



**UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA / INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MECATRÔNICA**

RUI CARLOS BOTELHO ALMEIDA DA SILVA

**VERIFICAÇÃO FORMAL DE PLANOS PARA AGENTES
AUTÔNOMOS E SISTEMAS MULTIAGENTES: UM ESTUDO
DE CASO APLICADO AO FUTEBOL DE ROBÔS**

Salvador
2012

RUI CARLOS BOTELHO ALMEIDA DA SILVA

**VERIFICAÇÃO FORMAL DE PLANOS PARA AGENTES
AUTÔNOMOS E SISTEMAS MULTIAGENTES: UM ESTUDO
DE CASO APLICADO AO FUTEBOL DE ROBÔS**

Dissertação apresentada ao Programa de Pós-graduação em Mecatrônica – PPGM, Escola Politécnica e Instituto de Matemática, Universidade Federal da Bahia, como requisito para obtenção do grau de Mestre em Mecatrônica.

Orientadora: Profa. Dra. Aline Maria Santos Andrade

Co-orientador: Prof. Dr. Augusto César Pinto Loureiro da Costa

Salvador
2012

Sistema de Bibliotecas da UFBA

Silva, Rui Carlos Botelho Almeida da.

Verificação formal de planos para agentes autônomos e sistemas multiagentes : um estudo de caso aplicado ao futebol de robôs / Rui Carlos Botelho Almeida da Silva. - 2012.
175 f.: il.

Inclui apêndices.

Orientadora: Profª. Drª. Aline Maria Santos Andrade.

Co-orientador: Prof. Dr. Augusto César Pinto Loureiro da Costa.

Dissertação (mestrado) - Universidade Federal da Bahia, Escola Politécnica, Instituto de Matemática, Salvador, 2012.

1. Engenharia de software. 2. Inteligência artificial distribuída. 3. Robôs - Dinâmica - Simulação por computador. I. Andrade, Aline Maria Santos. II. Costa, Augusto César Pinto Loureiro da. III. Universidade Federal da Bahia. Escola Politécnica. IV. Universidade Federal da Bahia. Instituto de Matemática. V. Título.

CDD - 005.1

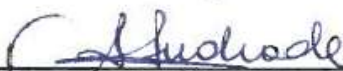
CDU - 004.4

TERMO DE APROVAÇÃO

RUI CARLOS BOTELHO ALMEIDA DA SILVA

VERIFICAÇÃO FORMAL DE PLANOS DE AGENTES AUTÔNOMOS E SISTEMAS MULTIAGENTES: UM ESTUDO DE CASO APLICADO AO FUTEBOL DE ROBÔS.

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Mecatrônica, Universidade Federal da Bahia, pela seguinte banca examinadora:



Dra. Aline Maria Santos Andrade – Orientadora
Doutora em Informática pela Pontifícia Universidade Católica do Rio de Janeiro, Brasil
Universidade Federal da Bahia – UFBA



Dr. David Boris Paul Déharbe
Doutor em Informática pela Université Joseph Fourier - Grenoble 1, Nancy, França
Universidade Federal do Rio Grande do Norte – UFRN



Dr. Flávio Morais de Assis Silva
Doutor em Informática pela *Technische Universität Berlin – TUB*, Berlim, Alemanha
Universidade Federal da Bahia - UFBA

Salvador, 09 de março de 2012.

A
Antônio Carlos Botelho, Judith Lacerda Botelho e Amália Rosa Cruz de Almeida
avós queridos e saudosos que foram e sempre serão modelos de perseverança, fraternidade, honestidade e
sabedoria.

AGRADECIMENTOS

Aos meus pais, José Carlos e Cristina, com seu amor e dedicação a família, formaram os fundamentos do meu caráter. Obrigado por serem a minha referência e estarem sempre presentes na minha vida, me apoiando e incentivando.

À minha esposa, Mila, minha amiga e companheira. Obrigado por me fazer sentir tão amado, por me apoiar em tudo o que faço e por iluminar os meus caminhos.

À Bruno, meu filho, legado e orgulho da minha existência. Obrigado por ter o privilégio de ter você na minha vida e me fazer acreditar, independente do eu tenha realizado ou venha a realizar, que já contribuí para um mundo melhor por ter trazido para ele uma pessoa tão especial como você.

À minha irmã e “cumadre” Lea, a meu afilhado Mateus e a Rogério, cunhado e “cumpadre”, depositários de extrema confiança e respeito.

Aos amigos e colegas de fora e dentro do mestrado, em especial Fred Barboza, que de perto ou de longe, sempre se mostraram solidários. Vocês que aliviaram minhas horas difíceis, me alimentando de certezas, força e alegria.

À minha orientadora, Aline Andrade, pela confiança ao acolher a idéia deste projeto e pelo apoio em todos os momentos. Obrigado pela paciência e por acreditar em mim, mesmo quando, em alguns momentos de especial dificuldade, esmureci. Nestas ocasiões, você, como grande educadora que é, me fez enxergar os problemas sob uma nova ótica e, desta perspectiva, me ajudou a encontrar as soluções, a vencer as adversidades e a reconhecer e suplantar as minhas deficiências.

Por fim, não posso deixar reverenciar a força maior que nos anima e que permeia a tudo e a todos, a quem denominamos Deus. Ainda que continue a buscar as explicações de tudo o que me rodeia na ciência, tenho fé de que quanto mais conheço, mais descubro a infinidade de coisas a desvendar e mais acredito e admiro a sua obra. Obrigado Pai.

“A ciência é uma só.”

SILVA, Rui Carlos Botelho Almeida da. Verificação formal de planos: Um estudo de caso aplicado ao futebol de robôs. 175f. :il 2012. Dissertação (Mestrado) – Escola Politécnica / Instituto de Matemática, Programa de Pós-Graduação em Mecatrônica, Universidade Federal da Bahia, Salvador, 2012.

RESUMO

Os Agentes Autônomos – AA e os Sistemas Multiagentes – SMA realizam suas tarefas baseados num planejamento e a sua complexidade vai depender de qual ambiente esteja envolvido, principalmente quando este ambiente é dinâmico e não determinista.

A verificação de modelos tem sido aplicada para a verificação de propriedades do planejamento de modo a checar a correção de aplicações baseadas em AAs e SMA's e tal verificação apresenta muitos desafios para contornar situações potenciais de explosão de estados. O futebol de robôs simulados é uma aplicação que apresenta muitas das características e problemas inerentes aos AA's e SMA's como um ambiente não determinista e dinâmico, fato este que vem tornando esta aplicação um relevante estudo de caso para a verificação de modelos de SMA's.

O presente trabalho considera a especificação e verificação de planos de um time de futebol de robôs simulado, o qual é baseado na arquitetura multicamada de Agentes Concorrentes (camada cognitiva, camada instintiva, e camada reativa), utilizando o verificador de modelos UPPAAL.

Para atingir os objetivos do trabalho foi proposta uma abordagem incremental e evolutiva para modelar e verificar os planos, a qual inclui abstrações e técnicas baseadas em verificação composicional de modelos (*Compositional Model Checking*), com o objetivo de contornar situações de explosão de estados. O método proposto também pode ser utilizado em aplicações similares, o qual poderia ser suportado por um ambiente computacional interativo para guiar os analistas no processo de verificação de planos de SMA's com arquitetura multicamada, usando a verificação de modelos.

Palavras-chave: Métodos Formais. Verificação de Modelos. Agentes Autônomos. Sistemas Multiagentes. Futebol de Robôs.

SILVA, Rui Carlos Botelho Almeida da. Formal Verification of plans: A study case applied to robot soccer. 175p. :il 2012. Dissertation (master) – Polytechnic School / Institute of Mathematics, Post Graduation Program on Mechatronics, Federal University of Bahia, Salvador, 2012.

ABSTRACT

Autonomous Agents - AA and Multi-agent Systems - MAS execute their actions based on planning which depends on the environment involved and it can be very complex, mainly when the environment is dynamic and non deterministic.

Model Checking has been applied to verify planning properties in order to check the correctness of AAs or MASs applications and model checking these types of systems presents many challenges in order to circumvent situations of state explosion. The simulated robot soccer is an application that shares the characteristics and problems inherent in AA and MAS, having a non deterministic and dynamic environment with partial vision, been a relevant case study of MAS model checking.

This work considers the modeling and verification of plans using the model checker UPPAAL of a simulated robot soccer team which is based on a multi-layer architecture (cognitive layer, instinctive layer and reactive layer) of concurrent AAs. We propose an incremental and evolutionary approach to model and verify the plans, which comprises abstractions and techniques based on compositional model checking in order to circumvent the state space explosion problem. Our method can also be used in similar applications and we forecast an iterative computer environment to guide analysts in the process of verification of concurrent multi-agent systems with multi-layer architecture plans using model checking.

Keywords: Formal Methods. Model Checking. Autonomous Agents. Multiagent Systems. Robot Soccer.

LISTA DE FIGURAS

Figura 1. Níveis de habilidades de agentes exibidos por diferentes tipos de agentes (adaptado) [20].	21
Figura 2. Representação esquemática da arquitetura de um agente autônomo concorrente de acordo com o fluxo de informações [24].	30
Figura 3. Visão do jogo proporcionada pelo <i>Soccermonitor</i> [25].	47
Figura 4. Modelo da comunicação entre os clientes (agentes) e o servidor (<i>soccerserver</i>) [25].	48
Figura 5. Autômato do plano do agente autônomo reativo de [28].	32
Figura 6. Autômato do conjunto de ações do agente autônomo reativo de [28].	32
Figura 7. Arquitetura dos agentes utilizados em [14].	38
Figura 8. Interface do editor do UPPAAL.	37
Figura 9. Interface do simulador do UPPAAL.	38
Figura 10. Interface do verificador do UPPAAL.	38
Figura 11. Interface do verificador do UPPAAL com resultado de propriedade não satisfeita.	39
Figura 12. Arquitetura do sistema baseado em conhecimento utilizado no Mecateam[24].	50
Figura 13. Representação da arquitetura de agente concorrente do mecateam.	51
Figura 14. Relação entre planos cognitivos e instintivos e os agentes (jogadores) do Mecateam.	52
Figura 15. Diagrama de fluxo de transições entre estados oriundo das regras cognitivas do goleiro.	52
Figura 16. Diagrama de fluxo de transições entre estados oriundos das regras cognitivas dos jogadores de defesa.	52
Figura 17. Diagrama de fluxo de transições entre estados oriundos das regras cognitivas dos jogadores de meio de campo e ataque.	53
Figura 18. Interação dos diversos níveis de um agente, do ambiente e da interface.	62
Figura 19. Interação dos diversos níveis de cada agente e da equipe como um todo, do ambiente e da interface.	59
Figura 20. Representação do modelo de verificação incremental e evolutivo adotado.	68
Figura 21. Autômato das Regras Cognitivas do Goleiro.	70
Figura 22. Autômato das Regras Cognitivas dos Jogadores de Defesa.	70
Figura 23. Autômato das Regras Cognitivas dos Jogadores de Meio de Campo e Ataque.	71
Figura 24. Declaração das variáveis cognitivas no UPPAAL.	71
Figura 25. Definição da relação entre os jogadores e as posições dos vetores de variáveis do sistema.	72
Figura 26. Divisão do campo em 06 regiões para o posicionamento dos jogadores e da bola.	77
Figura 27. Instância das regras de um jogador de cada grupo no UPPAAL.	87
Figura 28. Alteração das informações sobre os possíveis valores da variável <i>local_goal current</i> .	93
Figura 29. Especificação da regras <i>Mark_Hold_Ball</i> considerando o novo estado <i>kick_to_goal</i> ⇔ $LocalGoalCurrent[pNR] = 5$.	93
Figura 30. Caminho executado pela regras até o deadlock provocado pela regra <i>rule_ending_achieved</i> representada pelo <i>Message Sequence Chart</i> (MSC) do UPPAAL.	96

Figura 31. Resultado da verificação da propriedade da regra <i>Side_Attack_Pass_Ball</i> do jogador 1. ...	97
Figura 32. Resultado da verificação da propriedade das regras do jogador 2 inseridas nesta etapa...	98
Figura 33. Resultado da verificação da propriedade de não ocorrência de deadlocks nas regras do jogador 2.....	98
Figura 34. Resultado da verificação das propriedades das regras do jogador 4 inseridas nesta etapa.	99
Figura 35. Resultado da verificação da propriedade das regras do jogador 5 inseridas nesta etapa.	100
Figura 36. Resultado da verificação da propriedade das regras do jogador 6 inseridas nesta etapa.	101
Figura 37. Resultado da verificação da propriedade de alcançabilidade do estado <i>drive_ball_forward</i> da regra <i>P6_ADBFW</i>	101
Figura 38. Resultado da verificação da propriedade das regras do jogador 7 inseridas nesta etapa.	102
Figura 39. Resultado da verificação da propriedade de alcançabilidade do estado <i>drive_ball_forward</i> da regra do jogador 7 <i>P7_ADBFS</i>	102
Figura 40. Resultado da verificação da propriedade das regras do jogador 8 inseridas nesta etapa.	103
Figura 41. Resultado da verificação da propriedade das regras do jogador 9 inseridas nesta etapa.	104
Figura 42. Resultado da verificação da propriedade das regras do jogador 10 inseridas nesta etapa.	104
Figura 43. Representação das principais funções realizadas pelo <i>Soccerserver</i>	107
Figura 44. Representação do ambiente após sucessivos refinamentos.	109
Figura 45. Autômato do ambiente <i>Soccerserver</i>	109
Figura 46. Cenário ofensivo da equipe oponente com <i>SoccerserverBallPosition</i> ≤ 1	115
Figura 47. Representação de possíveis evoluções do (a) cenário ofensivo da equipe oponente com <i>SoccerserverBallPosition</i> ≤ 1 , (b) chute além do gol, (c) jogador mantém posse de bola e (d) gol realizado.	116
Figura 48. Representação esquemática do sistema de marcação por zona da equipe oponente.	117
Figura 49. Modelos de regras cognitivas goleiro com destaque para o canal <i>StartGame?</i> e a função <i>SetInitialValues()</i>	120
Figura 50. Comparativo entre as versões da regra <i>Mark_Hold_Ball</i> (a) utilizada na verificação das regras dos jogadores individualmente e (b) modificada para a verificação da equipe.	121

LISTA DE TABELAS

Tabela 1. Nomes, propriedades, descrição e equivalência entre TCTL e CTL.....	39
Tabela 2. Comparativo de algumas regras cognitivas dos três grupos de jogadores.	55
Tabela 3. Comparativo das regras cognitivas equivalentes dos três grupos.	56
Tabela 4. Relação entre as variáveis nas regras dos planos cognitivos e no modelo.	71
Tabela 5. Resumo dos jogadores e respectivas regras instintivas.....	74
Tabela 6. Resumo das Variáveis X Regras Instintivas.	76
Tabela 7. Resumo das variáveis existentes no nível instintivo, seus valores possíveis e tipos.	76
Tabela 8. Relação das regiões do campo e valores da variável <i>player localization</i>	77
Tabela 9. Resumo das regras cognitivas comuns por tipos de jogadores.....	86
Tabela 10. Resumo das regras instintivas comuns por tipos de jogadores.....	87
Tabela 11. Descrição e fórmulas em CTL das propriedades verificadas.	88
Tabela 12. Resultado das verificações das propriedades das regras cognitivas do jogador1 (grupo goleiro).....	89
Tabela 13. Propriedades de alcançabilidade de estados das regras instintivas do jogador 1 verificadas e satisfeitas.	89
Tabela 14. Resultado das verificações das propriedades das regras cognitivas do jogador2 (grupo defesa).....	90
Tabela 15. Propriedades de alcançabilidade de estados das regras instintivas do jogador 2 verificadas e satisfeitas.	90
Tabela 16. Resultado das verificações das propriedades das regras cognitivas do jogador7 (grupo meio de campo - ataque).....	91
Tabela 17. Propriedades de alcançabilidade de estados das regras instintivas do jogador 7 verificadas e satisfeitas.	91
Tabela 18. Resultado da verificação de ocorrência de inconsistência entre os estados dos jogadores e o valor da variável <i>LocalGoalCurrent[]</i> , após correção da regra instintiva <i>rule_mark_hold_ball</i>	94
Tabela 19. Resultado da verificação de ocorrência de deadlocks, após correção da regra instintiva <i>rule_mark_hold_ball</i>	95
Tabela 20. Resumo dos valores definidos para as variáveis.	114
Tabela 21. Resumo dos valores definidos para as variáveis utilizadas na marcação.....	116
Tabela 22. Resumo dos valores definidos por jogador para as variáveis locais quando a equipe tem a posse de bola no início a partida.	119
Tabela 23. Resumo dos valores por jogador para as variáveis locais quando equipe não tem a posse de bola no início a partida.....	119
Tabela 24. Propriedades e resultados da verificação do ambiente.....	126
Tabela 25. Propriedades que apresentaram resultados e foram satisfeitas na verificação da equipe com o ambiente.....	128

Tabela 26. Propriedades que não apresentaram resultados na verificação da equipe.....129

LISTA DE QUADROS

Quadro 1. Algoritmo do plano do agente autônomo de [28].	32
Quadro 2. Implementação do plano do agente autônomo de [28] na linguagem NQC.	33
Quadro 3. Representação das regras de produção em BNF [22].	50
Quadro 4. Regras cognitivas dos jogadores de meio de campo e ataque.	53
Quadro 5. Regras cognitivas dos jogadores de defesa.	54
Quadro 6. Regras cognitivas do goleiro.	55
Quadro 7. Exemplos de regras instintivas.	58
Quadro 8. Exemplos de regras cognitivas que mantém ou alteram o estado de um jogador.	54
Quadro 9. Interação das regras cognitivas e instintivas e o nível reativo.	60
Quadro 10. Declaração das variáveis encontradas nas regras instintivas no UPPAAL.	78
Quadro 11. Relação entre a regra <i>mark_opponent</i> e sua especificação no UPPAAL.	79
Quadro 12. Função <i>setVariables()</i> vinculada à regra <i>mark_opponent</i> .	80
Quadro 13. Relação entre as regras <i>advance_achieved</i> e <i>advance_fail</i> e suas especificações no UPPAAL.	82
Quadro 14. Destaque da mudança de estado local do jogador via regra <i>rule_mark_hold_ball</i> .	92
Quadro 15. Relação entre a alteração da regra <i>rule_mark_hold_ball</i> e o autômato equivalente especificado no UPPAAL.	94
Quadro 16. Regra <i>rule_ending_achieved</i> utilizada pelo goleiro.	96
Quadro 17. Declaração das variáveis globais e canais do ambiente.	111
Quadro 18. Código da função <i>setAction()</i> da regra <i>Mark_Hold_Ball</i> .	122
Quadro 19. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Search_Ball</i> .	156
Quadro 20. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Intercept_Ball</i> .	157
Quadro 21. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Strategic_Position</i> .	158
Quadro 22. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Mark_Opponent</i> .	158
Quadro 23. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Pass_Ball_Uncond</i> .	159
Quadro 24. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Pass_Ball_Cond</i> .	160
Quadro 25. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Drive_Ball_Forward</i> .	161
Quadro 26. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Pass_Ball_Forward</i> .	162

Quadro 27. Comparativo das regras <i>Advance_Pass_Ball_Forward</i> e <i>Side_Attack_Pass_Ball_Forward</i> e seus respectivos autômatos.	163
Quadro 28. Código da função <i>setAction()</i> das regras que acionam o comportamento reativo <i>Pass_Ball_Fast</i> da regra <i>Advance_Pass_Ball_Fast</i>	163
Quadro 29. Código da função <i>setAction()</i> <i>Side_Attack_Kick_to_Goal</i>	164

LISTA DE ABREVIATURAS E SIGLAS

AA	- Agente Autônomo / <i>Autonomous Agent</i>
SMA	- Sistema Multiagente
MAS	- <i>Multiagent System</i>
UPPAAL	- Conjunto de ferramentas para a verificação de modelos de sistemas de tempo real utilizando autômatos com tempo desenvolvido pelas universidades de Uppsala e Aalborg
IA	- Inteligência Artificial
IAD	- Inteligência Artificial Distribuída
UDP / IP	- <i>User Datagram Protocol / Internet Protocol</i>
NQC	- <i>Not Quite C</i> – Linguagem de programação
CTL	- <i>Computation Tree Logic</i>
LTL	- <i>Linear Temporal Logic</i>
OBDD	- <i>Ordered Binary Decision Diagram</i>
CIRCA	- <i>Cooperative Intelligent Real-Time Control Architecture</i>
SCALA	- <i>Système Coopératif d'Agents Logiciels Autonomes</i>
NASA	- <i>National Aeronautics Space Administration</i>
HSTS	- <i>Heuristic Scheduler and Testbed System</i>
PHAVer	- <i>Polyedral Hybrid Automata Verifier</i>
MATLAB	- <i>Matrix Laboratory - Software</i>
TCTL	- <i>Timed Computation Tree Logic</i>
SBRP	- Sistema Baseado em Regras de Produção
BNF	- <i>Backus-Naur Form</i>
pNR	- Variável <i>player Number</i>
1D	- Unidimensional
2D	- Bidimensional
bP	- Variável <i>ball Position</i>
pBK	- Variável <i>player Ball Kickable</i>
tHB	- Variável <i>teammate Has Ball</i>
pBP	- Parâmetro <i>Ball Position</i>
pPBK	- Parâmetro <i>Player Ball Kickable</i>
pTHB	- Parâmetro <i>Teammate Has Ball</i>
MSC	- <i>Message Sequence Chart</i>
KB	- Kilobyte
s	- Segundo

ms - Milisegundo
oPHB - Variável *opponent Player Has Ball*
pTOA - Variável *player Type Of Action*
pSOA - Variável *player Succeed Of Action*
tA - Variável *trying Action*

