistemas ideais que reagem de forma imediata a cada estímulo recebido, alterando o estado interno e emitindo sinais [Toscani and Monteiro 1994].

A RS possui comandos simples baseados em regras de "condição → ação", o que facilita o raciocínio e concentração na construção do sistema. Um programa escrito nessa linguagem é uma especificação quase direta das alterações internas e dos sinais emitidos para cada possível estímulo recebido do ambiente [Toscani and Monteiro 1994]. A estrutura básica da linguagem RS conta com um conjunto de entradas, que define o que pode ser recebido do ambiente; um conjunto de saídas, que apresenta as possíveis respostas aos estímulos; sinais internos, usados para a sincronização e comunicação interna de processos; e um conjunto de regras, que determinam as

respostas do sistema aos estímulos sofridos [Librelotto et al. 2007].

Seguindo a linha do funcionamento de sistemas síncronos, concluiu-se que os robôs funcionam basicamente da mesma forma que a linguagem RS. Assim optou-se por definir uma nova linguagem que tem como base a gramática da RS. Para a definição da nova linguagem, fez-se necessário o uso de um gerador de parser.

3. Metodologia

Esta seção apresenta as etapas para a construção da linguagem TauraLang.

3.1. Contextualização sobre a Linguagem TauraLang

A programação do robô dentro da equipe Taura Bots, está dividida em três subequipes: visão, comportamento e caminhada, como mostra a Figura 1.

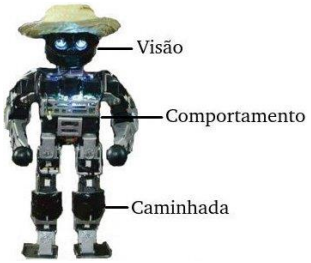

Figura 1. Subdivisão da equipe Taura Bots.A visão é a parte responsável por perceber o ambiente e alimentar o comportamento com informações sobre o mundo, essas informações podem ser dos tipos: bola, poste, robô, linhas (linha reta, no formato de X, L ou T) e objeto desconhecido. Para a percepção do objetos, a visão conta com uma identificação precisa de cores.

A visão é a parte responsável por perceber o ambiente e alimentar o comportamento com informações sobre o mundo, essas informações podem ser dos tipos: bola, poste, robô, linhas (linha reta, no formato de X, L ou T) e objeto desconhecido. Para a percepção do objetos, a visão conta com uma identificação precisa de cores.

O comportamento é considerado a mente do robô e fica responsável por processar as informações recebidas da visão e montar táticas de jogo. Ele não preocupa-se com a forma que essas informações foram coletadas, para seu funcionamento é necessário apenas saber o tipo e a posição dos objetos detectados. Outra informação importante é a posição da cabeça, para que o robô possa procurar objetos no ambiente ou movimentar-se de maneira correta. Por fim, depois de processar as informações, o comportamento repassa as ações a serem executadas pela parte da caminhada.

A caminhada fica com a responsabilidade de efetuar o que foi processado pela mente, podendo executar os seguintes comandos: caminhar, chutar ou defender (para robô goleiro). Os parâmetros recebidos são processados e os motores das juntas são setados de acordo com o movimento recebido do comportamento.

A ideia da criação da linguagem surgiu a partir da necessidade de tornar o desenvolvimento de técnicas de comportamento mais simples e intuitivo, não exigindo conhecimento de todo o sistema de hardware e software por trás do robô. Somando-se a isso, a possibilidade de portabilidade do código, ou seja, o código final pode ser gerado em *Python*, C++ ou qualquer outra linguagem desejada e ainda poderá ser, a longo

prazo, usado no robô real. Para a definição da sintaxe da linguagem é necessário coletar e definir os requisitos básicos.

3.2. Definição de Requisitos da TauraLang

Para a construção da TauraLang, foi necessária a definição dos requisitos fundamentais para a programação do comportamento no simulador 2D da equipe Taura Bots: entradas, estados e saídas. Esses requisitos foram coletados em uma fase inicial da equipe, onde o robô percebia objetos base, como bola e poste, e executava ações simples, como caminhar e chutar.

Requisitos de Entrada

As entradas correspondem aos objetos recebidos pelo comportamento, no simulador esses são enviados pela parte da visão por mensagens no formato *JSON*. A Figura 2 apresenta o código da mensagem de entrada no formato *JSON*:

```
{
    "head_angle": 0.0, //angulo da cabeça
    "object_list": ["kind": "ball", // tipo do objeto
        "position": [239.90319, -0.68769]]
}
```

Figura 2. Mensagem de entrada no formato *JSON*.