LISTA DE SÍMBOLOS

M	- Modelo
Φ	- Propriedade
E	- Operador de caminho para referenciar um ou mais futuros em CTL
A	- Operador de caminho para referenciar todos os futuros possíveis em CTL
φ	- Valor de um estado
F	- Operador modal para referenciar em um estado no futuro em CTL
G	- Operador modal para referenciar todos os estados possíveis em CTL
and	- Operador lógico “E” em TCTL
or	- Operador lógico “OU” em TCTL
imply	- Operador lógico “IMPLICA EM” em TCTL
not	- Operador lógico “NÃO” em TCTL
[]	- Operador temporal “sempre” em TCTL
<>	- Operador temporal “em algum momento” em TCTL
-->	- Operador temporal “leva a” em TCTL

SUMÁRIO

1. INTRODUÇÃO	21
2. AGENTES E SISTEMAS MULTIAGENTES.....	24
2.1. DEFINIÇÃO DE AGENTES E SISTEMAS MULTIAGENTES	24
2.2. CLASSIFICAÇÃO DE AGENTES E SISTEMAS MULTIAGENTES	25
2.3. ARQUITETURAS DE AGENTES.....	27
2.3.1. Arquitetura Deliberativa	27
2.3.2. Arquitetura Reativa.....	27
2.3.3. Arquitetura Híbrida	28
2.3.4. Agente Autônomo Concorrente.....	29
2.4. PLANEJAMENTO E PLANOS.....	30
3. MÉTODOS FORMAIS.....	35
3.1. VERIFICAÇÃO DE MODELOS.....	35
3.1.1. O Verificador de Modelos UPPAAL.....	36
3.1.2. Verificação Composicional de Modelos.....	40
3.2. VERIFICAÇÃO DE AA's E SMA's.....	42
4. FUTEBOL DE ROBÔS.....	46
4.1. O AMBIENTE DO FUTEBOL DE ROBÔS SIMULADO	46
4.2. MECATEAM.....	48
4.1.1. Sistema Baseado em Conhecimento	49
4.2.2. Representação dos Planos.....	51
4.2.2.1. Planos Cognitivos.....	52
4.2.2.2. Planos Instintivos	58
5. ESPECIFICAÇÃO E VERIFICAÇÃO DOS PLANOS.....	65
5.1. MÉTODO DE VERIFICAÇÃO E AMBIENTE UTILIZADOS NO PROCESSO DE MODELAGEM E VERIFICAÇÃO	65
5.2. ESPECIFICAÇÃO DOS PLANOS.....	68
5.2.1. Especificação dos Planos Cognitivos.....	69
5.2.2. Especificação dos Planos Instintivos	72
5.2.2.1 Análise e especificação das variáveis utilizadas nas regras instintivas	74
5.2.2.2 Modelagem e especificação das regras instintivas.....	79
5.3. APLICAÇÃO DE TÉCNICAS DE VERIFICAÇÃO COMPOSICIONAL DE MODELOS.....	83
6. VERIFICAÇÃO DOS PLANOS DOS JOGADORES.....	86
6.1. VERIFICAÇÃO DOS PLANOS POR TIPOS DE JOGADORES.....	86
6.1.1. Identificação das propriedades verificadas.....	88
6.1.2. Verificação das regras do grupo goleiro.....	88

6.1.3. Verificação das regras do grupo jogadores de defesa.....	89
6.1.4. Verificação das regras do grupo jogadores de meio de campo - ataque.....	91
6.1.5. Ajustes das regras e resolução de problemas encontrados na verificação dos grupos de jogadores	92
6.2. VERIFICAÇÃO DE CADA JOGADOR INDIVIDUALMENTE.....	97
6.2.1. Verificação das regras do jogador 1.....	97
6.2.2. Verificação das regras de cada jogador de defesa.....	98
6.2.3. Verificação das regras de cada jogador de meio de campo e ataque.....	100
7. ESTENDENDO A ESPECIFICAÇÃO PARA A VERIFICAÇÃO DA EQUIPE.....	106
7.1. ANÁLISE DO AMBIENTE.....	106
7.2. MODELAGEM DO AMBIENTE.....	109
7.2.1. Variáveis e funções de controle do ambiente.....	111
7.2.2. Canais de sincronização.....	112
7.2.3. Representação do comportamento da equipe adversária.....	113
7.3. EVOLUÇÃO DOS MODELOS DAS REGRAS COGNITIVAS DOS JOGADORES.....	118
7.4. EVOLUÇÃO DOS MODELOS DAS REGRAS INSTINTIVAS DOS JOGADORES.....	120
7.4.1. Adição da variável tA e Alteração das funções setAction()	121
7.4.2. Alteração da função setVariables()	124
8. VERIFICAÇÃO DO AMBIENTE E DA EQUIPE.....	125
8.1. VERIFICAÇÃO DO AMBIENTE.....	125
8.2. VERIFICAÇÃO DA EQUIPE.....	127
8.2.1. Nova verificação das propriedades dos jogadores com o ambiente	127
8.2.2. Verificação da equipe	127
8.2.3. Considerações sobre a verificação	129
9. CONCLUSÃO	131
9.1. CONSIDERAÇÕES SOBRE AS SOLUÇÕES ADOTADAS.....	131
9.2. TRABALHOS FUTUROS.....	133
REFERÊNCIAS.....	134
APÊNDICES.....	138

1. INTRODUÇÃO

Nas últimas décadas, as pesquisas envolvendo o desenvolvimento de Agentes Autônomos - AA (*Autonomous Agents*) e Sistemas Multiagentes - SMA (*Multiagents Systems*) vêm tendo grande destaque na área acadêmica e na indústria. Este fato se deve a utilização desses AA's ou SMA's, quer sejam entidades virtuais ou reais, nos mais diversos tipos de aplicações.

Dentre estes campos de atuação onde os AA's e os SMA's podem ser empregados, podem ser citadas, como principais áreas, a robótica, a engenharia de software e o controle e automação. Os robôs autônomos podem ser utilizados, individual ou coletivamente, para controle embarcado em explorações espaciais ou submarinas, reabilitação de pessoas com algum tipo de limitação motora, exploração e monitoramento aéreo de difícil acesso ou com grandes extensões territoriais, convívio social ou lazer, etc. Nas indústrias, pode-se aplicá-los para controle de sistemas de manufatura integrados por computador ou monitoramento de processos. Um exemplo para esse caso seria um SMA controlando robôs para pintura automotiva. Podem ser utilizados, além disso, em diversas outras aplicações em software, como controle de tráfego aéreo, na engenharia de software (em ferramentas de desenvolvimento ou teste de sistemas) ou robôs de busca de informações na *internet* (*softbots*).

O que motiva a utilização dos AA's e SMA's são as características inerentes e desejáveis destas entidades: autonomia, mobilidade, cooperação, comunicação e inteligência. Os AA's e SMA's têm capacidade de interagir (sentir e atuar) com o ambiente (quer sejam estes estáticos ou dinâmicos, simples ou complexos); de manter um certo grau de controle do seu estado interno e do ambiente (persistência); de cumprir seus objetivos (realizar tarefas ou prover serviços); de se comunicar com outras entidades (físicas ou virtuais); e, principalmente, de realizar tudo isso sem a intervenção direta (ou com a intervenção limitada) de entidades humanas.

A atividade fim destes sistemas está relacionada com o cumprimento de objetivos que são realizados com base em um planejamento, envolvendo capacidades citadas acima. O planejamento pode ser descrito como a tarefa de

apresentar um conjunto de ações (plano) para alcançar certo objetivo, a depender do ambiente no qual os agentes estão inseridos. Existem diversos algoritmos de planejamento diferentes, que podem ser divididos, em relação ao ambiente, em dois grupos: para ambientes observáveis e deterministas e para ambientes parcialmente observáveis e não-deterministas [1].

Garantir que AA's, individualmente, ou SMA's, coletivamente, possuam planos corretos é uma tarefa necessária para que estes não apresentem resultados indesejados e atinjam resultados desejáveis. Os métodos formais podem ser utilizados como ferramenta, quando se deseja primar pela correção de sistemas, trabalhando de forma complementar as técnicas convencionais de engenharia de software (como testes ou simulações) [2], a fim de permitir o desenvolvimento de sistemas mais robustos e confiáveis [3].

O futebol de robôs simulado possui características e problemas comuns a muitas áreas de pesquisa, o que o torna um ambiente muito favorável para estudos, podendo ser instanciado para diversas aplicações e campos de atuação, tais como: robótica, engenharia de software e controle e automação. Além disso, o ambiente deste sistema é bastante complexo, pois é não determinista e os agentes têm apenas visão parcial do ambiente e, portanto, o problema de especificação e verificação de planos é complexa e envolve questões em aberto que atraem a comunidade de pesquisa tanto da área de inteligência artificial como da área de métodos formais.

O objetivo principal deste trabalho é o estudo e aplicação de métodos formais para validação de planos de agentes autônomos e sistemas multiagentes, tendo como ambiente de estudo o futebol de robôs simulado (Robocup) e como objeto de estudo os planos da equipe de futebol de robôs da UFBA, Mecateam.

Este estudo se justifica pelos desafios proporcionados na especificação e validação de planejamentos individuais e coletivos. O problema de verificar planos neste contexto é complexo e métodos formais têm sido aplicados nestes problemas e, em especial, a verificação de modelos tem sido explorada em vários trabalhos de pesquisa [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] uma vez que estas técnicas permitem a verificação automática de propriedades.

Os trabalhos [4,5] baseiam-se na abordagem para a geração automática de planos, em ambientes não deterministas com visão total do ambiente. Em [6,7] o

modelo de [4,5] é evoluído situações em ambientes com visão parcial que utilizam técnicas como *Ordered Binary Decision Diagram* – OBDD e remoção de ciclos nos grafos dos planos para uma representação mais compacta dos mesmos. O trabalho [8,9] baseia-se na validação e controle de planos multiagentes para um ambiente de simulação tática para aeronaves de combate, o qual é altamente dinâmico, não determinista e com visão parcial. Em [10, 11, 12] é apresentado um estudo de caso de verificação de um sistema controlador e agendador de planos. Em [13] são demonstradas as possibilidades de uso de autômatos híbridos para agentes autônomos em ambiente dinâmico e com visão parcial. O trabalho apresentado em [14] utilizou ferramentas de model checking e simulação para verificação de times de futebol de robôs em tempo de execução

Contudo, como na maioria dos trabalhos acima, quando estes envolvem sistemas muito grandes e complexos, como o futebol de robôs, os quais, facilmente podem recair em situações de explosão de estados, é necessária a aplicação de técnicas que lidem com este problema e levem a resultados satisfatórios. Para tentar resolver o problema de modelar e verificar um sistema grande e contornar a potencial explosão de estados, este trabalho propõe uma abordagem incremental e evolutiva de verificação, aliado ao emprego de abstrações e técnicas baseadas na verificação composicional de modelos.

Em particular, autômatos são utilizados para a especificação dos planos e o verificador de modelos UPPAAL como ferramenta para a modelagem e verificação.

Para apresentação deste estudo no capítulos 2 são apresentados os conceitos inerentes aos agentes, sistemas multiagentes e planejamento. O capítulo 3 trata de verificação formal e trabalhos relacionados. No capítulo 4 apresenta informações sobre o futebol de robôs, suas características e o seu ambiente. O capítulo 5 apresenta a análise da arquitetura da equipe de futebol de robôs da UFBA e da sua representação dos planos. O capítulo 5 trata da metodologia de trabalho e da especificação dos planos dos agentes do Mecateam. No capítulo 6 são apresentadas as verificações dos agentes. No capítulo 7 é apresentada a especificação e verificação do ambiente. No capítulo 8 são apresentados os resultados das verificações do ambiente e da equipe conjuntamente. Por fim, o capítulo 9, apresenta a conclusão do trabalho e sugestões para trabalhos futuros.

2. AGENTES E SISTEMAS MULTIAGENTES

O termo agente (assim como seu correlato coletivo, sistema multiagente) está relacionado com a área de Inteligência Artificial (IA), cuja definição é expressa como conceito central da própria definição da IA, a qual é descrita como “uma subárea da Ciência da Computação que tem como alvo a construção de agentes que apresentem aspectos de comportamento inteligente”[15].

Para compreender melhor as idéias e os conceitos que envolvem tal área de estudos, as seções a seguir apresentam a definição, a classificação e as arquiteturas de agentes e sistemas multiagentes.

2.1. DEFINIÇÃO DE AGENTES E SISTEMAS MULTIAGENTES

Ao longo das últimas décadas, o conceito de agentes foi algo muito controverso entre os pesquisadores, ou correntes de pesquisa, deste tema. Esta conceituação variava bastante, ou mesmo sutilmente, a depender do tipo (ou natureza) do agente e do emprego deste, suscitando assim muitos questionamentos e as mais variadas definições sobre o que é um agente [16].

Em todas as definições existentes estava claro que era mais simples apresentar as características ou propriedades do agente do que, simplesmente, defini-lo. Tais propriedades se relacionam, mais ou menos explicitamente com as capacidades de interagir (sentir e atuar) com o ambiente (quer sejam estes estáticos ou dinâmicos, simples ou complexos), de manter certo grau de controle do seu estado interno e do ambiente (persistência), de cumprir seus objetivos (realizar tarefas ou prover serviços), de se comunicar com outras entidades (físicas ou virtuais) e, principalmente, de realizar tudo isso sem a intervenção direta (ou com a intervenção limitada) de entidades humanas.

Desta forma, considerando todas as propriedades acima elencadas, um agente, atualmente, pode ser definido de forma genérica como “uma entidade real ou abstrata que é capaz de agir sobre ela mesma e sobre seu ambiente, dispondo de uma representação parcial deste ambiente, sendo que, em um universo multiagente, pode comunicar-se com outros agentes, e cujo comportamento é

conseqüência de suas observações, de seu conhecimento e das interações com outros agentes“ [17].

Logo, tendo em vista um único agente como um componente de um sistema, onde vários agentes atuam para a solução de um problema maior, tem-se que o conjunto destes componentes (agentes) forma um sistema multiagente (SMA) – *Multiagent Systems* (MAS) [18].

Os SMA consideram a cooperação entre agentes, pressupondo um comportamento grupal inteligente. Isto significa que, mesmo que o agente individualmente não possua uma expressiva capacidade deliberativa / cognitiva, o comportamento coletivo de um conjunto de agentes pode ser considerado “inteligente” [19].

2.2. CLASSIFICAÇÃO DE AGENTES E SISTEMAS MULTIAGENTES

Da mesma forma que sempre houve bastante controvérsia na definição de agentes, a classificação destes tem muitas abordagens diferentes. Assim sendo, os mais variados pontos de vistas são adotados como forma de classificação, tais quais: tipo de tarefas que executam, arquitetura de controle, capacidade de percepção do ambiente, capacidade de atuação no ambiente, tipo de estado interno, habilidades dos agentes (autonomia, mobilidade, etc.), agentes de vida artificial, além de muitas outras.

No que diz respeito à classificação por habilidades, esta determina que cada propriedade que define um agente corresponde a uma das seguintes dimensões: autonomia, mobilidade, cooperabilidade, comunicabilidade e inteligência estão relacionadas com as seguintes habilidades. Cada dimensão possui uma gradação crescente natural [20], como está ilustrado na figura 1. As características das dimensões são as seguintes:

- Autonomia: delegação, negociação;
- Mobilidade: estática, execução remota, migração;
- Cooperabilidade: isolamento, estática, dinâmica;
- Comunicabilidade: isolada, troca de dados simples, passagem de dados semânticos, conversação;
- Inteligência: preferência, planejamento, raciocínio, aprendizado.

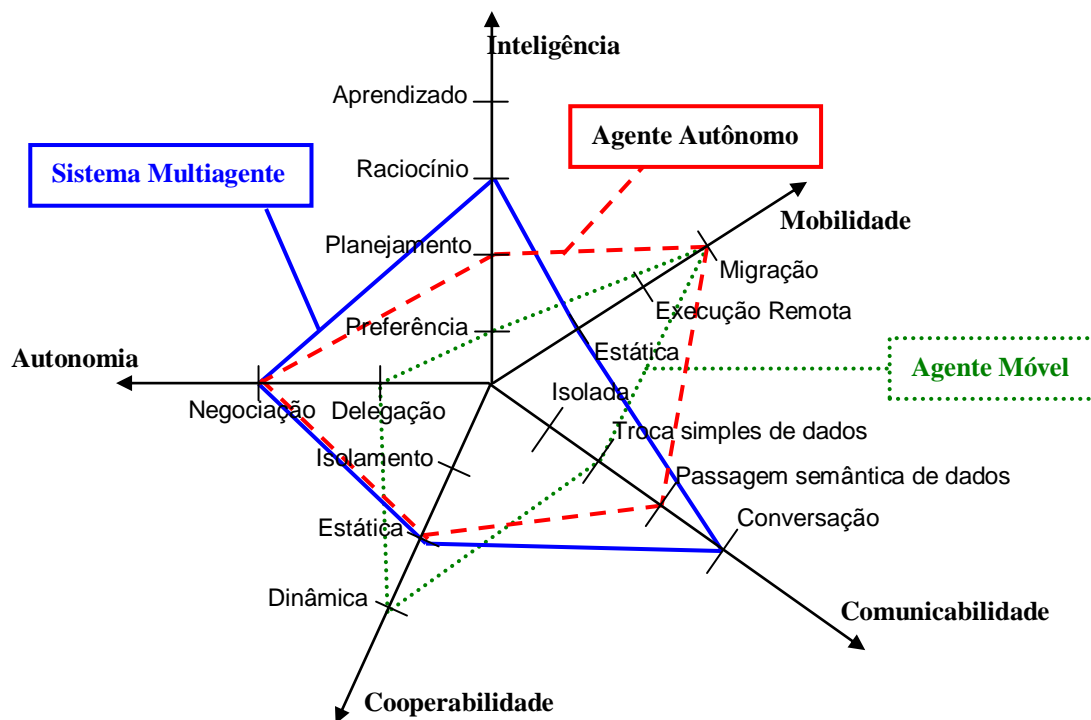


Figura 1. Níveis de habilidades de agentes exibidos por diferentes tipos de agentes (adaptado) [20].

De acordo com os exemplos de tipos de agentes apresentados na figura 1 tem-se que o Sistema Multiagente (linha contínua) possui as capacidades: de negociação em termos de autonomia; estática em termos de cooperabilidade; de conversação em termos de comunicabilidade; estática em termos de mobilidade; e de raciocínio em termos de inteligência.

O agente autônomo (linha tracejada) apresenta: autonomia de negociação; cooperabilidade estática; comunicabilidade de conversação; mobilidade estática; e inteligência de raciocínio. O exemplo de agente móvel (linha pontilhada) denota: autonomia de delegação; cooperabilidade dinâmica; comunicabilidade por troca simples de dados; mobilidade de migração; e inteligência por referência.

Importante destacar que a visão da complexidade relativa entre os agentes e/ou os SMA's, identificada pela classificação acima, possui relação direta com o propósito a que estes se destinam e/ ou são implementados, seguindo o padrão das arquiteturas existentes. Além da identificação da capacidade de cada agente e sua complexidade, é importante que seja identificada a arquitetura em que se baseia, para que se possa conhecer as suas potencialidades e limitações.

2.3. ARQUITETURAS DE AGENTES

As arquiteturas de agentes são divididas em duas classes principais: arquitetura deliberativa (ou cognitiva) e arquitetura reativa. Da junção das duas classes, surge a arquitetura híbrida, a qual encerra características de ambas [18].

2.3.1. Arquitetura Deliberativa

A arquitetura deliberativa ou cognitiva está ligada à própria origem da idéia de agentes pela Inteligência Artificial (idos dos anos 80 e 90). Ela se baseia na hipótese de um sistema físico-simbólico (*physical-symbol system hypothesis*), que se sustenta na representação simbólica de entidades físicas, e do relacionamento entre estas, formando estruturas, que são capazes de serem operadas por um conjunto de instruções simbolicamente codificadas [15].

Esta abordagem inspirou a idéia de um autômato de processamento sentencial (*sentential processing automaton*) ou agente deliberativo (*deliberative agent*) contendo uma representação explícita (modelo simbólico do mundo/ambiente) cujas decisões são tomadas por raciocínio lógico (ou pseudo-lógico), baseadas em padrões de identificação e reconhecimento simbólico, como se fosse parte de um provador de teoremas.

Entretanto, a idéia de representar simbólica e explicitamente o conhecimento e o raciocínio ocasionou uma série de problemas ligados a tempos de resposta para soluções críticas e de tempo real e, em resposta a estes, uma variada gama de abordagens foi apresentada. Contudo, ainda persistem os seguintes problemas em aplicações críticas: a tradução do mundo real em uma descrição simbólica precisa e adequada; e o raciocínio de entidades complexas do mundo real e os processos de interação entre estas.

Para solucionar os problemas vistos acima foi idealizada a arquitetura reativa, que se contrapõe conceitualmente à arquitetura deliberativa, como apresentado a seguir.

2.3.2. Arquitetura Reativa

A arquitetura reativa é conhecida como uma abordagem alternativa que busca resolver questões não solucionadas na arquitetura deliberativa, em termos de

tempos de resposta em sistemas críticos. Para isso, não apresenta nenhum tipo de representação simbólica centralizada do mundo, nem se utiliza de raciocínio simbólico complexo.

Esta arquitetura foi idealizada por Rodney Brooks [21] e está apoiada na sua idéia de *subsumption architecture*, a qual apresenta uma hierarquia de comportamentos, que define camadas mais primitivas e menos primitivas de acordo com o tipo de tarefa realizada em cada uma. As idéias por trás desta arquitetura são de que a inteligência real está no mundo, não em provadores de teoremas ou sistemas especialistas e que o comportamento inteligente surge da interação do agente com o ambiente que o cerca [15].

O modelo de funcionamento de um agente reativo é o “estímulo-resposta” [22]. Cada estímulo possui uma resposta mais adequada (comportamento) e os diversos tipos de comportamentos competem entre si pelo controle do agente, onde as camadas mais primitivas têm precedência sobre as camadas menos primitivas. Do mesmo modo, não possuem uma representação explícita e detalhada do ambiente, nem do raciocínio sob si mesmo (ou dos demais agentes, caso existam) e nem uma capacidade de comunicação (comunicabilidade) de mais alto nível entre estes agentes ou outros tipos, considerando as dimensões apresentadas na seção 2.2.

Logo, sendo mais simples, resultam num menor esforço computacional e implicam em tempos de respostas melhores que os agentes cognitivos, em determinados casos, onde a resposta ao ambiente é crítica. No entanto, ela perde capacidade proativa e de planejamento [15].

2.3.3. Arquitetura Híbrida

Fruto da junção das características das arquiteturas deliberativas e reativas, a arquitetura híbrida busca por uma solução intermediária que una a capacidade de reagir mais rapidamente (quando restrições temporais forem mais exigidas) e a capacidade de planejar e ser mais proativo dentro da conjuntura que se apresenta o ambiente e o próprio estado interno do agente [15].

Desta forma, o agente híbrido é dividido em, pelo menos, dois subsistemas: um deliberativo, que contém um modelo simbólico do mundo, responsável pelo planejamento das ações e tomada de decisões; e outro reativo,

capaz de reagir ao ambiente sem se engajar em raciocínios complexos. Normalmente, o componente reativo tem algum tipo de prioridade sobre o componente deliberativo para poder responder rapidamente a qualquer evento importante do ambiente.

Logo, a arquitetura híbrida apresenta uma idéia natural de arquitetura hierárquica em camadas. As camadas ou subsistemas deliberativos e reativos podem, explicitamente, serem separadas por uma camada/ subsistema de controle que serve como um filtro que separa o que deve ser, ou não, encaminhado para deliberação por uma camada de um nível superior. Daí surge o problema chave desta arquitetura, que é a definição do tipo de controle que deve ser adicionado aos subsistemas para gerenciar a interação entre as camadas.

Dentre os subtipos ou diferentes implementações de arquitetura híbrida destaca-se a do agente autônomo concorrente [23]. Esta arquitetura é a utilizada para implementação dos agentes da equipe Mecateam[24], que é objeto de estudo deste trabalho.

2.3.4. Agente Autônomo Concorrente

O agente autônomo concorrente é um agente de arquitetura híbrida que se baseia no modelo genérico para agente autônomo com tomada de decisão descentralizada proposto em [25]. Seguindo este modelo, o agente apresenta três níveis decisórios: o reativo, o instintivo e o cognitivo (vide figura 2). No agente autônomo concorrente, esses níveis são implementados de forma concorrente, onde três processos são responsáveis pelos três níveis decisórios, respectivamente: *interface*, *coordinator* e *expert*. Estes processos, apesar de trabalharem paralelamente, estão estruturados de forma hierárquica, estando o cognitivo no nível superior, o instintivo no nível intermediário e o reativo no nível mais baixo.

O nível cognitivo define o que deve ser feito, segundo o estado corrente do ambiente e do próprio agente. Esta definição se baseia na associação de cada estado corrente do ambiente a um planejamento de ações para aquele estado, visando alcançar certa meta em um estado futuro. O nível instintivo é quem escolhe o comportamento a ser selecionado e determina uma ação a ser atribuída ao agente no nível reativo, que, por sua vez, executa tais ações.

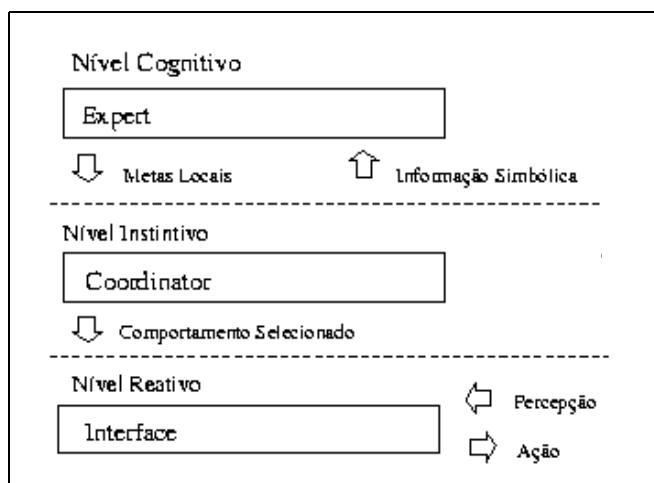


Figura 2. Representação esquemática da arquitetura de um agente autônomo concorrente de acordo com o fluxo de informações [24].

Esta proposta de agente com arquitetura híbrida, já utilizada em propostas de controle de fluxo de energia elétrica [22], vem sendo utilizada na implementação de agentes de futebol de robôs da equipe de futebol de robôs Mecateam da UFBA desde 2003/2004.

2.4. PLANEJAMENTO E PLANOS

O planejamento pode ser descrito como a tarefa de elaboração ou seleção de um ou mais planos para alcançar um ou mais objetivos. O plano, por sua vez, pode ser definido como uma seqüência de ações a fim de se atingir uma determinada meta.

Existem diversos algoritmos de planejamento diferentes, que podem ser divididos em dois grupos: planejamento clássico, para ambientes observáveis e deterministas, e planejamento não clássico, para ambientes parcialmente observáveis [26]. Devido ao ambiente dinâmico no qual o futebol de robôs está inserido, este requer a utilização de uma forma de planejamento não-clássica, além do planejamento ser multiagente.

O planejamento multiagente consiste num planejamento onde um agente reconhece outros agentes não somente como integrantes do ambiente, mas também como entidades que respondem ou reagem às ações deste. Assim, o planejamento deve considerar a elaboração ou seleção de planos para todos os agentes e a coordenação entre eles. Essa coordenação pode ser feita através de comunicação

ou através de convenções, como determinar uma relação de dependência fixa das ações entre um agente e outro(s) [25].

Diversas formas de planos podem ser aplicadas em um agente, a depender da situação. Como exemplo, essa possibilidade de escolha de plano poderia ser utilizada num determinado cenário onde a criticidade da resposta demandaria a execução de uma ação imediata (mais reativa), em detrimento de uma ação proveniente de um planejamento deliberativo (cognitivo) ou, ainda, planos resultantes de um planejamento não determinista, em que dentre uma coleção conhecida, os planos seriam escolhidos aleatoriamente [22].

Para selecionar seus planos um agente deve possuir um mecanismo de decisão que permita que ele possa atuar considerando seus objetivos e um perfil de atuação no ambiente onde está atuando [27]. A implementação de planos e os tipos de planejamento para a utilização dos mesmos estão ligadas ao tipo e arquitetura do agente, ao ambiente em que este vai atuar e ao propósito que se destina. Os planos de um agente reativo (físico), por exemplo, podem estar totalmente baseados em circuitos eletro-eletrônicos [21,22], enquanto, em um extremo oposto, um SMA (virtual) pode apresentar os seus planos em uma linguagem de descrição de planos em que se abstrai o tratamento e relacionamento com o mundo físico [22]. Entre estes extremos podem ainda existir algumas implementações intermediárias.

A título de exemplo de um planejamento de um robô físico reativo, vamos considerar o planejamento utilizado no agente autônomo apresentado em [28]. Este agente, um robô físico terrestre, possui planos que fazem com que o mesmo identifique a entrada de um labirinto, percorra o labirinto, encontre a saída deste e, após estar com seu corpo totalmente fora do labirinto, pare. O agente é dotado de dois sensores de toque (à esquerda e à direita), para identificar toques nas paredes do labirinto e poder manobrar, e um sensor de luz para identificar a entrada e a saída do labirinto.

O labirinto (ambiente), por sua vez, possui uma entrada e uma saída distintas entre si podendo ter qualquer tipo de divisão interna, não possuindo nenhum outro agente atuando no seu interior. A entrada e a saída do labirinto possuem duas tarjas (cinza e preta, respectivamente) fixadas ao piso para serem detectadas pelo sensor de luz.

O plano geral do agente para cumprir o seu objetivo pode ser representado pelo algoritmo apresentado no quadro 1.

De acordo com o algoritmo em questão, a atuação do agente é de, após entrar no labirinto, percorrer o mesmo até encontrar a saída. Caso o agente se depare com a entrada, durante a busca, ele deve executar meia-volta e retomar a busca pela saída do labirinto. O plano do agente pode ser expresso pelo autômato da figura 5.

```

1 Iniciar
2 Entrar;
3 Se Sensor de Luz == Cinza então
4 Situação = Dentro do labirinto;
5 Faça
6 Procurar saída;
7 Se Sensor de Luz == Cinza e Situação == Dentro do labirinto então
8 Meia volta;
9 Procurar saída;
10 Fim Se
11 Se Sensor de Luz == Preto e Situação == Dentro do labirinto então
12 Situação = Saída encontrada;
13 Fim Se
14 Enquanto Situação <> Saída encontrada;
15 Sair;
16 Parar;
17 Fim Se
18 Fim

```

Quadro 1. Algoritmo do plano do agente autônomo de [28].

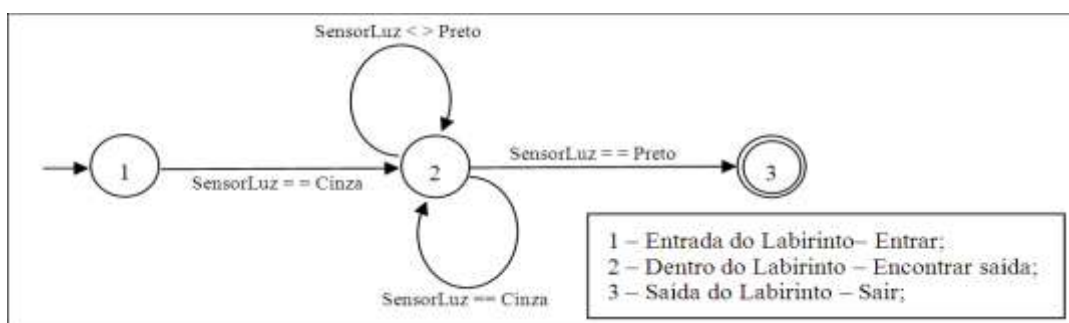


Figura 3. Autômato do plano do agente autônomo reativo de [28].

O plano geral do agente se relaciona com um conjunto de ações para cada estado do agente. Como exemplo, no estado 2 (agente dentro do labirinto), apresentado pelo autômato da figura 6, existe um conjunto de ações para se atingir o objetivo deste estágio que é encontrar a saída do labirinto.

Este conjunto de ações visa apresentar algum tipo de solução para os estímulos apresentados pelo ambiente, fazendo com que o agente responda de maneira mais apropriada a cada estímulo, o que, em conjunto permitirá ao robô atingir o objetivo.

A implementação do conjunto de ações pode ser observada no quadro 2.


```

1 task ChecaSensorLuz()
2 {
3   while (true) {
4     if (SENSOR_2 < valorLimiteSupPreto) { // Se sensor de luz encontrar faixa preta=>saida do labirinto
5       stop MovimentaRobo;
6       Off(OUT_A+OUT_B); // Pára o robô
7       OnFwd(OUT_A+OUT_B); // Reinicia movimento para frente
8       SomFimLabirinto(); // Emite som característico p/ identificar termino do labirinto
9       Off(OUT_A+OUT_B); // Pára o robô em definitivo
10      stop ChecaSensorLuz; /* Neste ponto o robo terminou sua missão e saiu do labirinto
11                          parando após a faixa preta*/
12    }
13
14    if (SENSOR_2 > valorLimiteInfCinza) { // Se sensor de luz encontrar faixa Cinza=>inicio do labirinto
15      if (travalnicioLabirinto == 0) { // Se for a primeira vez que identifica faixa Cinza(inicio)
16        SomInicioLabirinto(); // Emite som característico para identificar inicio do labirinto
17        start MovimentaRobo; // Inicia movimentacao para sair do labirinto
18        travalnicioLabirinto = 1; // Muda valor do bloqueio para nao mais entrar nesta condicao
19      }
20      else { // Se encontrar faixa Cinza(inicio) pela segunda vez ou N vezes
21        stop MovimentaRobo; // Para movimentacao pra retornar ao labirinto
22        Estrategia3(); // Mude para estratégia 3
23        start MovimentaRobo; // Retorna a movimentacao para buscar a saida
24      }
25    }
26  }
27 }

```

Quadro 2. Implementação do plano do agente autônomo de [28] na linguagem NQC.

No plano do robô este deve sempre seguir em frente até encontrar a saída. Caso o robô encontre o marcador do fim do labirinto (faixa preta no chão), através do sensor de luz, deverá se preparar para sair do mesmo. Se ainda não saiu do labirinto e for percebido acionamento dos sensores de toque, significa que o agente colidiu com a parede do labirinto e deve tomar as ações para desfazer a rota de colisão e voltar a seguir em frente.

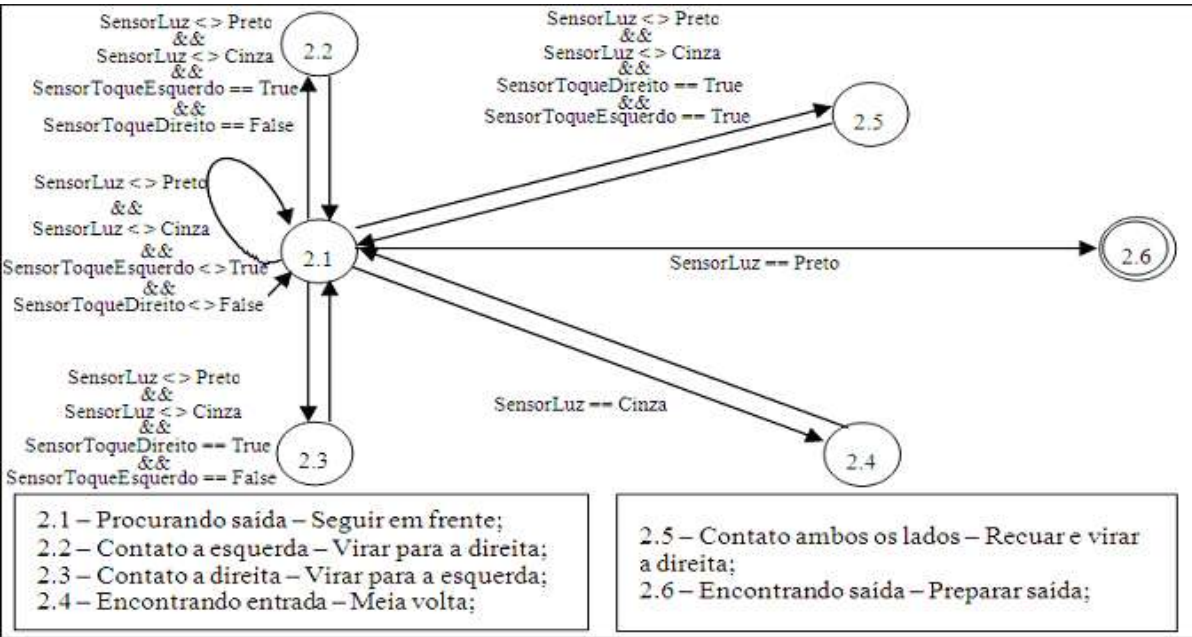


Figura 4. Autômato do conjunto de ações do agente autônomo reativo de [28].

Por outro lado, se o agente encontrar o marcador de início do labirinto (faixa cinza no chão), através do sensor de luz, o agente deverá retornar ao interior do labirinto e voltar a procurar a saída. O conjunto de ações do robô estão representados pelo autômato da figura 6.

3. MÉTODOS FORMAIS

Nos dias atuais, o crescimento do tamanho e da complexidade dos sistemas propicia o surgimento de erros no desenvolvimento que acarretam em falhas nas aplicações que podem ocasionar prejuízos financeiros, perda de tempo e, em casos de sistemas críticos, perdas irreparáveis [2]. A engenharia de software almeja o desenvolvimento de sistemas confiáveis, independentemente do seu tamanho ou complexidade. Os métodos formais contribuem com este intento, através da utilização de modelos matemáticos, linguagens formais, técnicas e ferramentas para especificação e verificação de sistemas.

A especificação formal consiste na descrição de um sistema e de suas propriedades, utilizando-se de sintaxes e de semânticas bem definidas para propiciar uma modelagem rigorosa e consistente de um sistema [2,29]. A verificação formal consiste em provar que a descrição formal do sistema satisfaz as propriedades do mesmo, ou seja, que o sistema é correto em relação às suas propriedades.

Dentre as técnicas de especificação e verificação formal de sistemas, os verificadores de modelos [6] têm sido cada vez mais utilizados, particularmente em sistemas concorrentes e reativos. A utilização efetiva destes verificadores se deve ao fato dos mesmos envolverem técnicas automáticas para a verificação de propriedades.

3.1. VERIFICAÇÃO DE MODELOS

A verificação de modelos consiste na construção de um modelo finito (um autômato ou uma variação deste tipo de representação) de um sistema e na aplicação de um algoritmo de verificação de propriedades (normalmente especificadas utilizando uma linguagem lógica temporal), que varre exaustivamente o espaço de estados do modelo [2].

Na prática, um sistema pode ser descrito por um conjunto de autômatos dos seus componentes, obtidos pela decomposição do sistema em questão e na sincronização / relacionamento destes componentes.

A aplicação do algoritmo de verificação consiste em dados um modelo M e uma propriedade Φ , verificar se M satisfaz Φ .

Dentre as lógicas temporais utilizadas para especificação de propriedades a CTL (*Computation Tree Logic*) é utilizada por muitos verificadores de modelos, e o verificador de modelos utilizado neste trabalho utiliza um dialeto desta linguagem. CTL considera um modelo de tempo ramificado, ou seja, a partir de um certo instante poderá haver futuros distintos. Existem linguagens que tratam o tempo com uma única linha de tempo, como por exemplo, a linguagem LTL (*Linear Temporal Logic*) [2,30].

CTL contém quantificadores de caminho para referenciar um ou mais futuros (E) ou em todos os futuros (A) possíveis. Utiliza operadores modais para expressar propriedades sobre o tempo, como em um estado no futuro vale φ ($F\varphi$) ou em todos os estados possíveis vale φ ($G\varphi$), entre outros, sem especificar unidades de tempo de forma explícita.

Os verificadores de modelos possuem limitações, como, por exemplo, os seus modelos serem finitos e também a possibilidade de explosão de estados, que é a principal desvantagem destes verificadores [2]. Na prática, a ordem de grandeza dos sistemas verificados pode, com essas técnicas, atingir valores da ordem de 10^{1300} estados alcançáveis [2]. Algumas técnicas como OBDD's (*Ordered Binary Decision Diagram*), identificação de informações de ordem parcial e minimização semântica são utilizadas para representar eficientemente sistemas de transição de estados e eliminar estados desnecessários [2], reduzindo consideravelmente o espaço de estados do modelo, aumentando a extensibilidade dos sistemas a serem verificados. No caso de sistemas em que o espaço de estados explode mesmo com o uso de otimizações, pode-se contornar o problema considerando características do domínio específico da aplicação, e buscar por “boas” modelagens, utilizando abstrações no modelo. Além disso, forma utilizadas técnicas baseadas na Verificação Composicional de Modelos (*Compositional Model Checking*), a ser descrita na próxima seção, podem ser aplicadas [31].

3.1.1. O Verificador de Modelos UPPAAL

O UPPAAL é um conjunto de ferramentas para verificação formal de sistemas de tempo real que podem ser modelados como uma rede de autômatos com tempo [11]. Desde o lançamento da primeira versão, por volta do ano de 1995,

após o seu desenvolvimento em conjunto pelas Universidades de Uppsala e Aalborg (cujo acrônimo dá nome à ferramenta, ou seja, **UPP**sala + **AAL**borg = **UPPAAL**), esta ferramenta vem sendo constantemente evoluída. Dentre os experimentos e implementações que permitiram a evolução e maturidade do UPPAAL podem ser destacados: redução simétrica e de ordem parcial de estados; alcançabilidade de custo mínimo, técnicas de aceleração de busca de estados e reduções de estruturas de dados e de utilização de memória [32].

O conjunto de ferramentas do UPPAAL possui um ambiente de modelagem que utiliza uma linguagem que estende o formalismo dos autômatos com tempo (vide figura 5), um simulador que serve para acompanhar interativamente a execução do sistema (vide figura 6) e um verificador de propriedades (vide figura 7) que utiliza uma linguagem de consulta baseada em um subconjunto da linguagem *CTL* (*Computation Tree Logic*) chamada *TCTL* (*Timed Computation Tree Logic*) [11,32].

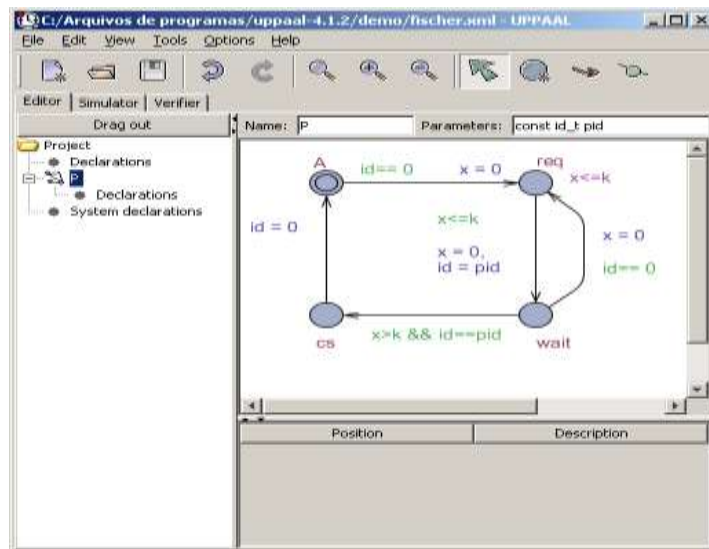


Figura 5. Interface do editor do UPPAAL.

As imagens apresentadas como exemplo são de modelos que constam como exemplos na instalação da ferramenta que, neste caso particular, foi o protocolo de exclusão mútua de Fischer. Na figura 8 consta o modelo de um processo e na figura 9 a instanciação de 05 (cinco) processos que executam em paralelo. A figura 10 apresenta a verificação da propriedade de não ocorrência de *deadlock*. Quando uma propriedade não é verificada, o verificador apresenta um

contra-exemplo, que é uma parte do modelo na qual a propriedade falha, conforme visto na figura 8.