Juntamente com o tipo, a visão envia a posição em coordenadas polares de cada objeto, r e *theta*. Levando em conta as informações e requisitos coletados, os sinais de entrada foram definidos da forma apresentada na Figura 3.

```
in: [ball(r, theta), robot(r, theta), pole(r, theta), T_line(r, theta),
X_line(r, theta), L_line(r,theta),line(r, theta), unknown(r,theta)]
```

Figura 3. Sinais de entrada na TauraLang.

Os objetos identificados pela visão e enviados ao comportamento podem ser de oito tipos: *ball*, *pole*, *robot*, *line*, *X_line*, *T_line*, *L_line* e *unknown*.

- Tipo *ball*: representa a bola e é identificado pela cor vermelha. Segundo as regras da competição, pode existir apenas um objeto desses em jogo e o mesmo pode estar posicionado dentro das linhas do campo ou na lateral dele, essa última situação representa a bola fora de jogo, ou seja, bola a ser reposta.
- Tipo *pole*: representa o poste e é identificado pela cor branca. Podem existir no máximo quatro objetos desse tipo. Dependendo da posição e do tamanho do campo de visão do robô, a quantidade de postes recebidos pelo comportamento pode variar de zero ao número máximo. A linguagem considera que a visão envia um poste como um objeto do tipo *poleI*, onde I é um índice que varia de 1 a 4.
- Tipo *robot*: atualmente a visão do robô real não faz a identificação de outros robôs, porém o simulador já representa esse tipo de objeto. Apesar de representar um objeto robô, o simulador não é capaz de diferenciar um robô oponente de um robô parceiro (do mesmo time). Na linguagem essa representação é feita levando em conta que a visão passe um robô oponente como um objeto do tipo *robotI_o*, onde I é um índice de 1 a 3. E um robô parceiro como um objeto do tipo *robotI_p*, onde I representa o mesmo índice citado anteriormente. Podem existir no máximo três robôs oponentes e três

robôs parceiros.

- Tipo *line*: representa linhas do campo e é identificado pela cor branca e diferenciado do objeto poste pela variação entre a cor branca e a cor verde do gramado.
- Tipo *X_line*: variação do objeto *line*, encontrado nas intersecções das linhas do meio de campo.
- Tipo *T_line*: variação do objeto *line*, encontrado nas intersecções das linhas laterais com a linha do meio de campo e das linhas da grande área com a linha de fundo.
- Tipo *L_line*: variação do objeto *line*, encontrado nas intersecções das linhas laterais com a linha de fundo e nas intersecções das linhas (cantos) da grande área.
- Tipo *unknown*: objeto percebido pela visão, mas não identificado ou não especificado em nenhum dos tipos anteriores.

Requisitos de Saída

Definiram-se também os requisitos de saída, esses constituem as ações a serem executadas pela caminhada. O envio das ações também é feito no formato *JSON*. Na Figura 4 é apresentado o código da mensagem de saída no formato *JSON*.

```
{    /* movement_vector:[r, theta, phi] */
     "movement_vector": [ 5, -0.97585, -0.97585]
     "index": 0,
     "head_angle": -0.9758561608856633 // angulo da cabeca
}
```

Figura 4. Mensagem de saída no formato *JSON*.

No momento da coleta de requisitos a equipe estava em um nível inicial, dessa forma foram definidos três movimentos básicos: *walk*, *kick* e *defend* (para robô goleiro). Assim, as saídas do comportamento ficaram da forma apresentada na Figura 5.

```
out: [walk(r, theta, phi), kick(r, theta),defend(x)]
```

Figura 5. Sinais de saída na TauraLang.

- Ação *walk*: comando enviado quando deseja-se que o robô movimente-se no campo. Um *movement vector* é setado com valores que indicam qual a nova posição do robô. O movimento é representado pelas componentes polares r e theta e o ângulo de rotação do robô sobre o próprio eixo é representado pela componente *phi*.
- Ação *kick*: usada para que o robô chute a bola. As componentes polares r e theta indicam a posição para onde a bola deve ser chutada.
- Ação *defend*: essa ação é executada somente por robôs goleiros e é usada para informar ao robô que ele deve defender a bola. O parâmetro x pode assumir três valores: -1 para indicar defesa à esquerda, 0 para indicar não movimente-se e 1 para defesa à direita.