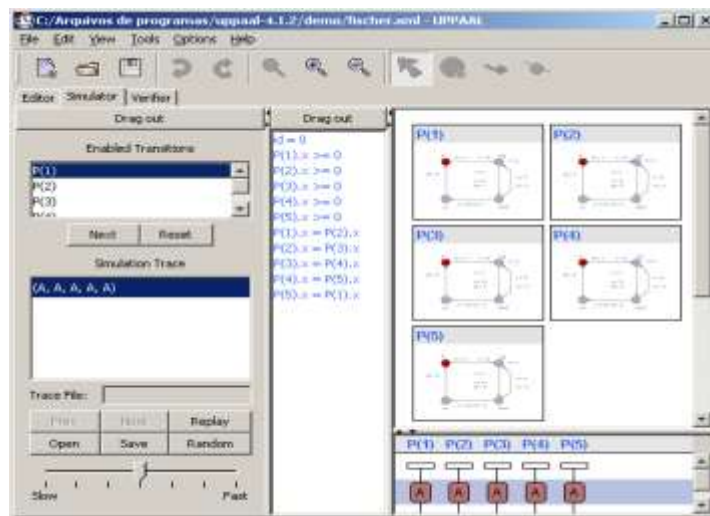


Figura 6. Interface do simulador do UPPAAL.

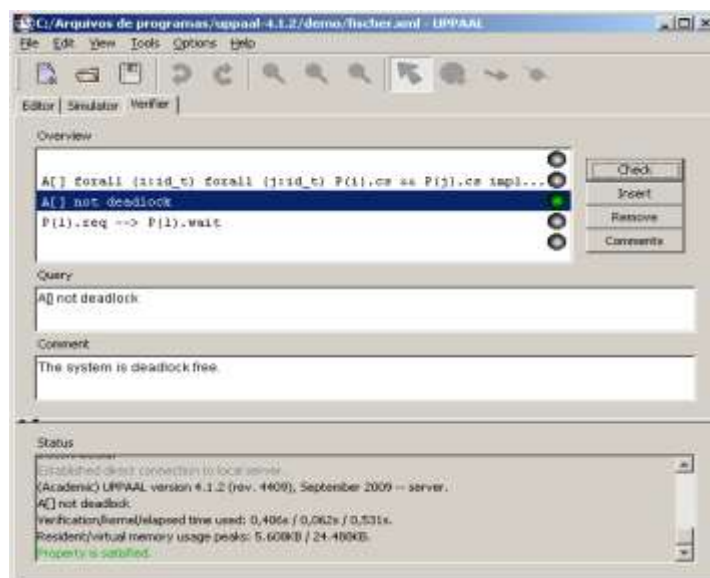


Figura 7. Interface do verificador do UPPAAL.

Por ser um subconjunto de CTL a TCTL apresenta características próprias na sua especificação de propriedades. Dentre estas características podem ser citadas:

- a) operadores lógicos: and, or, imply, not;
- b) operadores temporais¹: [], <>, →;
- c) operadores de caminho: A, E;

¹ TCTL não tem operador Next pois o tempo é denso.

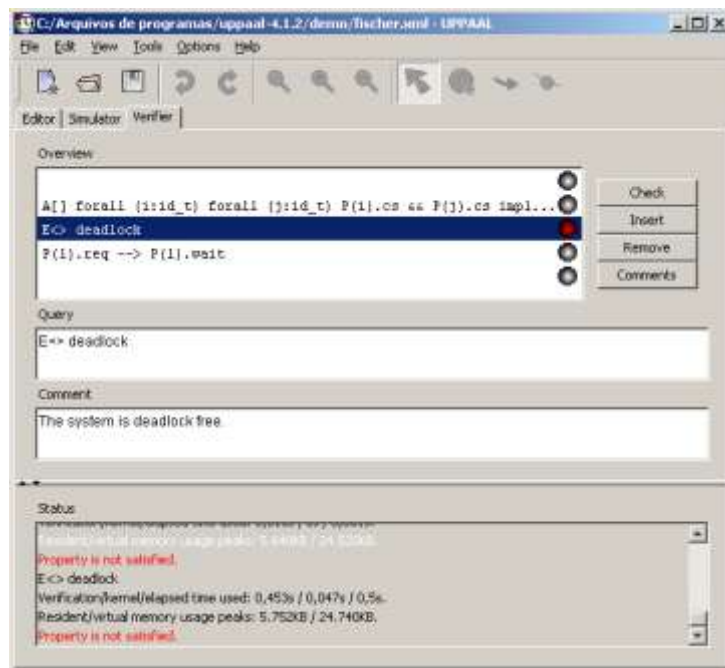


Figura 8. Interface do verificador do UPPAAL com resultado de propriedade não satisfeita.

Considerando os operadores podem ser apresentadas as seguintes fórmulas válidas: $E\langle\rangle p$, $E[] p$, $A\langle\rangle p$, $A[] p$ e $p \rightarrow q$. As sentenças p e q das fórmulas podem ser: *Processo.estado* | *clock* ² *valor* | p or q | p and q | not p | p imply q | *deadlock*.

A tabela 1 apresenta a semântica dos operadores:

Tabela 1. Nomes, propriedades, descrição e equivalência entre TCTL.

Nome	Propriedade	Descrição	Equivale a
Possivelmente	$E\langle\rangle p$	Existe ao menos um caminho onde p futuramente acontece.	
Invariavelmente	$A[] p$	Para todo caminho p sempre acontece.	not $E\langle\rangle$ not p
Potencialmente sempre	$E[] p$	Existe ao menos um caminho onde p sempre acontece.	
Futuramente	$A\langle\rangle p$	Para todo caminho p futuramente acontece.	not $E[]$ not p
Leva a	$p \rightarrow q$	Uma vez que p é verdade q será verdade futuramente.	$A[] (p \text{ imply } A\langle\rangle q)$

² ~ representa relações binárias do tipo: $<$, \leq , $=$, $>$, \geq .

3.1.2. Verificação Composicional de Modelos

A verificação composicional de modelos (*Compositional Model Checking*) pode ser definida como a utilização de técnicas para a redução da complexidade da verificação de modelos em sistemas compostos de vários processos paralelos [31], com o objetivo de prevenir a potencial explosão de estados inerentes à verificação deste tipo de sistema [33].

Uma abordagem óbvia para este tipo de problema é a decomposição natural do sistema [33]. Neste caso, a redução de complexidade dos modelos se baseia, principalmente, na representação e verificação individual e isolada de cada componente do sistema para, posteriormente, inferir-se a correção das propriedades globais do mesmo [31, 33].

A verificação composicional de modelos considera duas abordagens para especificação que se aplicam aos sistemas dos seguintes tipos: complexos e paralelizados e/ou hierárquicos e modulares.

Na abordagem de sistemas complexos e paralelizados, deve-se focar na definição das suas partes integrantes e nas relações entre estas, enquanto na abordagem de sistemas hierárquicos e modularizados foca-se na simplificação e ocultação de detalhes dos processos de baixo nível. As duas abordagens merecem atenção em relação à especificação do modelo, respectivamente, nos seguintes aspectos: saber se alguma propriedade de um processo componente permanece verdadeira numa composição paralela na qual esteja presente e garantir que a representação simplificada do modelo satisfaz as mesmas propriedades.

Algumas técnicas e abordagens vêm sendo propostas e aprimoradas para permitir que a verificação composicional de modelos possa ser utilizada para os tipos de aplicações citados anteriormente, conforme apresentado em [31,33,34].

Algumas destas técnicas são automáticas e quase que totalmente transparentes para o usuário. Outras técnicas exigem a intervenção maior do usuário, mas, em compensação, apresentam, normalmente, melhores resultados que as técnicas automáticas. A utilização de uma técnica automática ou não automática depende da aplicação e da sua complexidade e do nível de conhecimento do usuário sobre verificação de modelos e na aplicação das respectivas técnicas [33]. Muitas dessas técnicas já foram usadas em sistemas reais com grande sucesso [31,33,34].

Entre as técnicas automáticas podem ser citadas como exemplo: *partitioned transition relations*, *lazy parallel composition* e *interface processes* [33].

As técnicas *partitioned transition relations* e *lazy parallel composition* provêem uma forma de computar os estados predecessores ou sucessores de um conjunto de estados de um grafo, sem a necessidade de construir todas as relações das transições globais do sistema. Ambas são utilizadas quando não é conveniente a intervenção do usuário e utilizam a relação de transições de cada componente separadamente para, posteriormente, combinar os resultados individuais da verificação e apresentar o conjunto de estados do grafo global correspondente ao resultado da funcionalidade verificada [31,34].

A técnica *interface processes* é um outro exemplo de técnica automática. Esta se apóia na minimização do grafo global de transição de estados e foca na comunicação entre os processos. Esta abordagem abstrai detalhes de implementação de cada processo internamente, considerando somente as variáveis de interação entre os processos que se comunicam entre si, tornando o modelo bem menor [31].

Dentre as técnicas não automáticas cabe destacar a *Assume-guarantee reasoning*. Esta é uma técnica *top-down* que considera que o comportamento de cada componente depende do resto do ambiente ou do resto do sistema e, por esse motivo, o usuário deve especificar as propriedades que o ambiente deve satisfazer e assumi-las como verdadeiras, para, depois, garantir a correção do componente.

Conceitualmente, se as propriedades assumidas (*Assume*) são satisfeitas, os componentes deverão satisfazer outras propriedades denominadas garantias (*guarantee*). A combinação do conjunto de propriedades assumidas / garantidas possibilita a demonstração da correção de todo o sistema sem ter a necessidade de construir um grafo global de transição de estados [33].

Na prática, quando um componente é verificado pode ser necessário ASSUMIR (*assume*) que o ambiente se comporta de determinada maneira, mas só é possível saber se uma propriedade verificada é verdadeira para todo o sistema se outro(s) componente(s) do sistema GARANTIREM (*guarantee*) este comportamento.

Considerando que o futebol de robôs contém uma grande quantidade de processos e componentes do sistema que atuam concorrentemente, as técnicas de verificação composicional de modelos podem ser utilizadas para evitar a explosão de estados durante a verificação.

3.2. VERIFICAÇÃO DE AA's E SMA's

A existência de ferramentas capazes de realizar automaticamente a verificação de propriedades [2] faz da verificação de modelos uma técnica bastante utilizada nas pesquisas ligadas a planejamentos em ambientes complexos e não deterministas para AA's e SMA's. Nesta seção apresentamos alguns trabalhos relacionados que consideramos relevantes para a nossa pesquisa.

Os trabalhos [4,5] baseiam-se na abordagem para a geração automática de planos, em ambientes não deterministas com visão total do ambiente, utilizando fórmulas em *Computation Tree Logic* (CTL), através de um algoritmo baseado em técnicas de verificadores de modelos. Para poder explorar um maior espaço de estados, o algoritmo implementado utilizou diagramas de decisão binários (*Ordered Binary Decision Diagram* - OBDD) [6] para uma representação mais compacta dos estados do modelo.

Em [7] é apresentada uma evolução em relação ao trabalho [4], considerando visão parcial do ambiente. Neste caso é apresentada uma nova abordagem para o desafio de planejar com este tipo de restrição em um ambiente dinâmico e não determinista. Para tanto, foi implementado um algoritmo que procura por formações de estados e/ou grafos cíclicos que tenham sido gerados por ações do ambiente. O algoritmo gera planos seqüenciais acíclicos, a despeito da incerteza do estado inicial e do quão incerto seja o resultado dos planos gerados. OBDD's também foram utilizados para minimizar o problema da explosão de estados.

Os trabalhos apresentados em [8,9] baseiam-se na validação dos planos multiagentes e controle de recursos para um ambiente de simulação de tática para aeronaves de combate, levando em conta a flexibilidade no comportamento destes planos. Para realizar a verificação são definidas restrições sobre os recursos individuais dos agentes, como o controle de combustível ou quantidade de munição e a propagação destas restrições para o planejamento do SMA. Para tanto foi gerado um modelo de planos de agentes e sistemas multiagentes para a plataforma do framework CIRCA (*Cooperative Intelligent Real-Time Control Architecture*). Este modelo foi posteriormente utilizado numa arquitetura de sistemas multiagentes SCALA (*Système Coopératif d'Agents Logiciels Autonomes*) proprietária da Dassault Aviation. Essa especificação foi realizada considerando, inicialmente, restrições

temporais e, posteriormente, restrições de recursos. Tais resultados foram aplicados na verificação de planos coletivos em simulações de vôos de aeronaves de combate que originalmente tinham um comportamento mais reativo do que cognitivo. Este ambiente, apesar das suas particularidades, possui desafios similares aos encontrados no futebol de robôs, quais sejam: ambiente altamente dinâmico, não-determinista e com visão parcial.

Em [10] foi utilizada uma representação com autômatos híbridos para especificar os planos individuais e coletivos com restrições temporais e restrições de recursos inerentes aos agentes. Desta forma, os planos multiagentes foram modelados como uma rede de autômatos híbridos, onde cada autômato híbrido representava o plano de um agente. Decorrente desta primeira parte do trabalho foi feita a integração do controle da execução dos planos e a verificação dos mesmos utilizando o verificador Hytec.

Em [11,12] é apresentado um estudo de verificação de planos para um agente autônomo utilizado nas missões da agência espacial americana (NASA - *National Aeronautics Space Administration*). Este trabalho apresenta resultados da verificação de um sistema controlador e agendador de planos chamado HSTS (*Heuristic Scheduling and Testbed System*) para o controle remoto do robô *Deep Space 1*. Foi utilizada a ferramenta UPPAAL para especificar os planos do HSTS e as restrições do domínio e verificar as propriedades necessárias ao cumprimento dos objetivos do agente.

Já em [13] são demonstradas as possibilidades de uso de autômatos híbridos para a verificação de planos de agentes autônomos. Neste trabalho é apresentada a verificação de planos de um agente autônomo aéreo não tripulado de reconhecimento, utilizando o verificador para autômatos híbridos PHAVer. Os planos consideram características de dinâmica da aeronave sob interferência de: condições meteorológicas previstas previamente, condições internas da aeronave e condições externas e demais restrições de segurança (terreno, rota, altitudes, etc.). A solução em questão integra a ferramenta MATLAB como gerador de scripts de planos para serem analisados pelo verificador.

No trabalho [14], foram utilizadas ferramentas de verificação formal e de simulação para a validação de equipes de futebol de robôs. Foram utilizados a linguagem de especificação ERLANG e o verificador McERLANG, ambos de

propriedade da Ericsson Company para verificar propriedades específicas do futebol de robôs em tempo de execução.

Os jogadores foram modelados e implementados numa arquitetura composta por 04 (quatro) processos em 02 (dois) níveis, conforme a figura 07.

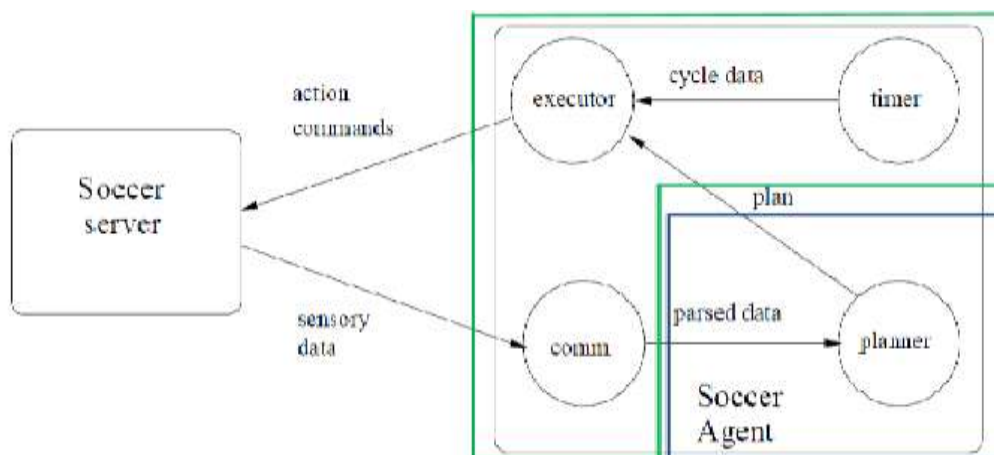


Figura 9. Arquitetura dos agentes utilizados em [14].

Os quatro processos que formam cada jogador são: *planner*, *executor*, *communicator* e *timer*. Fazendo uma comparação com os tipos de arquiteturas de agentes autônomos (Deliberativa, Reativa e Híbrida), o processo *planner* (planejador) está ligado à capacidade cognitiva dos jogadores e os demais processos dizem respeito às interações de percepção, de atuação e de restrições temporais com o ambiente, neste caso, o Soccerserver.

Para não ter que implementar o Soccerserver em Erlang (o que exigiria um esforço muito grande) foi utilizada a simulação de jogos no próprio Soccerserver, fazendo com que os jogadores desenvolvidos em Erlang fossem executados sob o monitoramento do verificador de modelos e se comunicassem com o simulador, em partidas contra outras equipes, utilizando o protocolo UDP. Considerando que cada jogador possui somente uma visão parcial do ambiente e o verificador de modelos não teria acesso ao Soccerserver, foi utilizada uma solução instanciando um agente treinador no simulador para participar da simulação. Isto se deve ao fato dos treinadores terem visão total do jogo e, com as suas informações, pode-se comparar a visão dos jogadores com a do ambiente.

Em relação aos trabalhos apresentados acima, algumas considerações são apresentadas a seguir. Nos trabalhos [11,12], devido à complexidade de algumas regras e restrições, não foi possível especificar completamente todos os

planos para serem verificados, fazendo com que os autores sinalizassem a necessidade de uso de técnicas de abstrações para poder realizar tal tarefa.

Em [13], os agentes são modelados para condições quase sempre deterministas e não dinâmicas. Os estudos em [8,9] foram aplicados na verificação de planos coletivos em simulações de vôo de aeronaves de combate que originalmente tinham um comportamento mais reativo do que cognitivo que utilizaram arquiteturas proprietárias de SMA's.

Em [14] a solução adotada utilizou a linguagem Erlang de propriedade da Ericsson, fazendo com que os agentes tivessem que ser recodificados nesta linguagem. Como a verificação foi realizada de forma combinada com simulação de jogos das equipes verificadas, não houve uma clara distinção entre verificação dos planos e verificação do código executável dos jogadores da equipe. Além disso, como era necessário que todos os jogadores fossem utilizados na simulação para que se pudesse proceder a verificação, não foi possível realizar a verificação de cada jogador da equipe individualmente.

Pelo fato de utilizar simulação combinada com verificação existe a possibilidade deste tipo de experimento não percorrer todo o espaço de estados do modelo, apesar de ter sido relatado que os autores realizaram muitas simulações com as mesmas equipes e mesmas condições do ambiente, para tentar minimizar tal inconveniente.

O nosso trabalho realiza a verificação dos planos de um sistema de futebol de robôs multiagente que tem um ambiente dinâmico e não determinista, com visão parcial. Neste sentido, ele tem alguns pontos semelhantes com alguns dos trabalhos apresentados acima, mas traz contribuições relativas ao uso de abstrações e modularização da verificação através do uso de técnicas de verificação composicional de modelos durante o processo de especificação e verificação o qual se baseia em um método evolutivo e incremental de verificação.

4. FUTEBOL DE ROBÔS

Dentre os mais variados tipos de emprego para agentes pode ser destacado, como ambiente de estudo e pesquisas, o Futebol de Robôs. Este ambiente pode ser utilizado como ferramenta para o desenvolvimento de soluções em Inteligência Artificial com possibilidades de reuso em diversas áreas do conhecimento.

O comportamento inteligente deste ambiente está associado à representação e interpretação do conhecimento individual e coletivo dos seus agentes que devem executar um conjunto de ações relativas a um planejamento. As questões relativas ao planejamento são essenciais num sistema multiagente e a correção dos planos definidos em um planejamento deve ser garantida para que o sistema atenda aos objetivos esperados.

Proposto pela *Robocup Federation* [24], o futebol de robôs (*Robocup*) surgiu como um laboratório para o estudo da Inteligência Artificial, em especial a Inteligência Artificial Distribuída (IAD) [26] e suas ramificações. O futebol de robôs serve de plataforma para a compreensão de uma grande variedade de problemas inerentes aos agentes autônomos, à robótica cooperativa e aos sistemas multiagentes. Os resultados das pesquisas de diversas fontes diferentes são apresentados e discutidos em campeonatos realizados em todo o mundo. Além disso, por ser um domínio complexo, apresenta-se como um rico laboratório para avaliação dos modelos propostos e a possibilidade de estudo de conceitos aplicáveis em outras áreas [26].

Em uma partida, os robôs têm que ser capazes de reconhecer o espaço onde estão (campo de futebol) e todas as suas referências, representar o ambiente, determinar objetivos, planejar e executar ações para atingir tais objetivos.

O futebol de robôs tem várias categorias, entre elas a de robôs físicos de diversos tamanhos e a de robôs simulados.

4.1. O AMBIENTE DO FUTEBOL DE ROBÔS SIMULADO

A *Robocup Federation* disponibiliza um simulador de futebol de robôs, que simula os robôs físicos, a bola e o campo com suas referências, chamado

Soccerserver [25]. O *Soccerserver* apresenta ainda um visualizador do jogo, o *Soccermonitor*, conforme mostra a figura 3.



Figura 10. Visão do jogo proporcionada pelo *Soccermonitor* [25].

O *Soccerserver* fornece a visão de cada robô em intervalos de tempo e espera o recebimento de comandos para os mesmos, permitindo o controle de cada um individualmente, já que o recebimento das mensagens é feito separadamente, com destinatário certo, através de *socket* via protocolos UDP/IP (vide figura 4). Isto determina uma implementação independente de linguagem, que deve possuir suporte a *socket*.

Assim, as partidas do futebol de robôs se caracterizam por uma arquitetura cliente-servidor. Os clientes são os agentes e o servidor é o *Soccerserver*.