Os requisitos definidos consistem na estrutura básica da linguagem, ou seja, são as condições obrigatórias para a linguagem atingir o objetivo.

3.3. Estrutura da linguagem

A linguagem será composta por uma parte inicial com informações básicas para seu funcionamento e por um bloco com regras de comportamento. Na primeira parte estão localizadas as seguintes informações:

- Sinais de Entrada: correspondem aos objetos que a visão envia para o comportamento. Esses sinais são estímulos do ambiente e combinados com os estados serão responsáveis por disparar as regras do comportamento. São representados pelo conjunto *in:[objects]*. O conjunto de sinais de entrada é estático, ou seja, em todos os casos a visão pode perceber apenas os oito tipos apresentados anteriormente.
- Sinais de Saída: são formados pelo conjunto de ações que o comportamento repassa para a caminhada executar. Essas ações serão resultado do processamento das regras. São representados pelo conjunto *out: [actions]*. O conjunto de sinais de saída também é estático, somente as três ações apresentadas anteriormente poderão ser repassadas para a saída.
- Variáveis: contém as variáveis definidas pelo usuário. Somente as definidas nesse conjunto poderão ser usada nas regras. São representadas por *var: [variables]*.
- Estados: esse conjunto representa os estados internos do robô. Em uma disputa, o robô pode estar no estado de ataque (*attack*), no estado de defesa (*defense*), ser um goleiro (*goalkeeper*) ou ainda estar em um estado definido pelo usuário. Representados pelo conjunto *states: [state]*.
- Estado inicial: corresponde ao estado em que o robô começará o jogo. Deve ser um dos estados declarados em *states*. Representado pela instrução *inittialy: [state]*.

A segunda parte é formada por um bloco de regras do tipo Condição → Ação e informação a caminhada o que o robô deve fazer. A condição é formada pelo conjunto de sinais de entrada, recebidos do ambiente, que ativam as regras e disparam as ações. A ação é formada por um conjunto de instruções que resultam em um comando que deve ser executado.

Os estímulos recebidos, combinados com o estado interno atual, formarão a parte condicional da regra, por exemplo, o robô recebe o objeto *ball* e está no estado de *attack*, a partir dessa condição o robô deve se comportar de alguma maneira. Nesse momento entra a segunda parte da regra, a ação, considerando o exemplo citado acima, supomos que o objeto *ball* está próximo do robô, a possível ação a ser tomada é chutar a bola na direção especificada e continuar no estado de *attack*.

Cada estado interno terá um conjunto de regras próprio, isso significa que após a execução de qualquer uma das regras do conjunto o robô permanecerá no estado portador da regra. Por exemplo, se uma regra do conjunto de regras do estado *attack* é acionada, após sua execução o robô continuará no estado de *attack*. Para mudanças de estados, existe um conjunto de regras de transição, ou seja, após uma regra desse conjunto ser executada o robô muda seu estado. Por exemplo, se uma regra de transição é executada quando o robô está no estado *attack*, após o término da execução o robô estará no estado *defense*. A permanência ou a variação de estado é definida pela instrução *up(state)*. As ações que a caminhada deve executar são representas pela instrução *emit(action)*. O robô só pode estar em um estado por vez, assim cada regra

contém apenas uma instrução up. A Figura 6 exemplifica os conjuntos de regras.

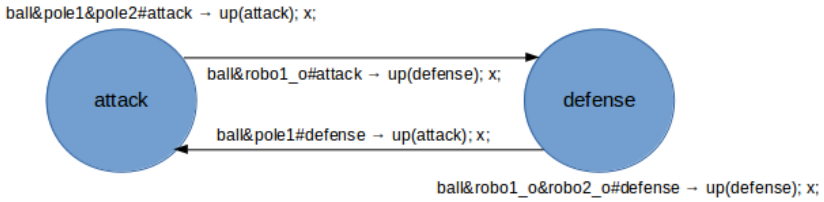

Figura 6. Autômato do conjunto de regras da linguagem.

Além das instruções up e emit a parte de ação da regra é formada por:

- CondIf e CondIfElse: responsáveis pela representação das condições.
- Atr: responsáveis pelas atribuições (operações lógicas e aritméticas).

3.4. Implementação: Gramática da Linguagem TauraLang e Parser

Para a construção da sintaxe da linguagem foram seguidas as três fases iniciais de um compilador: Análise Léxica, Análise Sintática e Análise Semântica.

Na análise léxica foram definidos os *tokens* da linguagem. As classes de *tokens* identificadas são: números, palavras reservadas e identificadores. Os números da linguagem podem ser inteiros ou *floats* e são representados pelos *tokens* NUM e DIG, respectivamente. As palavras reservadas constituem o conjunto de palavras que não podem ser usadas como identificadores, ou seja, se a palavra *if* faz parte do conjunto de palavras reservadas, não pode definir-se uma variável com esse nome. Essa lista de palavras é de uso exclusivo da gramática da linguagem. Os *tokens* palavras reservadas da linguagem TauraLang estão representas na Tabela 1.

Tabela 1. Palavras reservadas da TauraLang.

module	in	out	var
states	initially	ball	pole
robot	X_line	L_line	T_line
line	r	theta	phi
walk	defend	kick	float
attack	defense	if	else
up	emit	true	false
boolean	:	#	&
{	}	[	]
+	*	=	/
-	->	<	==
>	<=	>=	!=
:=	,	(	)

Os identificadores são todos os nomes de funções e variáveis que podem ser definidas pelo programador. Por exemplo, int x, x é um identificador. Os identificadores são representados *pelo token* ID. Os *tokens* foram agrupados em coleções com sentido coletivo na fase de análise sintática, assim a gramática da linguagem foi definida da forma apresentada na Figura 7.

Na fase de análise semântica, algumas verificações foram realizadas. São elas: verificação de tipo, verificação de variável duplicada e verificação de argumentos. A verificação de tipo é responsável por garantir que a variável recebe os valores correspondentes ao tipo definido. A verificação de variável duplicada impede que mais de um identificador seja declarado com o mesmo nome, por exemplo, duas variáveis *int cont*. Por último, na verificação de argumentos foi constato se cada argumento estava no lugar correto, por exemplo, em *initially: [arg]*, o arg só pode ser uma palavra já definida em *state: [words]*.

Expr → Name Body

Name → module ID:

Body → {Input Output Var Stts Init Rules*}

Input → in: [ParametersIn pInput

ParametersIn → ball | POLE |

(ROBO_O | ROBOT_P | ROBOT) | Xline | Tline | Lline | line | unknown

pInput → (ID, ID) ListIn

ListIn → , ParametersIn pInput|];

Output → out: [parametersOut];

parametersOut → walk' (ID, ID, ID), defend(ID), kick(ID, ID)

Var → var: [ ]; | var: [Decl listVar

ListVar → , Decl listVar |];

Decl → float ID | boolean ID

Stts → states: [parametersStts];

parametersStts → attack,

defense, goalkeeper, Unknown

Unknown → ID

Init → initially: [ParametersInit]

ParametersInit: attack | defense |

goalkeeper | ID

Rules → Condition Action

Condition → ParametersIn(ID, ID)

And#ParametersInit →

And → &ParametersIn(ID, ID) And | vazio

Action → Emit Term | Atr Term | condIf Term | condIfElse Term | Up Term

CondIf → if(OprLflt | OprLbool)

{ Term }

CondIfElse → CondIf CondElse

CondElse → else {Term}

Term → Emit Term | Atr Term |

CondIf Term | CondIfElse Term | vazio

Emit → emit(walk(Parameter,

Parameter, Parameter));

| emit(defend(NUM | DIG));

| emit(kick(Parameter, Parameter));

Parameter → NUM | DIG | ID

Up → up(ParametersInit);

Atr → ID := (OprA | Member); | ID := Bool;

Bool → true | false

OprLflt→ (OprA | Member)

OperatorL (OprA | Member)

OprA → Member OperatorA ListaA

ListaA → Member | Member OperatorA ListaA

Member → ID | NUM | DIG

OperatorA → + | - | * | /

OperatorL → > | < | <= | >= | == | !=

OprLbool → ID == Bool

POLE → pole[1-4]

ROBOT_O → robot[1-3]_o

ROBOT_P → robot[1-3]_p

ROBOT → robot[1-3]

ID → [_a-zA-Z][_a-zA-Z0-9]*

DIG → (-)?[0-9]+

NUM → (-)?[0-9]+.[0-9]+

Figura 7. Gramática da TauraLang.