Portanto, segundo este modelo, o simulador / servidor é responsável por representar o ambiente (campo de futebol, os jogadores (agentes), a bola), além de prover o controle sobre as movimentações efetuadas no ambiente, considerando, similarmente ao que acontece no futebol humano, questões como atrito, colisões, inércia, ruídos, etc. [25].

De acordo com este modelo, nenhum agente se comunica diretamente com outro agente sem a passagem pelo servidor. Assim, a depender da distância, do vento e do ruído existente na simulação do ambiente, a informação passada (por uma mensagem na forma de uma *string*) pode sofrer distorções ou chegar incompleta ao destinatário.

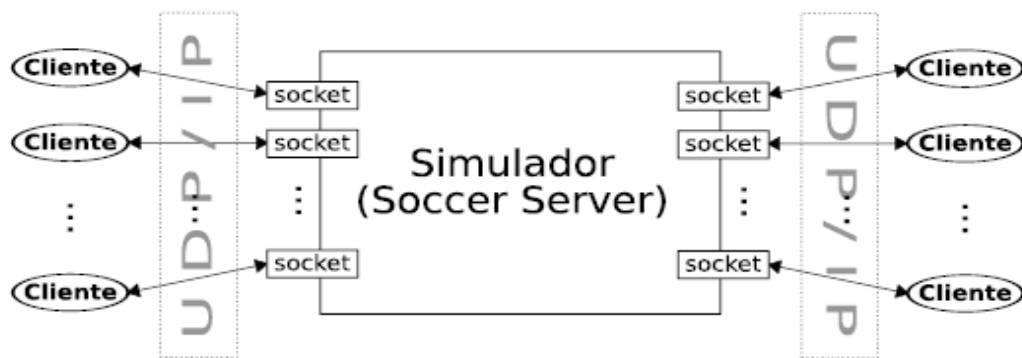


Figura 11. Modelo da comunicação entre os clientes (agentes) e o servidor (*soccerserver*) [25].

Os clientes (variando entre 6 a 11 agentes para cada equipe), por sua vez, percebem as informações do ambiente representadas pelo servidor através de capacidades perceptivas “visuais” e “auditivas”. Em contrapartida repassam ao servidor as ações que vão adotar a partir da interpretação da realidade apresentada, dos seus estados internos, do conhecimento anterior que possuem e dos objetivos que desejam atingir.

Estas ações dividem-se em primárias e concorrentes, sendo que, a cada ciclo, o agente pode executar uma única ação primária em conjunto com uma ou múltiplas ações concorrentes. As ações primárias são: chutar, acelerar, girar, agarrar a bola e mover. Já as ações concorrentes são: falar, girar o pescoço, mudar a visão e sentir o corpo (perceber seu estado interno).

Além disso, é importante destacar que as ações primárias e concorrentes são os comandos mais elementares entre os agentes e o ambiente. Desta forma, a utilização desses comandos de forma isolada, para um ambiente dinâmico, não determinista e parcialmente observável, que é o caso do futebol de robôs, fica aquém de um comportamento inteligente. Para tratar e apresentar respostas consistentes e inteligentes, nesse ambiente, o agente deve, inicialmente de forma individual e depois coletivamente, combinar uma seqüência de ações primárias associadas a um conjunto de ações concorrentes, ou seja, deve elaborar um planejamento que atenda aos objetivos individuais e do grupo.

4.2. MECATEAM

O Mecateam é um time de futebol de robôs baseado na arquitetura híbrida de agentes conhecida como agente autônomo concorrente. Esta arquitetura

é suportada pela biblioteca Expert-Coop [19,22], que em 2003 evoluiu para Expert-Coop++ conforme descrito em [24]. A equipe é resultado do esforço e desenvolvimento promovido por alunos da graduação em Ciência da Computação e do mestrado em Mecatrônica da Universidade Federal da Bahia e tem participado de competições nacionais de futebol de robôs nos últimos anos [24].

No Mecateam os agentes interagem com o *Soccerserver* para atuar de acordo com os planos gerados, os quais são especificados utilizando regras de produção. Nesta seção é apresentada a arquitetura do Mecateam e a representação dos planos deste time.

4.1.1. Sistema Baseado em Conhecimento

O sistema baseado em conhecimento do Mecateam (vide figura 12) é um sistema baseado em regras de produção (SBRP). Este é composto por uma base de conhecimento, formada por uma base de regras e uma base de fatos referente a cada agente (jogador), e um motor de inferência que avalia e emprega as regras de acordo com as informações da base de conhecimento (regras e fatos).

As regras de produção exemplificam uma forma de representação procedimental do conhecimento [35], baseado em lógica de primeira ordem. Esta representação se apresenta como uma coleção de sentenças do tipo situação-ação (se < condição(ões)> então <ação(ões)>). Cada condição de uma regra representa a informação na forma objeto-atributo-valor ou objeto-valor. As regras e fatos da base de conhecimento do Mecateam são especificadas de acordo com as regras de produção definidas pela BNF (*Backus-Naur Form*) do quadro 3.

A base de regras de produção formaliza a inteligência do agente e é responsável por prever os possíveis estados do ambiente e determinar qual o comportamento do agente em cada um destes estados [25].

A base de fatos é responsável por representar o estado do ambiente através das informações recebidas do nível instintivo (sejam informações do próprio agente ou de mensagens oriundas de outros agentes) e do motor de inferência.

A base de conhecimento é atualizada durante a execução do sistema através de informações resultantes das regras executadas ou do ambiente.

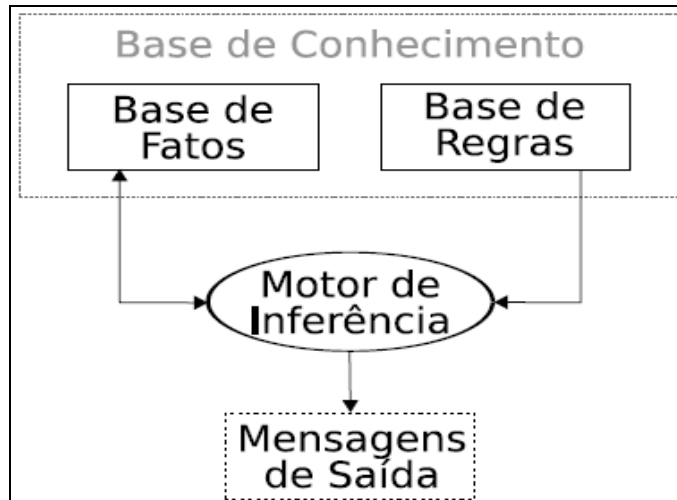
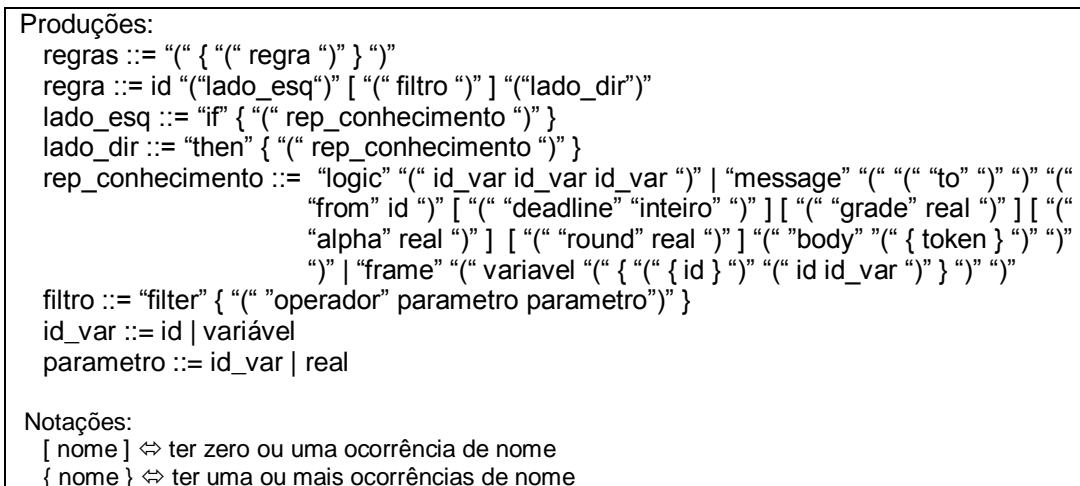


Figura 12. Arquitetura do sistema baseado em conhecimento utilizado no Mecateam[24].



Quadro 3. Representação das regras de produção em BNF [22].

O motor de inferência é o principal componente do sistema e funciona em ciclos divididos em três etapas [22]:

- a) Seleção das regras que satisfazem o estado atual do ambiente baseado no raciocínio tipo *forward chaining* (encadeamento progressivo):
 - o lado esquerdo da regra é comparado ao estado atual do ambiente;
 - pode se utilizar algum tipo de filtro para efetuar a comparação;
 - aceita a regra que não retornar falso para tal averiguação;
- b) Resolução de conflitos entre as regras selecionadas na etapa anterior, caso o conjunto de regras selecionadas não seja vazio e mais de uma regra seja selecionada;
- c) Execução das regras selecionadas e não conflitantes.

Após as três etapas é determinado o plano mais adequado a ser ativado pelo agente.

4.2.2. Representação dos Planos

Por ser composto de agentes baseados na arquitetura de agentes autônomos concorrentes, cada jogador é uma instância desta arquitetura e tem, como forma de representação do seu conhecimento, conjuntos de regras de produção nos níveis cognitivos e instintivos, que compõem os planos, conforme ilustram as figuras 13 e 14.

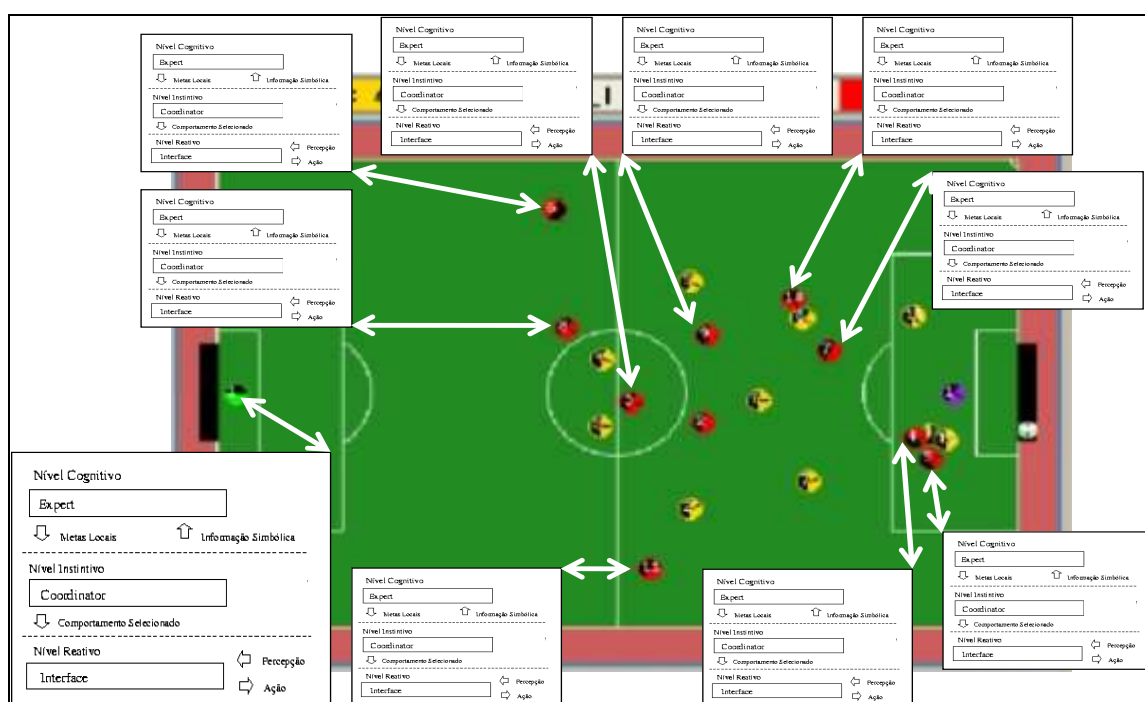


Figura 13. Representação da arquitetura de agente concorrente do Mecateam.

A figura 13 mostra que todos os jogadores do Mecateam possuem a mesma arquitetura, contudo as regras de produção dos planos cognitivos e instintivos dos agentes autônomos concorrentes são definidas de acordo com a função que cada jogador executa na equipe. A figura 14 apresenta que cada agente tem um plano cognitivo comum com os demais jogadores que executam o mesmo tipo de função (goleiro, defesa e meio de campo / ataque), mas pode possuir um plano instintivo diferente dos demais jogadores.

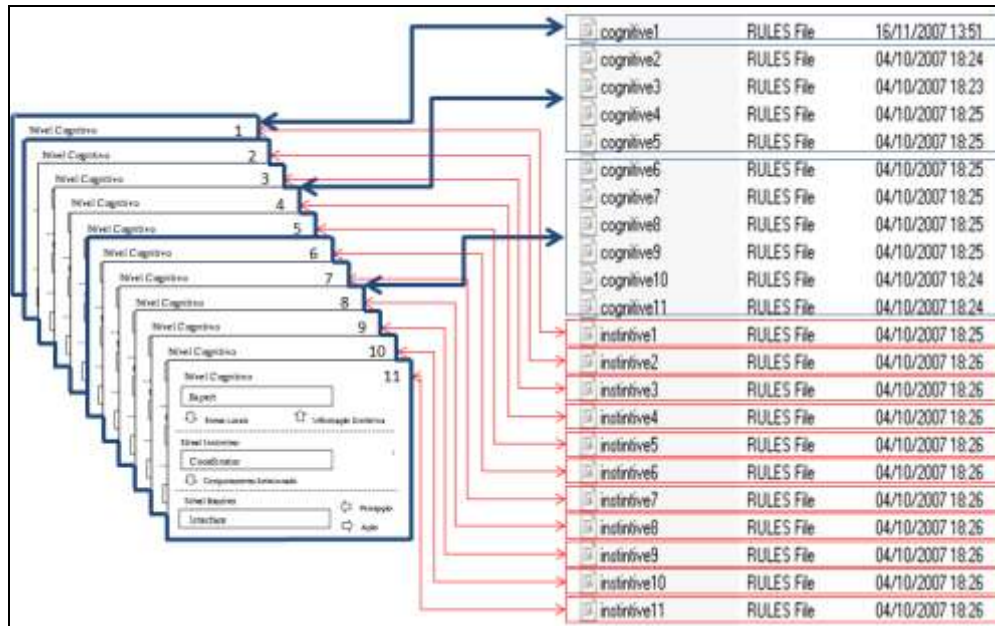


Figura 14. Relação entre planos cognitivos e instintivos e os agentes (jogadores) do Mecateam.

O nível reativo possui, de acordo com o que foi indicado na seção 3.1, diversas ações básicas (comandos de atuação compreendidos pela API do ambiente *Soccerserver*) que agrupadas dentro de uma sequência apropriada representam um determinado comportamento que o agente irá adotar. Desta forma, o nível reativo não possui nenhum tipo de comportamento decisório. O nível reativo percebe o ambiente e atua sobre este através de sequências de ações conhecidas como comportamentos.

Para permitir um melhor entendimento dos planos cognitivos e instintivos foi realizada uma análise do domínio relativa a estes planos e as respectivas regras que os compõem, com o objetivo de modelar a especificação formal destes planos para representar o mais fielmente estes componentes do sistema.

4.2.2.1. Planos Cognitivos

Os planos cognitivos são compostos por um conjunto de regras cognitivas que definem a mudança de estados dos agentes. Esta mudança decorre das informações recebidas da camada instintiva que, por sua vez, depende de informações recebidas da camada reativa. Os estados dos jogadores são representados pela variável *local_goal_current*, a qual depende do tipo de função executada por cada jogador, e pode possuir no caso mais genérico os seguintes

valores: *none*, *mark*, *side_attack* e *ending*. Os estados de cada jogador possuem um *status* associado. Este status é representado pela variável *local_goal status*, a qual pode possuir os seguintes valores: *none*, *active*, *achieved* e *fail*. A transição entre os estados acontece com a mudança do valor do *status* para cada estado.

Como os jogadores que executam funções similares na equipe possuem as mesmas regras cognitivas, ou seja, têm planos cognitivos iguais, os valores possíveis para a variável *local_goal current* de cada jogador depende da função ou grupo ao qual este pertence. O grupo dos jogadores de defesa (grupo composto pelos jogadores 2, 3, 4 e 5) e o grupo dos jogadores de meio de campo e ataque (grupo composto pelos jogadores 6, 7, 8, 9, 10 e 11) possuem regras idênticas entre os seus jogadores, haja vista que estes executam funções muito similares. Já o grupo goleiro possui uma regra distinta de todos os outros.

As diferenças entre os planos de cada grupo de agentes (jogadores) podem ser observadas através das regras cognitivas de cada um destes grupos, conforme apresentado nos quadros 4, 5 e 6.

<pre>((rule_0_start (if (logic (local_goal current none))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))) (rule_1_defense (if (logic (local_goal current mark)) (logic (local_goal status active))) (then (logic (local_goal current mark)) (logic (local_goal status active))))) (rule_2_defense (if (logic (local_goal current mark)) (logic (local_goal status achieved))) (then (logic (local_goal current side_attack)) (logic (local_goal status active)))))</pre>	<pre>(rule_3_attack (if (logic (local_goal current side_attack)) (logic (local_goal status active))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))) (rule_4_attack (if (logic (local_goal current side_attack)) (logic (local_goal status achieved))) (then (logic (local_goal current mark)) (logic (local_goal status active))))) (rule_5_defense (if (logic (local_goal current side_attack)) (logic (local_goal status fail))) (then (logic (local_goal current mark)) (logic (local_goal status active)))))</pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Quadro 4. Regras cognitivas dos jogadores de meio de campo e ataque.

Comparando os quadros 4, 5 e 6 é possível perceber as diferenças entre os planos cognitivos dos três grupos em termos de quantidades e em conteúdo de algumas regras. No que diz respeito à quantidade de regras, é tem-se que: o plano do goleiro contém 11 (onze) regras, os jogadores de defesa possuem 9 (nove) e os jogadores de meio de campo e de ataque têm 6 (seis). Esta diferença quantitativa se deve ao fato de algumas regras serem específicas de certos grupos como, por exemplo, as regras *rule_7_attack*, *rule_8_attack* e *rule_9_attack* são específicas do goleiro e as regras cognitivas *rule_1_defense*, *rule_3_defense*, *rule_4_defense* e *rule_5_defense* são específicas do goleiro e dos jogadores de defesa.

<pre> (rule_0_start (if (logic (local_goal current none))) (then (logic (local_goal current advance)) (logic (local_goal status active))))) (rule_1_defense (if (logic (local_goal current advance)) (logic (local_goal status fail))) (then (logic (local_goal current mark)) (logic (local_goal status active))))) (rule_2_defense (if (logic (local_goal current mark)) (logic (local_goal status active))) (then (logic (local_goal current mark)) (logic (local_goal status active))))) (rule_3_defense (if (logic (local_goal current mark)) (logic (local_goal status achieved))) (then (logic (local_goal current advance)) (logic (local_goal status active))))) (rule_4_defense (if (logic (local_goal current advance)) (logic (local_goal status active))) (then (logic (local_goal current advance)) (logic (local_goal status active))))) </pre>	<pre> (rule_5_defense (if (logic (local_goal current advance)) (logic (local_goal status achieved))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))) (rule_6_attack (if (logic (local_goal current side_attack)) (logic (local_goal status active))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))) (rule_7_attack (if (logic (local_goal current side_attack)) (logic (local_goal status achieved))) (then (logic (local_goal current mark)) (logic (local_goal status active))))) (rule_8_defense (if (logic (local_goal current side_attack)) (logic (local_goal status fail))) (then (logic (local_goal current mark)) (logic (local_goal status active))))) </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Quadro 5. Regras cognitivas dos jogadores de defesa.

Em termos de diferenças de conteúdo das regras, a tabela 2 mostra dois exemplos destas diferenças relativas aos grupos de jogadores. Assim, as regras *rule_0_start* diferem em cada um dos grupos e as regras *rule_3_defense* e *rule_2_defense* diferem nos grupos Goleiro e Jogadores de Defesa e no grupo Jogadores de Meio de Campo e Ataque. Estas diferenças se apresentam no lado direito da regra, mais precisamente no valor da variável que representa o objetivo local de cada jogador, a variável *local_goal current*. Nestes exemplos, as regras da coluna esquerda mudam o objetivo local de *none* para *advance*, enquanto as regras da coluna direita mudam o objetivo local para *side_attack*.

Apesar de existirem diferenças entre algumas regras de cada grupo de agentes, existem regras que são equivalentes em todos estes grupos. Mesmo que estas regras não possuam a mesma denominação, as mesmas possuem os mesmos lados esquerdo e direito, ou seja, conduzem de um mesmo estado atual para um mesmo estado futuro. A tabela 3 apresenta as regras equivalentes a todos os grupos de jogadores.

<pre>(rule_0_start (if (logic (local_goal current none))) (then (logic (local_goal current advance)) (logic (local_goal status active))))) (rule_1_defense (if (logic (local_goal current advance)) (logic (local_goal status fail))) (then (logic (local_goal current mark)) (logic (local_goal status active))))) (rule_2_defense (if (logic (local_goal current mark)) (logic (local_goal status active))) (then (logic (local_goal current mark)) (logic (local_goal status active))))) (rule_3_defense (if (logic (local_goal current mark)) (logic (local_goal status achieved))) (then (logic (local_goal current advance)) (logic (local_goal status active))))))</pre>	<pre>(rule_4_defense (if (logic (local_goal current advance)) (logic (local_goal status active))) (then (logic (local_goal current advance)) (logic (local_goal status active))))) (rule_5_defense (if (logic (local_goal current advance)) (logic (local_goal status achieved))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))) (rule_6_attack (if (logic (local_goal current side_attack)) (logic (local_goal status active))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))) (rule_7_attack (if (logic (local_goal current side_attack)) (logic (local_goal status achieved))) (then (logic (local_goal current ending)) (logic (local_goal status active)))))</pre>	<pre>(rule_8_attack (if (logic (local_goal current ending)) (logic (local_goal status active))) (then (logic (local_goal current ending)) (logic (local_goal status active))))) (rule_9_attack (if (logic (local_goal current ending)) (logic (local_goal status achieved))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))) (rule_10_defense (if (logic (local_goal current side_attack)) (logic (local_goal status fail))) (then (logic (local_goal current mark)) (logic (local_goal status active)))))</pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Quadro 6. Regras cognitivas do goleiro.