A última fase executada foi a geração do código alvo, esse pode ser um código de máquina ou simplesmente um código em outra linguagem de programação. No caso do trabalho, o código final escolhido foi o *Python*, pelo fato de essa ser a linguagem usada no simulador da equipe.

O *ANTLR* foi usado também para a tradução entre a TauraLang e o *Python*. O tradutor foi construído em Java, usando o *Netbeans* e os arquivos gerados pelo *ANTLR* durante a construção do parser. Além dos arquivos gerados pelo *ANTLR*, foi necessária a criação de mais duas classes: uma classe principal, LPtoPython.java, e uma para geração da saída na linguagem alvo, Translate.java.

4. Caso de Estudo

A linguagem e o simulador 2D, da equipe Taura Bots, encontram-se no momento em níveis diferentes, pois o levantamento de requisitos foi feito em um estágio inicial da equipe e alguns pontos foram adaptados pensando nos próximos passos da equipe. Dessa forma, somente comportamentos simples escritos na TauraLang e traduzidos para *Python* executam no simulador. Como validação, primeiramente é apresentado um caso de comportamento que executa no simulador atualmente.

Os códigos a seguir descrevem um comportamento que faz com que o robô vá até bola quando ela está distante ou chute quando ela está próxima. A Figura 8 mostra esse comportamento na linguagem TauraLang e a Figura 9 mostra o código equivalente em *Python.*

```
module robo1: {
        in: [ball(r_ball, t_ball)];
        out: [walk(r_w, theta_w, phi_w), defend(x),
        kick(r_k, theta_k)];
        var: [];
        states:[attack, defense, goalkeeper, outro];
        initially: [attack];
                ball(r_ball, t_ball)#attack -> if(r_ball > 10) {
                                                emit(walk(0.5, t_ball, t_ball)); }
                                        else{
                                                emit(kick(1, 1)); }
}
```

Figura 8. IA escrita na linguagem TauraLang.

```
import time
from math import pi
robot = Simulation.start()
while robot.updateSimulation():
  world = robot.perceiveWorld()
  if not world:
    sys.exit("No world received")
  attack = True
  ball = None
  for obj in  world.objects_list:
    if obj.kind == "ball":
      ball = obj
      r_ball = ball.position.r
      t_ball = ball.position.a
  if ball and attack:
      if r_ball > 10:
robot.setMovementVector(Point2(0.5, t_ball, t_ball))
      else:
        robot.setKick(1)
```

Figura 9. IA traduzida para *Python*.

5. Trabalhos Relacionados

Uma das linguagens já existentes usadas para treinar as equipes é a linguagem *Clang*, essa é uma linguagem de propósito geral usada para comunicação entre jogadores e treinadores durante uma simulação de jogo. Foi projetada para que o treinador possa informar e aconselhar os jogadores no campo. Além disso, pode também ser usada para representar estratégias, já que as mensagens são basicamente regras de produção, onde situações são mapeadas para ações [Bonarini et al. 2003].

A linguagem permite a definição de três estruturas básicas: as condições, ações e regiões. As condições são proposições sobre o estado da simulação, por exemplo, a posição da bola e quem está com ela. Essas proposições são combinadas usando operadores lógicos. As ações correspondem aos comandos que podem ser passados para os jogadores e as regiões definem zonas do campo utilizando pontos, áreas poligonais e áreas radiais [Abreu et al. 2010].

Outra linguagem usada para a simulação 2D é a *Coach Unilang*. Essa linguagem foi desenvolvida, assim como a *Clang*, para reduzir as ambiguidades de conceitos permitindo uma melhor comunicação entre jogadores e treinadores. Essa linguagem permite treinar usando diferentes níveis de abstração: instruções, estatísticas mais modelagem do adversário e definições mais instruções. A primeira destina-se a treinar equipes com robôs inteligentes e usados para jogar juntos permitindo um alto nível de treinamento, porém pelo uso de conceitos do futebol genéricos e fixos torna-se bastante inflexível. O nível dois de treinamento visa ajudar o treinador com a parte de estatísticas do jogo como o resultado de certa ação, e com a obtenção de informações sobre a modelagem do adversário. No terceiro nível existe um alto grau de flexibilidade, permitindo que o treinador defina conceitos do futebol tanto em baixo quanto em alto nível [Lau and Reis 2002].

As linguagens apresentadas anteriormente não são adequadas ao problema apresentado neste trabalho, pois, foram desenvolvidas especificamente para simulação. Ou seja, não são voltadas para robótica e sim para regras de futebol, o que impede a portabilidade entre simulador e robô real.