Tabela 2. Comparativo de algumas regras cognitivas dos três grupos de jogadores.

Goleiro e Jogadores de Defesa	Jogadores de Meio de Campo e Ataque
<pre>(rule_0_start (if (logic (local_goal current none))) (then (logic (local_goal current advance)) (logic (local_goal status active)))))</pre>	<pre>(rule_0_start (if (logic (local_goal current none))) (then (logic (local_goal current side_attack)) (logic (local_goal status active)))))</pre>
<pre>(rule_3_defense (if (logic (local_goal current mark)) (logic (local_goal status achieved))) (then (logic (local_goal current advance)) (logic (local_goal status active)))))</pre>	<pre>(rule_2_defense (if (logic (local_goal current mark)) (logic (local_goal status achieved))) (then (logic (local_goal current side_attack)) (logic (local_goal status active)))))</pre>

Em resumo, existem regras cognitivas que denotam um conjunto de estados possíveis comuns mínimos a todos os jogadores e regras peculiares aos tipos de atuação de cada conjunto de jogadores apresentado.

Tabela 3. Comparativo das regras cognitivas equivalentes dos três grupos.

Regras Cognitivas Equivalentes		
Goleiro	Jogadores de Defesa	Jogadores de Meio de Campo e Ataque
<pre>(rule_2_defense (if (logic (local_goal current mark)) (logic (local_goal status active))) (then (logic (local_goal current mark)) (logic (local_goal status active))))</pre>	<pre>(rule_2_defense (if (logic (local_goal current mark)) (logic (local_goal status active))) (then (logic (local_goal current mark)) (logic (local_goal status active))))</pre>	<pre>(rule_1_defense (if (logic (local_goal current mark)) (logic (local_goal status active))) (then (logic (local_goal current mark)) (logic (local_goal status active))))</pre>
<pre>(rule_6_attack (if (logic (local_goal current side_attack)) (logic (local_goal status active))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))</pre>	<pre>(rule_6_attack (if (logic (local_goal current side_attack)) (logic (local_goal status active))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))</pre>	<pre>(rule_3_attack (if (logic (local_goal current side_attack)) (logic (local_goal status active))) (then (logic (local_goal current side_attack)) (logic (local_goal status active))))</pre>
<pre>(rule_10_defense (if (logic (local_goal current side_attack)) (logic (local_goal status fail))) (then (logic (local_goal current mark)) (logic (local_goal status active))))</pre>	<pre>(rule_8_defense (if (logic (local_goal current side_attack)) (logic (local_goal status fail))) (then (logic (local_goal current mark)) (logic (local_goal status active))))</pre>	<pre>(rule_5_defense (if (logic (local_goal current side_attack)) (logic (local_goal status fail))) (then (logic (local_goal current mark)) (logic (local_goal status active))))</pre>

A fim de representar visualmente o comportamento dos planos cognitivos de cada grupo de jogadores, foram elaborados, respectivamente, os fluxogramas das figuras 15, 16 e 17. Estes diagramas permitem um melhor entendimento do relacionamento entre as regras de cada plano dos grupo de jogadores. Estes fluxos viriam a ser utilizados como base na modelagem dos autômatos para verificação das referidas regras.

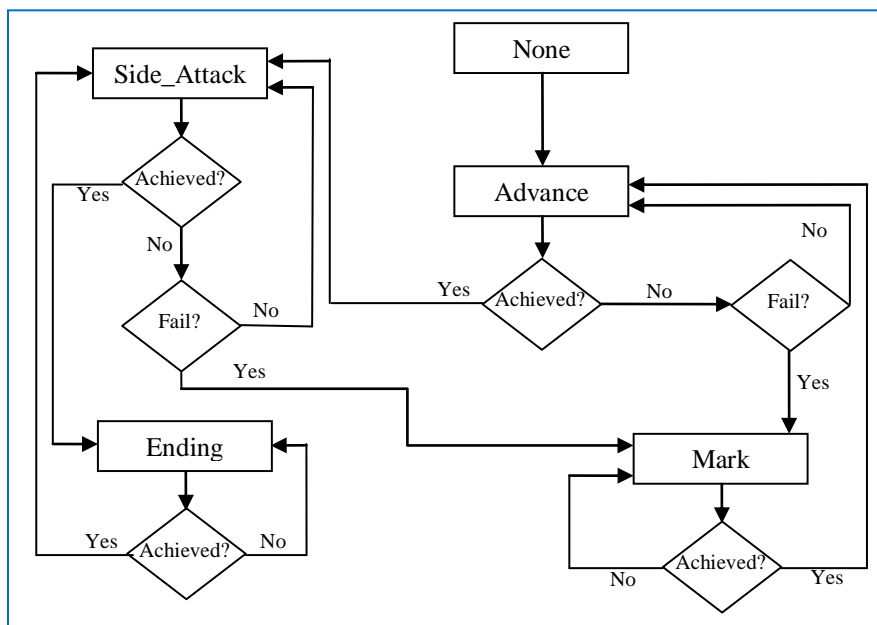


Figura 15. Diagrama de fluxo de transições entre estados oriundo das regras cognitivas do goleiro.

Em todos os diagramas, cada retângulo representa um estado, ou seja, um valor para a variável *local_goal current* com *status* igual a ativo (*local_goal status active*). Logo, se o estado do fluxo for Mark significa dizer que o agente está em um estado de marcação ativo. Os losangos representam um teste lógico para os possíveis *status* de objetivo atingido (*local_goal status achieved*) ou objetivo falhado (*local_goal status fail*).

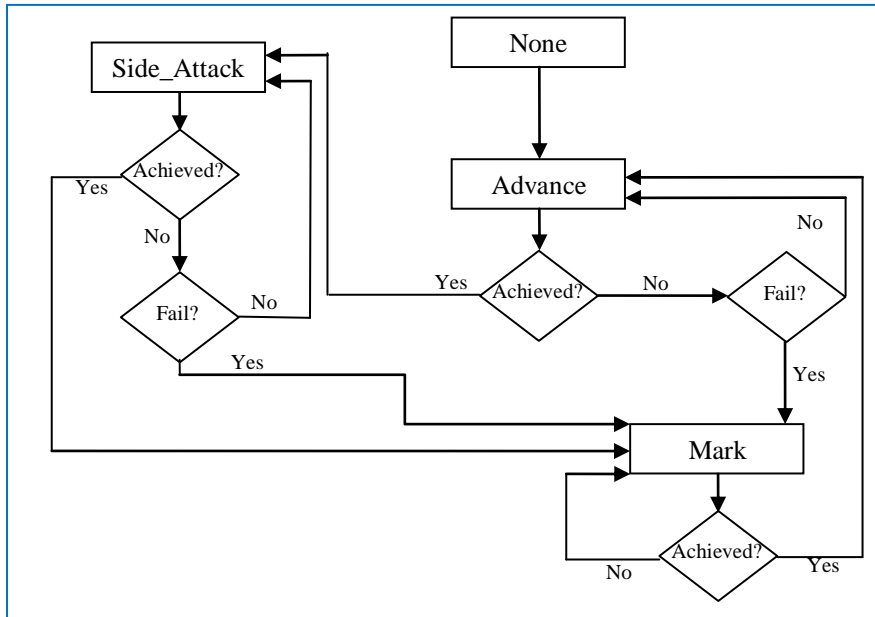


Figura 16. Diagrama de fluxo de transições entre estados oriundos das regras cognitivas dos jogadores de defesa.

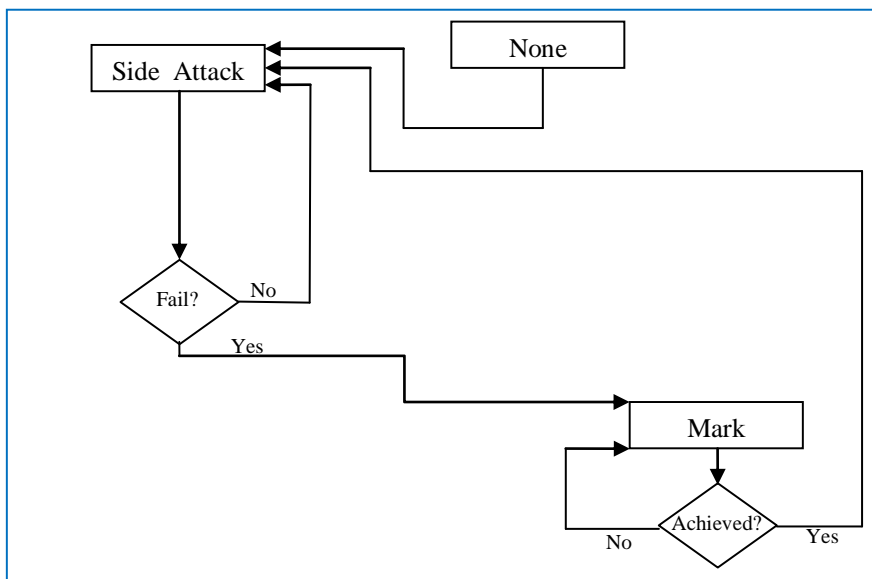


Figura 17. Diagrama de fluxo de transições entre estados oriundos das regras cognitivas dos jogadores de meio de campo e ataque.

A figura 15 apresenta o fluxo de transições entre os estado possíveis do grupo goleiro de acordo com as regras do quadro 6. Já as figuras 16 e 17, mostram o relacionamento entre as regras e os possíveis estados dos demais grupos de jogadores, de acordo com os quadros 4 e 5, respectivamente.

Estes diagramas apresentam de forma mais clara as diferenças entre as regras cognitivas dos três grupos de jogadores. Como exemplo dessas diferenças é a existência do estado *ending* somente nas regras do goleiro. Outro exemplo é que somente nas regras cognitivas dos jogadores de meio de campo e de ataque a transição inicial vai para *side_attack* ao invés de *advance*.

4.2.2.2. Planos Instintivos

Os planos instintivos de cada agente determinam o conjunto de regras que repassam a situação do ambiente para o nível cognitivo e, também, determinam os comportamentos que o nível reativo deve assumir para responder e atuar sobre o ambiente, a exemplo das regras apresentadas no quadro 7.

```

1 (rule_mark_interceptBall
2   (if (logic ( local_goal current mark ))
3     (logic ( local_goal status active ))
4     (logic ( player ball_kickable false ))
5     (logic ( ball position known ))
6     (logic ( player fastest_to_ball true ))
7   )
8   (then (logic ( reactive_behavior active intercept_ball ))
9     (logic ( local_goal status active ))
10    (logic ( local_goal current mark ))
11  )
12 )
13 (rule_mark_achieved
14   (if (logic ( local_goal current mark ))
15     (logic ( local_goal status active ))
16     (logic ( teammate has_ball true ))
17   )
18   (then (logic ( local_goal status achieved ))
19     (logic ( local_goal current mark ))
20  )
21 )

```

Quadro 7. Exemplos de regras instintivas.

A regra *rule_mark_interceptBall* (linha 1) contém 05 (cinco) condições (das linhas 2 a 6) que têm que ser satisfeitas:

- 1) o objetivo local atual (*local_goal current*) é *mark* (marcação);
- 2) o *status* está *active*;
- 3) a bola não está ao alcance do jogador, ou seja, não é “chutável” (*player ball_kickable false*);

- 4) a posição da bola é conhecida (*ball position known*);
- 5) o jogador em questão é aquele que está mais perto da bola (*player fastest_to_ball true*).

Portanto, se todas estas condições forem satisfeitas será ativado o comportamento do nível reativo para interceptar a bola (*reactive_behavior active intercept_ball* (linha 8)) e a base de fatos será mantida (linhas 9 e 10) com o objetivo local e *status* iguais aos anteriores (linhas 2 e 3), haja vista que, efetivamente, a posse de bola não foi retomada ainda.

A regra *rule_mark_achieved* (linha 13) verifica 03 (três) condições:

- 1) o objetivo local atual (*local_goal current*) é *mark* (marcação);
- 2) o estado é ativado (*local_goal status active*);
- 3) a bola é recuperada por algum jogador da equipe (*teammate has_ball true*).

```

1 (rule_2_defense
2 (if (logic ( local_goal current mark ))
3 (logic ( local_goal status active ))
4 )
5 (then (logic ( local_goal current mark ))
6 (logic ( local_goal status active ))
7 )
8 )
9 (rule_3_defense
10 (if (logic ( local_goal current mark ))
11 (logic ( local_goal status achieved ))
12 )
13 (then (logic ( local_goal current advance ))
14 (logic ( local_goal status active ))
15 )
16 )

```

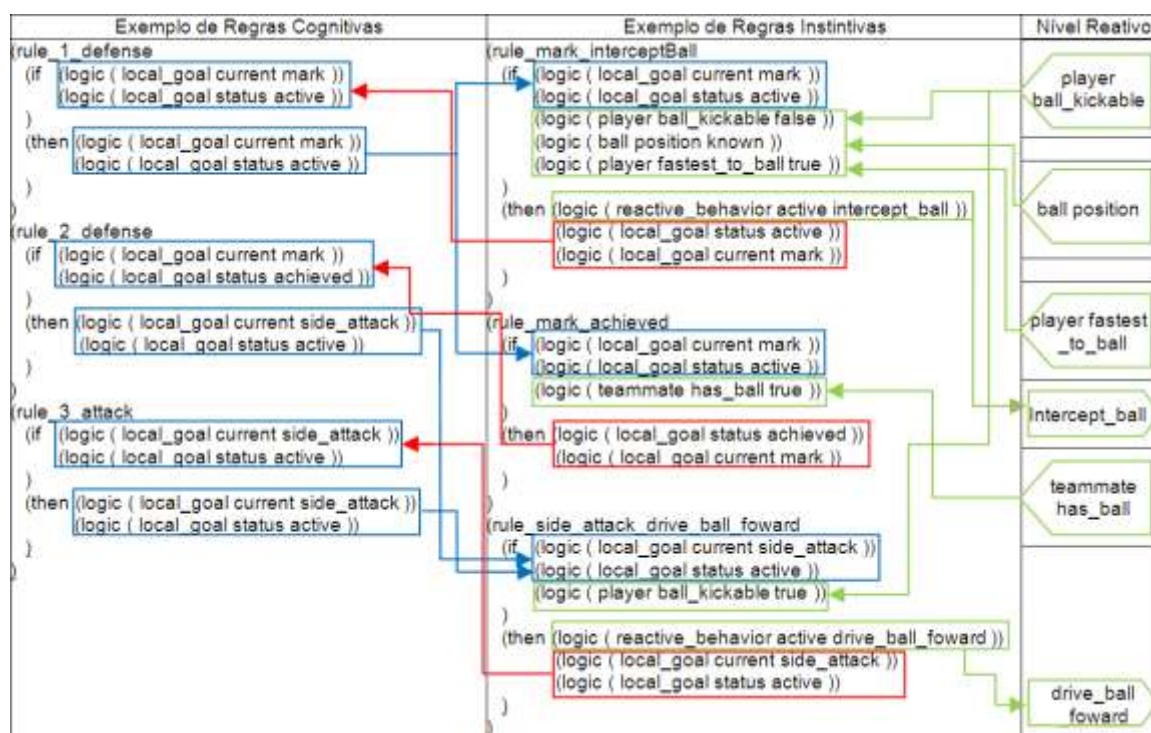
Quadro 8. Exemplos de regras cognitivas que mantém ou alteram o estado de um jogador.

Se as três condições da regra forem satisfeitas, então o lado direito da regra é acionado e o valor do *status* passa a ser atingido (*local_goal status achieved*). Neste caso, o cognitivo deve alterar o objetivo local para *advance*, se for um jogador de defesa ou o goleiro, ou para *side_attack*, se o jogador for meio de campo ou atacante. Em ambas as situações o cognitivo deve alterar o valor do *status* para ativado (*local_goal status active*). A título de exemplo o quadro 8 apresenta as regras cognitivas para os jogadores de defesa ou o goleiro relacionadas com os exemplos de regras instintivas apresentados anteriormente.

Dessa forma, é possível estabelecer uma relação entre as regras cognitivas e as instintivas, considerando que vinculado a cada estado atual e a cada estado futuro (definidos no nível cognitivo) existe um conjunto de regras no nível

instintivo que, caso satisfeito, conduz o cognitivo a promover a alteração do estado do agente.

A interação entre as regras cognitivas e instintivas de um jogador com o nível reativo é apresentada no quadro 9. As regras instintivas verificam nos respectivos lados esquerdos quais são as informações atuais do ambiente, passadas pelo nível reativo, e verificam o estado atual, definido pelo nível cognitivo. Quando todas as sentenças do lado esquerdo de uma determinada regra são satisfeitas, o lado direito desta é alcançado, acionando assim os comportamentos que serão executados pelo nível reativo e alterando o valor do *status* para que o cognitivo possa atualizar o estado do jogador.



Quadro 9. Interação das regras cognitivas e instintivas e o nível reativo.

Por exemplo, na regra cognitiva *rule_1_defense* e as regras instintivas *rule_mark_interceptBall* e *rule_mark_achieved* do quadro 9, o estado atual é marcar (*local_goal current mark*) e o *status* é ativo (*local_goal status active*), o que faz com que as duas regras instintivas tenham as suas duas primeiras condições satisfeitas.

A regra *rule_mark_interceptBall* só poderá ativar o comportamento reativo de interceptar a bola (*reactive_behavior active intercept_ball*) se as informações do ambiente repassadas pelo reativo contiverem as condições:

- 1) a bola não está ao alcance do jogador (*player ball_kickable false*);

- 2) a posição da bola é conhecida (*ball position known*);e
- 3) o jogador é o agente que pode chegar mais rápido à bola (*player fastest_to_ball true*).

Caso uma dessas três condições não seja satisfeita a regra será descartada, ou seja, o lado direito não será acionado.

No caso da regra *rule_mark_achieved* o lado direito só será acionado se o nível reativo informar que algum jogador da equipe recuperou a posse de bola (*teammate has_ball*). Se tal situação ocorrer, significa que não há necessidade de ativar nenhum comportamento reativo, pois o objetivo de marcação já foi atingindo. Já neste caso, o lado direito da regra só alterará o valor de *status* para atingido (*local_goal status achieved*), o que, no nível cognitivo, implicará no acionamento da regra *rule_2_defense* que, por sua vez, promoverá a alteração e ativação do estado atual *mark* para *side_attack*. Estes interrelacionamentos estão representados na figura 18.

Como exemplo, no caso do jogador 9, existe correspondência e interação entre as regras *side_attack* dos dois níveis, bem como, analogamente, existe a mesma relação para as regras *mark* cognitivas e instintivas. Já a regra *start* do nível cognitivo (que identifica que não existe um objetivo local, ou seja, *local_goal current none*) não possui nenhuma regra instintiva relacionada. Isso significa dizer que, incondicionalmente, no início da partida, o estado futuro será *side_attack*, sem nenhum tipo de consulta prévia ao ambiente.

É importante pontuar que a mudança do estado inicial *none* para outro estado futuro se aplica a todos os agentes, mas para alguns o estado futuro será sempre *side_attack* (jogadores de meio de campo e ataque) e para os demais (goleiro e jogadores de defesa) o estado futuro será *advance*.

Similarmente ao que ocorre com a regra *start* do nível cognitivo de todos os jogadores, nos jogadores de defesa 2 e 3, diferentemente do que ocorre com os outros jogadores de defesa 4 e 5, apesar de existirem regras cognitivas que mudam o estado atual para um estado futuro *side_attack*, não existem regras instintivas para estes jogadores que promovam as suas correspondentes mudanças de estados correspondentes.

Considerando que cada agente (jogador) possui uma visão parcial do ambiente, que difere da visão dos demais membros da mesma equipe, e que o ambiente é

extremamente dinâmico, pode-se deduzir que cada agente, ao ser analisado individualmente, possui uma relação peculiar com o ambiente.

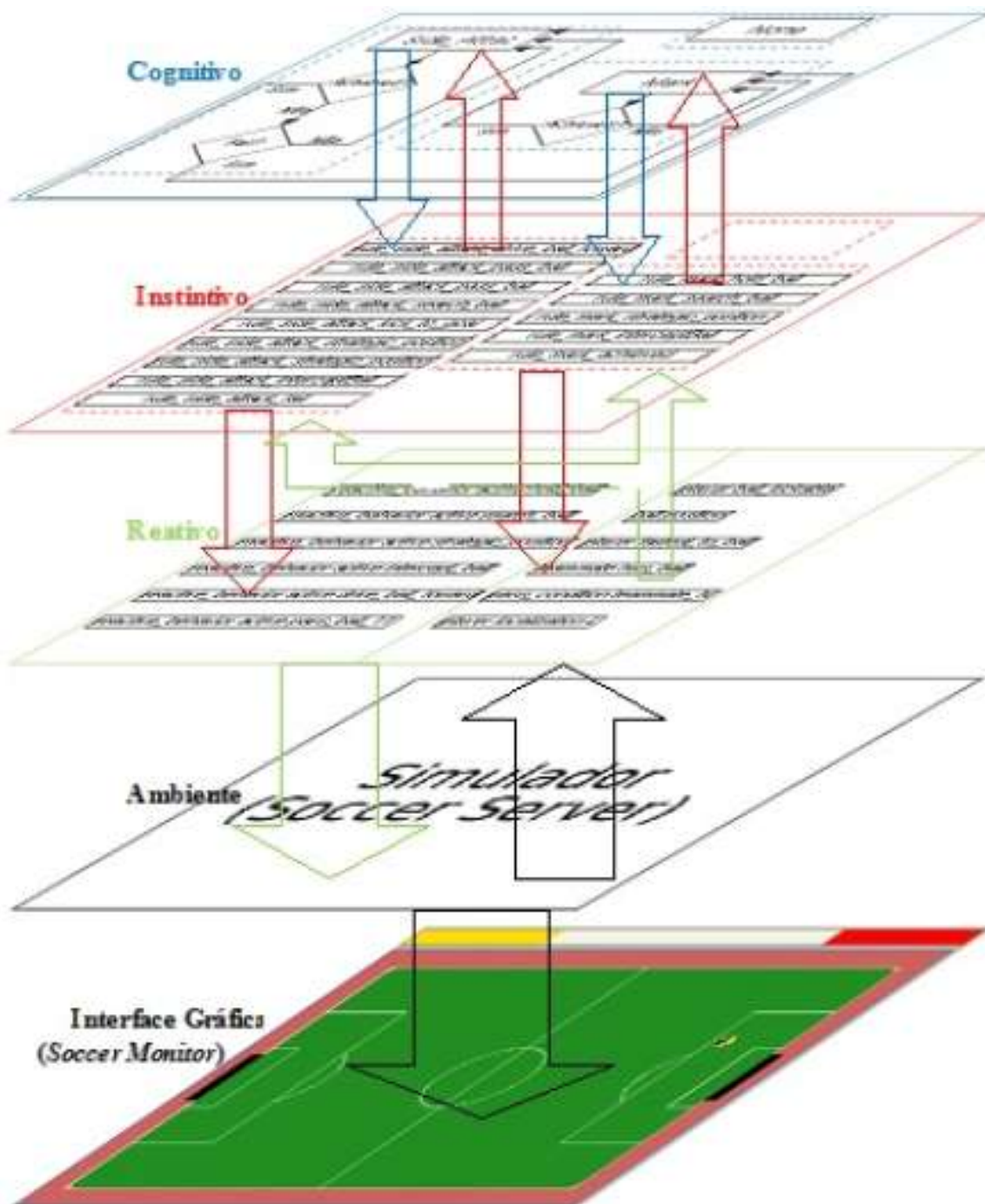


Figura 18. Interação dos diversos níveis de um agente, do ambiente e da interface.

De acordo com as peculiaridades das interações dos níveis cognitivos e instintivos de cada agente, é possível identificar o plano individual de cada jogador, assim como a interação entre suas regras instintivas e cognitivas. Analogamente, ao

serem identificadas as interações entre os níveis de cada grupo de jogadores (goleiro, jogadores de defesa e jogadores de meio de campo e ataque) e da equipe como um todo é possível identificar suas possibilidades de atuação coletiva, em resposta às mudanças do ambiente e ao comportamento da própria equipe.

O modelo apresentado na figura 18, que trata de um único agente, pode ser expandido para representar um modelo de interação coletivo da equipe, considerando as peculiaridades de cada grupo de jogadores, conforme apresentado na figura 19.

A figura 19 representa o fluxo de informações entre os agentes individualmente e o ambiente simulador *Soccerserver*, e deste último com a interface *Soccermonitor*. Esta última interação é unidirecional no sentido *Soccerserver* para o *Soccermonitor* e visa representar graficamente o jogo simulado pelo ambiente. Já a primeira relação, numa visão *bottom-up*, indica que cada agente recebe, via nível reativo, informações que lhes são pertinentes (vide setas tracejadas finas sentido *Soccerserver* – nível reativo), o que implica numa visão do ambiente própria de cada agente. A partir da sua visão do ambiente, cada agente terá suas regras instintivas selecionadas para poder definir o seu estado futuro (através das regras cognitivas) e suas ações do nível reativo para atuar sobre o ambiente (vide setas verdes sentido nível reativo – *Soccerserver*).

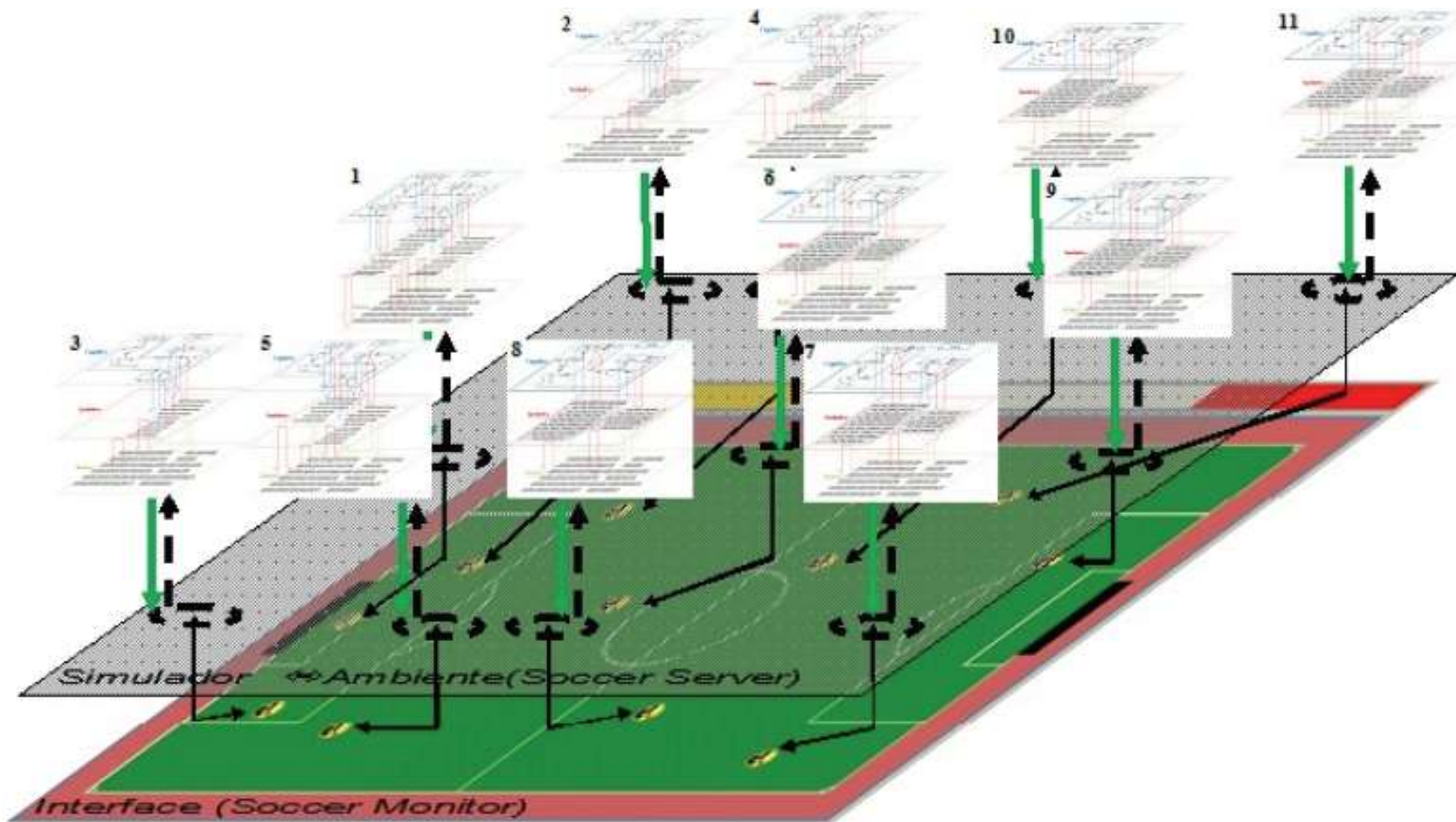


Figura 19. Interação dos diversos níveis de cada agente e da equipe como um todo, do ambiente e da interface.

5. ESPECIFICAÇÃO E VERIFICAÇÃO DOS PLANOS

A verificação de planos de AA's e SMA's não é uma tarefa trivial. A depender do tamanho e complexidade dos planos e do ambiente, esta tarefa pode demandar tempo e esforço para analisar, especificar e verificar estes sistemas. Neste sentido, o futebol de robôs apresenta características como não determinismo e dinamismo que tornam a verificação de planos deste sistema, além de complexa, sujeita a incorrência de situações de explosão de estados.

Para atender a estas questões foi definido um método de verificação baseado nas abordagens de sistemas complexos e paralelizados e/ou hierárquicos e modulares referenciados pela verificação composicional de modelos. O método de verificação definido se baseou na idéia de uma modelagem e verificação incremental por etapas, com sucessivos refinamentos nos modelos especificados, desde as regras mais elementares até a verificação do time de jogadores.

Complementarmente ao método de verificação foram aplicadas, em cada etapa deste, algumas técnicas baseadas na verificação composicional de modelos e abstrações. A utilização destas técnicas e de abstrações teve como objetivo reduzir o tamanho do modelo do sistema e assim contornar / minimizar as ocorrências de explosões de estados.

Nas próximas seções são apresentados o método utilizado no processo de modelagem e verificação do sistema, dados relativos ao ambiente utilizado para a execução da verificação, a especificação dos planos e as técnicas de verificação composicional de modelos utilizadas.

5.1. MÉTODO DE VERIFICAÇÃO E AMBIENTE UTILIZADOS NO PROCESSO DE MODELAGEM E VERIFICAÇÃO

O método de verificação foi concebido considerando aspectos da verificação composicional de modelos. Tal utilização seguiu o conceito de decomposição natural do sistema para validá-lo por partes, a fim de permitir uma redução da complexidade do modelo e sua evolução incremental, não só para facilitar o processo de verificação, como evitar situações de explosão de estados. Esta visão possibilita a representação e verificação individual e isolada de cada

componente do sistema para, posteriormente, inferir-se a correção das propriedades globais do mesmo.

A verificação composicional de modelos leva em consideração a sua aplicabilidade para a especificação de sistemas hierárquicos e modulares e/ou sistemas complexos e paralelizados. Como os SMA's, em geral, apresentam ambas as características, o método de verificação foi idealizado levando em conta esta condição.

Os SMA's se apresentam como sistemas que possuem características de hierarquização e modularização. Normalmente, os agentes que compõem estes sistemas têm arquiteturas multicamadas, as quais possuem uma hierarquia entre si e são componentes (módulos) destes agentes.

Por outro lado, as características de complexidade e paralelismo são inerentes a própria natureza dos SMA's, pois, por definição, tais sistemas são compostos de dois ou mais agentes que devem resolver problemas de forma colaborativa. O grau de complexidade e paralelismo depende diretamente de fatores como: o dinamismo e o não determinismo do ambiente, o domínio do ambiente, a quantidade de agentes, etc.

Considerando os aspectos levantados e as características do futebol de robôs e sua arquitetura em multicamadas método utilizado foi composto das etapas 1 a 6 abaixo:

- 1) Análise das regras cognitivas e das regras instintivas dos jogadores.
- 2) Especificação e verificação individual dos jogadores que possuíam características similares, de acordo com os seguintes passos:
 - Análise, levantamento e agrupamento dos jogadores que possuíam regras cognitivas e instintivas comuns;
 - Especificação de um jogador de cada grupo;
 - Levantamento das propriedades a serem verificadas;
 - Verificação de um jogador de cada grupo com o registro dos erros encontrados e realização dos ajustes nas regras que impediam a verificação de outras propriedades;
 - Refinamento da especificação das regras para a etapa posterior.
- 3) Especificação e verificação individual de todos os jogadores, de acordo com os seguintes passos:

- Especificação de cada jogador incluindo as regras que demandavam interação com outros jogadores;
 - Levantamento das propriedades a serem verificadas;
 - Verificação individual de cada jogador com o registro dos erros encontrados e a realização de ajustes nas regras que impediam a verificação de outras propriedades;
 - Refinamento da especificação das regras para a etapa posterior.
- 4) Especificação e verificação do ambiente, de acordo com os seguintes passos:
- Especificação do ambiente de forma a permitir a reprodução do comportamento coletivo da equipe;
 - Levantamento das propriedades a serem verificadas;
 - Verificação do modelo do ambiente com o registro dos resultados e a realização de ajustes, refinamentos na especificação do ambiente e reavaliação das propriedades.
- 5) Especificação e verificação da equipe, de acordo com os seguintes passos:
- Especificações complementares necessárias à reprodução do comportamento coletivo e identificação de suas limitações;
 - Levantamento das propriedades a serem verificadas;
 - Verificação coletiva de grupos de jogadores até a equipe completa, identificando soluções de contorno quando ocorrer explosão de estados;
 - Ajustes, refinamentos na especificação das regras e nova verificação das propriedades.
 - Coleta e estudo dos resultados.

A figura 20 apresenta uma representação gráfica do método de trabalho adotado.

Foram utilizadas 02(duas) versões acadêmicas³ e de uso não comercial do UPPAAL para a especificação e verificação dos planos dos jogadores, do ambiente e da equipe:

³ Foi testada a versão 4.0.13(rev. 4409), de setembro de 2010, 32 bits para Windows, contudo a mesma não foi utilizada pelo fato de não reportar o tempo decorrido e a quantidade de memória usada após a verificação das propriedades.

- 1) UPPAAL 4.1.2 (rev. 4409), de setembro de 2009, 32 bits para Windows, instalado nos microcomputadores 01 e 02;
- 2) UPPAAL 4.1.4 (rev. 4857), de julho de 2011, 64 bits para MacOS, instalado no microcomputador 03;

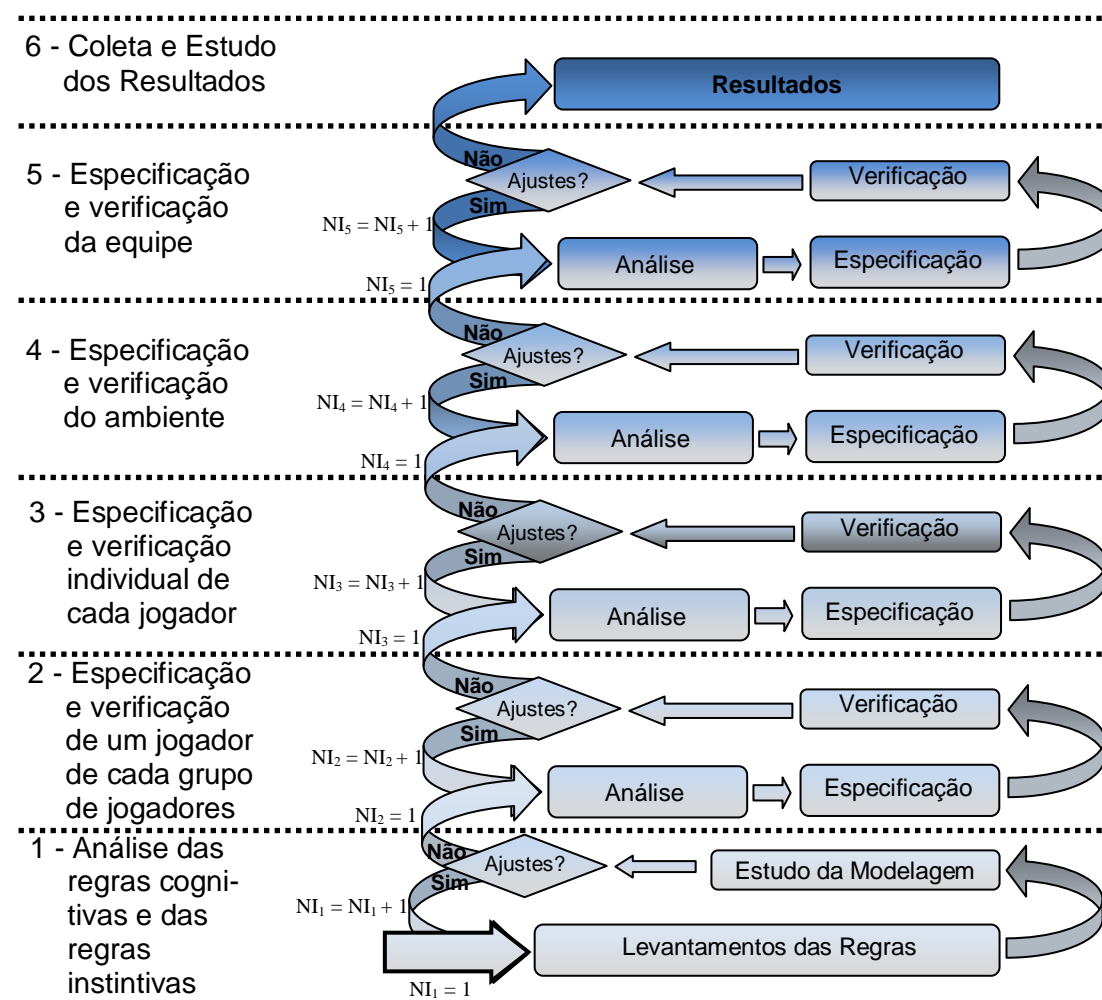


Figura 20. Representação do modelo de verificação incremental e evolutivo adotado.

5.2. ESPECIFICAÇÃO DOS PLANOS

Como definido no escopo do projeto, os comportamentos e ações adotados pelo nível reativo foram abstraídos, pois os mesmos não compõem a elaboração dos planos dos agentes. Comparando ao futebol de robôs físico, a camada reativa equivale aos comandos de acionamento dos mecanismos de atuação e percepção do ambiente (“como fazer” e “como perceber”), enquanto os níveis cognitivos e instintivos estão vinculados ao processo de definição do “o que fazer”, o planejamento.

As regras cognitivas e instintivas de todos os agentes foram representadas de modo a permitir a sua reutilização quando estas são comuns a alguns agentes.

No processo de verificação foi adotada uma abordagem *top-down* (do nível cognitivo para o instintivo) de acordo com os estudos relatados no capítulo 4. Deste modo, foram inicialmente especificados os planos cognitivos e, posteriormente, os planos instintivos dos jogadores.

5.2.1. Especificação dos Planos Cognitivos

Os planos cognitivos dos jogadores foram especificados por grupos de jogadores. Cada jogador possui o mesmo plano cognitivo dos demais membros do seu grupo, tendo em vista que, em cada grupo, todos possuem as mesmas regras cognitivas.

Cada grupo de jogadores teve o seu plano especificado em um único autômato. Sendo assim, as regras cognitivas foram modeladas como *templates* do UPPAAL para os três tipos de agentes (goleiro (1), jogadores de defesa (2, 3, 4 e 5) e jogadores de meio de campo e ataque (6, 7, 8, 9, 10 e 11)), conforme as figuras 21, 22 e 23, que correspondem às figuras 15, 16 e 17 da seção 4.2.1.

Para a especificação dos planos as variáveis *local_goal current*, *local_goal status* e seus respectivos conjuntos de valores possíveis, foram representados por estados, transições, invariantes e outros elementos do formalismo em questão de modo a manter uma correspondência entre o modelo e o sistema.

A variável *local_goal current* das regras define o estado em que o jogador encontra-se durante cada instante da partida e pode assumir os valores *mark*, *advance*, *side_attack*, *ending* e *none*. Assim, para cada valor possível foi criado um estado correspondente nos autômatos do modelo.

A variável *local_goal current* foi representada como *LocalGoalCurrent* e teve os seus possíveis valores representados da seguinte forma⁴: *mark* = 0, *advance* = 1, *side_attack* = 2, *ending* = 3 e *none* = 4.

⁴ Os valores das variáveis em questão foram representados como inteiros devido ao fato de UPPAAL não possuir tipo char ou string.

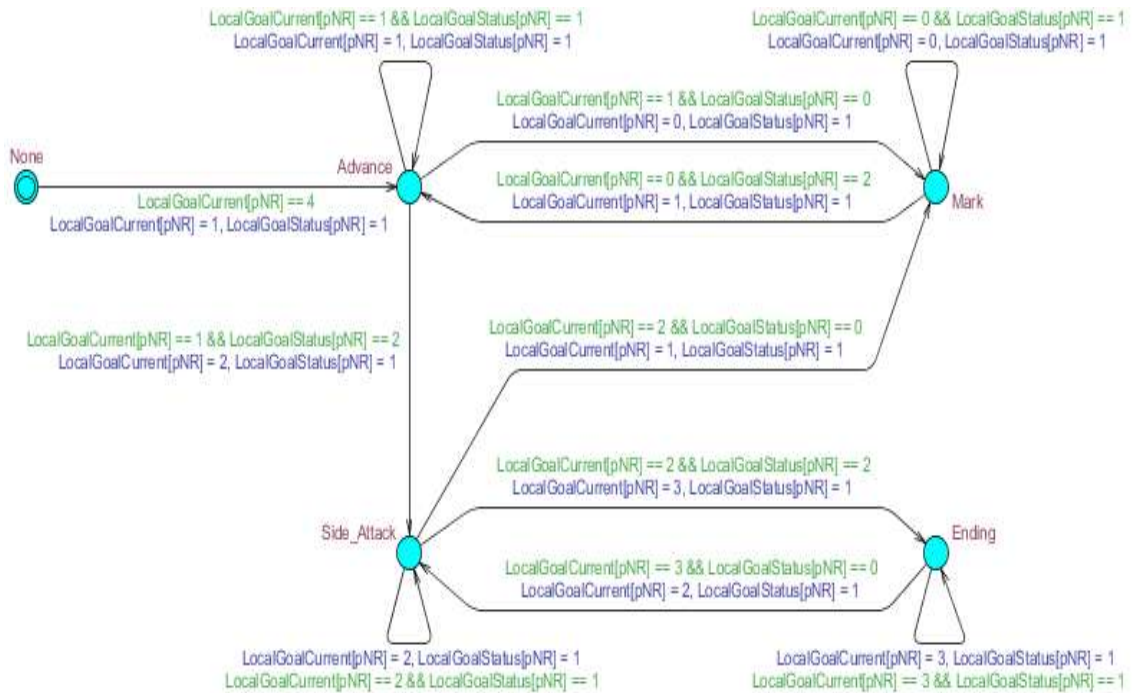


Figura 21. Autômato das Regras Cognitivas do Goleiro

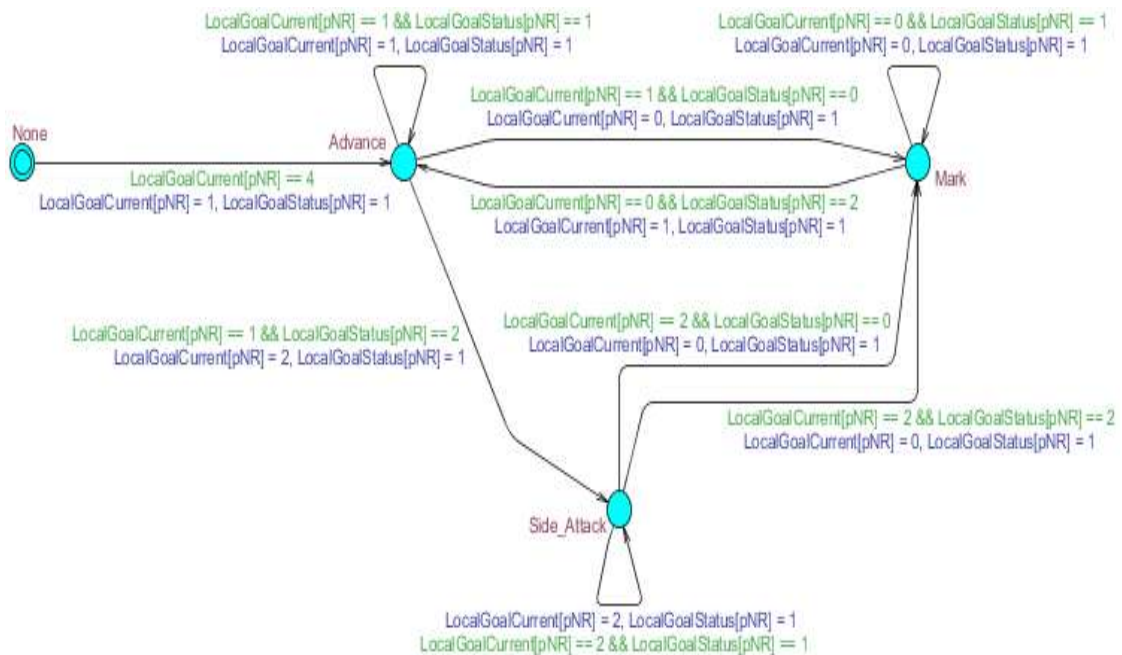


Figura 22. Autômato das Regras Cognitivas dos Jogadores de Defesa.