6. Conclusão

As pesquisas na construção de robôs autônomos, capazes de realizar ações desempenhadas por seres humanos, estendem-se ao longo de anos. Na UFSM, a equipe Taura Bots realiza pesquisas na área de robótica, focando no desenvolvimento de robôs capazes de jogar futebol. Para a programação do comportamento dos robôs a equipe conta com um simulador, desenvolvido pela própria Taura, e utiliza linguagens como *Python* e C++.

Diante da dificuldade de programação visto que muitos não têm total conhecimentos da sintaxe das linguagens utilizadas ou não entendem sobre as regras do futebol, o que leva o programador a não saber como o robô deve comportar-se diante certa situação, surgiu a possibilidade de criação de uma linguagem de domínio específico mais intuitiva, portável e com nível maior de abstração.

O objetivo da criação desta linguagem gira em torno de facilitar a programação das ações que devem ser tomadas pelo robô durante a partida, permitindo que as estratégias possam ser criadas tanto por pessoas que tem bastante conhecimento em programação quanto por pessoas que possuem conhecimento na área de futebol, porém

não têm tanta experiência com programação.

Levando em conta a execução do trabalho, os objetivos centrais foram alcançados: A sintaxe da linguagem foi definida e validada, através da construção da gramática e do parser. E a tradução do código na linguagem TauraLang para *Python* foi realizada. A portabilidade do código ainda deve ser testada e provavelmente algumas atualizações na gramática devam ser realizadas, pois a equipe evoluiu desde a coleta de requisitos até este momento. Por outro lado algumas instruções foram mapeadas já pensando nos próximos passos da equipe, precisando assim de algumas adaptações tanto no simulador quanto no robô real. Por fim, concluiu-se que é possível a criação de uma linguagem para a programação do comportamento de robôs, porém essa precisa estar em constante evolução.

Futuramente, pode-se trabalhar em cima do refinamento e atualização da gramática, fazendo com que a linguagem tenha uma abrangência maior de casos e esteja no mesmo nível da equipe, ou seja, possa atender a todas as necessidades de programação tanto do simulador quanto do robô real. Outra possibilidade, é a tradução da TauraLang para outras linguagens de programação como C++.

Referências Bibliográfica

Abreu, P., Garganta, J., Faria, M. and Reis, L. (2010) “Knowledge representation in soccer domain: An ontology development”, Iberian Conference on Information Systems and Technologies.

Asada, M., Kitano, H., Matsubara, H., Noda, I. and Suzuki, S. (1998) “RoboCup-97: The First Robot World Cup Soccer Games and Conferences”, AI Magazine, Vol. 19, Number 3. USA, Massachusetts.

Bonarini, A., Bronwing, B., Polani, D., and Yoshida, K (2003) “RoboCup 2003: Robot Soccer World Cup VII”, London, UK.

Lau, N. and Reis, L. (2002) “COACH UNILANG - A Standard Language for Coaching a (Robo) Soccer Team”, Springer-Verlag Berlim Heidelberg, Alemanha, Berlin.

Librelotto, G., Toscani, S. and Turchetti, R. (2007) “A Semântica Operacional da Linguagem RS”, HÍFEN, volume 31, número 59/60, Uruguaiana, Rio Grande do Sul. Disponível em <http://revistaseletronicas.pucrs.br/ojs/index.php/hifen/article/download/3880/2948>, Último acesso em: Março, 2016

Montenegro, F. (2015) “Simulador Computacional Para Auxiliar No Desenvolvimento De Estratégias De Comportamento Para Futebol de Robôs”, Trabalho de Conclusão de Curso, Ciência da Computação, Universidade Federal de Santa Maria (UFSM), Santa Maria, Rio Grande do Sul.

RoboCup. (2012) “Em:<www.robocup.org.br/index.php>”. Brasil, São Paulo: Site Oficial RoboCupBrasil. Último Acesso: Março, 2016.

Toscani, S. (1993) “RS: Uma Linguagem para Programação de Núcleos Reactivos”, Tese de doutoramento, Depto de Informática da UNL, Portugal.

Toscani, S. and Monteiro, F. (1994) “Apresentação da linguagem reativa síncrona RS”, Simpósio Brasileiro de Engenharia de Software, 1994, Curitiba, Paraná.