Analogamente, a variável *local_goal_status* das regras foi representada como *LocalGoalStatus* e teve os seus valores representados como⁴: *fail* = 0, *active* = 1, *achieved* = 2 e *none* = 3.

A tabela 4 apresenta um resumo da representação das variáveis vinculadas às regras cognitivas e sua correspondência em relação à especificação.

As variáveis *LocalGoalCurrent* e *LocalGoalStatus* foram definidas como vetores de 12 (doze) posições de inteiros de escopo global no sistema, conforme a figura 24. Com esta representação um único *template* seria instanciado para cada jogador, bastando passar o número do jogador na equipe via parâmetro *pNR* (***player Number***), conforme visto nas transições dos autômatos das figuras 21, 22 e 23.

Portanto, cada jogador acessará sua posição correspondente de acordo com o seu número na equipe, ou seja, o seu *pNR*. Assim, o jogador 2 recebe o parâmetro *pNR* = 2 e acessa a posição 2 dos vetores *LocalGoalCurrent*[*pNR*] \Leftrightarrow *LocalGoalCurrent*[2] e *LocalGoalStatus*[*pNR*] \Leftrightarrow *LocalGoalStatus*[2].

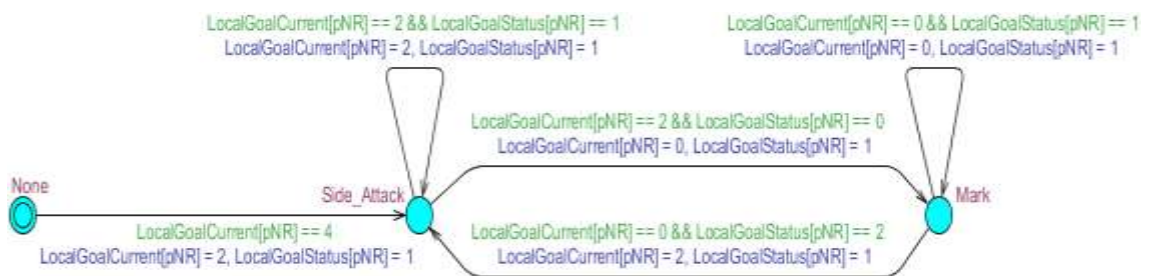


Figura 23. Autômato das Regras Cognitivas dos Jogadores de Meio de Campo e Ataque.

Tabela 4. Relação entre as variáveis nas regras dos planos cognitivos e no modelo.

Variáveis Existentes nas Regras		Representação das Variáveis no Modelo	
Nome da Variável	Valores possíveis	Nome da Variável	Valores possíveis
<i>local_goal current</i>	<i>mark</i>	<i>LocalGoalCurrent</i>	0
	<i>advance</i>		1
	<i>side_attack</i>		2
	<i>ending</i>		3
	<i>none</i>		4
<i>local_goal status</i>	<i>fail</i>	<i>LocalGoalStatus</i>	0
	<i>active</i>		1
	<i>achieved</i>		2
	<i>none</i>		3

```

Editor | Simulator | Verifier
Drag out // Place global declarations here.
int LocalGoalCurrent [12] = {-1,4,4,4,4,4,4,4,4,4,4,4}; /* Mark = 0, Advance = 1,
Side Attack = 2, Ending = 3,
None = 4 */
int LocalGoalStatus [12] = {-1,3,3,3,3,3,3,3,3,3,3,3}; /* Fail = 0, Active = 1, Achieved = 2,
None = 3 */

```

Figura 24. Declaração das variáveis cognitivas no UPPAAL.

Esta definição vale para todos os jogadores e, por conveniência⁵, somente a posição 0 (zero) do vetor não está associada a nenhum jogador, conforme visto na figura 25.

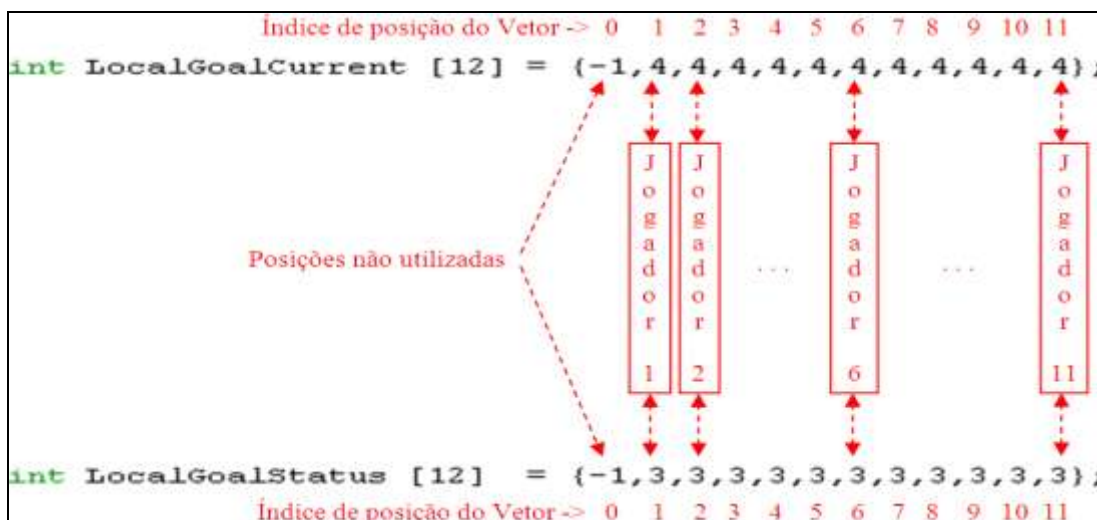


Figura 25. Definição da relação entre os jogadores e as posições dos vetores de variáveis do sistema.

Nos exemplos das figuras 24 e 25, os valores apresentados para todos os jogadores são: *LocalGoalCurrent*[pNR] igual a 4 e *LocalGoalStatus*[pNR] igual a 3. Isso significa dizer que todos os jogadores estão no estado “None” e o *status* do estado é “None”, ou seja, todos os jogadores estão no estado inicial.

5.2.2. Especificação dos Planos Instintivos

Os planos instintivos dos jogadores, assim como os cognitivos, são resultantes da combinação das regras instintivas próprias de cada jogador, conforme foi identificado na seção 4.2.2. Contudo, cada jogador pode possuir um conjunto de regras diferente dos outros jogadores, mesmo que estes realizem funções similares.

Portanto, diferentemente do que ocorre com os planos cognitivos, os planos instintivos podem apresentar padrões de comportamento muito variados, o que não permite que uma regra padrão atenda a um grupo de jogadores, como no plano cognitivo.

Por esse motivo foi identificado que seria mais produtivo especificar uma a uma as regras do plano instintivo ao invés de agrupá-las num conjunto de regras

⁵ O UPPAAL considera o índice dos seus vetores e matrizes a partir da posição 0.

(plano), como foi feito no nível cognitivo. Apesar disto, por outro lado, foi identificado que existem regras instintivas que compõem os planos instintivos de diversos jogadores

Sendo assim, cada regra foi especificada separadamente como um *template*. Isto possibilita um bom nível de reusabilidade, além de permitir a verificação de forma modular, incremental e composicional, na medida em que novas regras podem ser adicionadas conforme a evolução do modelo e da complexidade de sua especificação.

Diante disso, complementarmente aos estudos relatados na seção 4.2.2 do capítulo 4, foi realizado um levantamento para identificar as regras e quais jogadores que as utilizam na composição de seus respectivos conjuntos de regras instintivas, ou seja, nos seus planos instintivos.

Este levantamento teve como resultado a tabela apresentada no Apêndice A, a qual está resumida na tabela 5, e teve como resultado a identificação da existência de um total de 37 (trinta e sete) regras nos planos instintivos de todos os jogadores.

Considerando o rol das regras instintivas e o método de trabalho definido para a verificação na seção 5.1.1, foi identificado que, das 37(trinta e sete) regras, 08 (oito) implicam em interação entre os jogadores.

Tendo em vista que as regras que implicam em interação entre os jogadores possuem um grau de complexidade maior para especificar e verificar, por conveniência, estas foram desconsideradas em um primeiro momento. A idéia por trás desta decisão é que, seguindo o método, estas regras seriam especificadas e adicionadas aos planos dos jogadores de forma gradual, a medida que o modelo e o processo de verificação fossem evoluindo.

As regras não especificadas nas etapas iniciais foram:

- *rule_advance_pass_ball_6*;
- *rule_advance_pass_ball_7*;
- *rule_advance_pass_ball_8*;
- *rule_side_attack_pass_ball_7*;
- *rule_side_attack_pass_ball_8*;
- *rule_side_attack_pass_ball_9*;
- *rule_side_attack_pass_ball_10*;e
- *rule_side_attack_pass_ball_11*.

Tabela 5. Resumo dos jogadores e respectivas regras instintivas.

Nr	Regras Instintivas	Jogadores											Total de jogadores por regras
		1	2	3	4	5	6	7	8	9	10	11	
1	rule_mark_hold_ball	x	x	x	x	x	x	x	x	x	x	x	11
2	rule_mark_search_ball	x	x	x	x	x	x	x	x	x	x	x	11
3	rule_mark_opponent	x	x	x	x	x							5
4	rule_mark_interceptBall	x	x	x	x	x	x	x	x	x	x	x	11
5	rule_mark_strategic_position						x	x	x	x	x	x	6
6	rule_mark_achieved	x	x	x	x	x	x	x	x	x	x	x	11
7	rule_advance_search_ball		x	x	x	x							4
8	rule_advance_interceptBall	x	x	x	x	x							5
9	rule_advance_strategic_position	x	x	x	x	x							5
10	rule_advance_pass_ball_foward	x	x	x	x								4
11	rule_advance_pass_ball_6					x							1
12	rule_advance_pass_ball_7		x										1
13	rule_advance_pass_ball_8		x			x							2
14	rule_advance_drive_ball_foward						x						1
15	rule_advance_drive_ball_fast						x						1
16	rule_advance_fail	x	x	x	x	x							5
17	rule_advance_achieved	x			x								2
18	rule_side_attack_search_ball	x			x		x	x	x	x	x	x	8
19	rule_side_attack_interceptBall	x			x		x	x	x	x	x	x	8
20	rule_mark_markOpponent	x			x								2
21	rule_side_attack_pass_ball_7	x			x								2
22	rule_side_attack_pass_ball_8						x	x					2
23	rule_side_attack_pass_ball_9						x	x	x		x	x	5
24	rule_side_attack_pass_ball_10							x	x	x			3
25	rule_side_attack_pass_ball_11						x		x	x			3
26	rule_side_attack_drive_ball_foward								x	x	x	x	4
27	rule_side_attack_kick_to_goal									x	x	x	3
28	rule_side_attack_strategic_position_free_mark						x	x	x	x			4
29	rule_side_attack_strategic_position									x	x	x	3
30	rule_side_attack_drive_ball_fast										x	x	2
31	rule_side_attack_fail	x			x		x	x	x	x	x	x	8
32	rule_side_attack_achieved	x			x								2
33	rule_ending_kick_ball	x											1
34	rule_ending_search_ball	x											1
35	rule_ending_interceptBall	x											1
36	rule_ending_fail	x											1
37	rule_ending_achieved	x											1
Total de regras por jogador		20	12	10	17	11	14	12	13	14	13	13	149
		1	2	3	4	5	6	7	8	9	10	11	
Jogadores													

5.2.2.1 Análise e especificação das variáveis utilizadas nas regras instintivas

Diferentemente do que ocorre no plano cognitivo, as regras instintivas utilizam variáveis que representam, para cada jogador, a visão particular que o

mesmo recebe do seu nível reativo, ou seja, a percepção que é passada pelo ambiente de simulação (*Soccerserver*).

Além disso, as regras do nível instintivo apresentam o acionamento de comportamentos do nível reativo (*reactive_behavior active*), já citados neste capítulo e no capítulo 4 (vide quadro 9 da seção 4.2.2).

O nível reativo e seus comportamentos foram abstraídos como um único estado a ser atingido, depois de validada uma determinada regra do nível instintivo correlacionada ao acionamento de um comportamento específico.

No *Soccerserver* os jogadores recebem e enviam informações de auto-posicionamento e também recebem informações sobre o posicionamento relativo dos demais componentes do jogo (jogadores da mesma equipe, jogadores adversários e, principalmente, a bola). Essas informações são passadas e recebidas pelo simulador para cada jogador, sendo que os componentes do jogo que estiverem mais distantes recebem as informações passadas pelo *Soccerserver* com menor exatidão que aqueles que estejam mais próximos. Esta forma de apresentar as informações para os jogadores é o fator preponderante para que estes tenham uma visão particular e parcial do ambiente. Logo, é necessário que leve em conta estas características na modelagem destas representações.

As tabelas 6 e 7 apresentam, respectivamente, as variáveis existentes nas regras do nível instintivo e identificação de seus respectivos tipos de dados e valores possíveis, de acordo com a percepção do ambiente *Soccerserver* pelo nível reativo.

Do mesmo modo que as variáveis das regras cognitivas, todas as variáveis das regras instintivas foram representadas como vetores de 12(doze) posições. As variáveis que possuíam valores *true* e *false* ou *know* e *unknow* foram especificadas como vetores do tipo booleano. Já a variável *player localization* não teve, inicialmente, um tipo definido, tendo em vista que a mesma guarda a região do campo na qual cada jogador se encontra em um determinado momento da partida, conforme as informações recebidas do ambiente via nível reativo.

Dentre as variáveis encontradas, cabe uma especial consideração sobre a variável *player localization*. Esta variável identifica a posição do jogador como um dos fatores de decisão para a execução de ações vinculadas às 03(três) regras :

- *side_attack_kick_to_goal*;
- *side_attack_strategic_position_free_mark*;

- *side_attack_drive_ball_fast.*

Tabela 6. Resumo das Variáveis X Regras Instintivas.

Nr	Regras Instintivas	Variáveis						Total de variáveis por regras
		player_ball_kickable	ball_position	player_fastest_to_ball	teammate_has_ball	player_localization	midplayers_has_ball	
1	rule_mark_hold_ball	x						1
2	rule_mark_search_ball	x	x					2
3	rule_mark_opponent	x			x			2
4	rule_mark_interceptBall	x	x	x				3
5	rule_mark_strategic_position	x	x	x				3
6	rule_mark_achieved				x			1
7	rule_advance_search_ball	x	x					2
8	rule_advance_interceptBall	x	x	x				3
9	rule_advance_strategic_position	x	x	x				3
10	rule_advance_pass_ball_foward	x	x					2
14	rule_advance_drive_ball_foward	x						1
15	rule_advance_drive_ball_fast	x						1
16	rule_advance_fail				x			1
17	rule_advance_achieved					x		1
18	rule_side_attack_search_ball	x	x					2
19	rule_side_attack_interceptBall	x	x	x				3
20	rule_mark_markOpponent	x	x	x				3
26	rule_side_attack_drive_ball_foward	x						1
27	rule_side_attack_kick_to_goal	x				x		2
28	rule_side_attack_strategic_position_free_mark	x	x	x		x		4
29	rule_side_attack_strategic_position	x	x	x				3
30	rule_side_attack_drive_ball_fast	x				x		2
31	rule_side_attack_fail				x			1
32	rule_side_attack_achieved				x			1
33	rule_ending_kick_ball	x	x					2
34	rule_ending_search_ball	x	x					2
35	rule_ending_interceptBall	x	x	x				3
36	rule_ending_fail				x			1
37	rule_ending_achieved							0
Total de regras por variáveis		22	15	9	5	3	1	55

Tabela 7. Resumo das variáveis existentes no nível instintivo, seus valores possíveis e tipos.

Variável	Conjunto de valores possíveis encontrados nas regras	Tipo
<i>player_ball_kickable</i>	{ true, false }	Booleana
<i>player_fastest_to_ball</i>		
<i>teammate_has_ball</i>		
<i>midplayers_has_ball</i>		
<i>ball_position</i>	{ known, unknown }	Caracter
<i>player_localization</i>	{ AREA_ATTACK, AREA_SIDE_ATTACK }	



Figura 26. Divisão do campo em 06 regiões para o posicionamento dos jogadores e da bola.

Além disso, é necessário observar que, durante toda a partida, os jogadores e a bola mudam de posição e essa posição relativa de cada jogador e bola influencia na percepção do mesmo sobre todas as outras variáveis. Para representar o deslocamento da bola e os jogadores pelo campo e os valores para a variável *player localization*, foi estabelecida uma simplificação do modelo de posicionamento do ambiente *Soccerserver* de 2D(duas dimensões) para 1D(unidimensional). Por esses motivos a variável *player localization* foi definida como uma variável inteira cujos valores possíveis podem variar de 0 a 5, conforme a figura 26.

De acordo com a definição adotada para a representação do campo, o posicionamento dos jogadores e da bola e os valores da variável *player localization* correspondem a uma das seis regiões apresentadas na figura 26, conforme a tabela 8.

A idéia por trás dessa simplificação é permitir que durante a verificação individual dos agentes seja possível simular a movimentação dos jogadores e da bola, sem ter a necessidade de implementar um algoritmo de movimentação 2D mais complexo, e que permita manter as características da visão particular e parcial do ambiente para cada jogador.

Desta forma, seguindo a mesma abordagem adotada para as demais variáveis, a variável *player localization* foi declarada como um vetor de 12 (doze) posições, no qual a posição 0 não foi utilizada e, da posição 1 a 11, cada espaço do vetor está associado a um dos jogadores e sua visão do jogo (localização).

Sendo assim, as variáveis identificadas foram declaradas no UPPAAL, conforme o quadro 10.

Tabela 8. Relação das regiões do campo e valores da variável *player localization*.

Região do Campo	Valor correspondente à posição no campo
Grande área	0
Intermediária	1
Meio de campo - defesa	2
Meio de campo do adversário	3
Defesa do adversário	4
Área do adversário	5

```
// Instinctive Layer's Variables
bool playerBallKickable [12] = {false, /* Not used */
                                false, /* Keeper 1 */
                                false,false, /* Defenders 2 - 3 - 4 - 5 */
                                false,false,
                                false,false,false,false,false,false}; /* Midfield - attack 6 - 7 - 8 - 9 - 10 - 11*/

bool playerFastesttoBall [12] = {false, /* Not used */
                                  false, /* Keeper 1 */
                                  false,false, /* Defenders 2 - 3 - 4 - 5 */
                                  false,false,
                                  false,false,false,false,false,false}; /* Midfield - attack 6 - 7 - 8 - 9 - 10 - 11*/

bool teammateHasBall [12] = {false, /* Not used */
                              false, /* Keeper 1 */
                              false,false, /* Defenders 2 - 3 - 4 - 5 */
                              false,false,
                              false,false,false,false,false,false}; /* Midfield - attack 6 - 7 - 8 - 9 - 10 - 11*/

bool midplayersHasBall [12] = {false, /* Not used */
                                false, /* Keeper 1 */
                                false,false, /* Defenders 2 - 3 - 4 - 5 */
                                false,false,
                                false,false,false,false,false,false}; /* Midfield - attack 6 - 7 - 8 - 9 - 10 - 11*/

bool ballPosition [12] = {false, /* Not used */
                           false, /* Keeper 1 */
                           false,false, /* Defenders 2 - 3 - 4 - 5 */
                           false,false,
                           false,false,false,false,false,false}; /* Midfield - attack 6 - 7 - 8 - 9 - 10 - 11*/

int playerLocalization [12] = {-1,0,1,1,1,1,1,1,2,2,2,2}; /* Own Area = 0, Defence Area = 1,
                                                         Defence Midfield Area = 2 ,
                                                         Opposing Midfield Area = 3,
                                                         Opposing Defence Area = 4,
                                                         Opposing Area = 5 */
```

Quadro 10. Declaração das variáveis encontradas nas regras instintivas no UPPAAL.

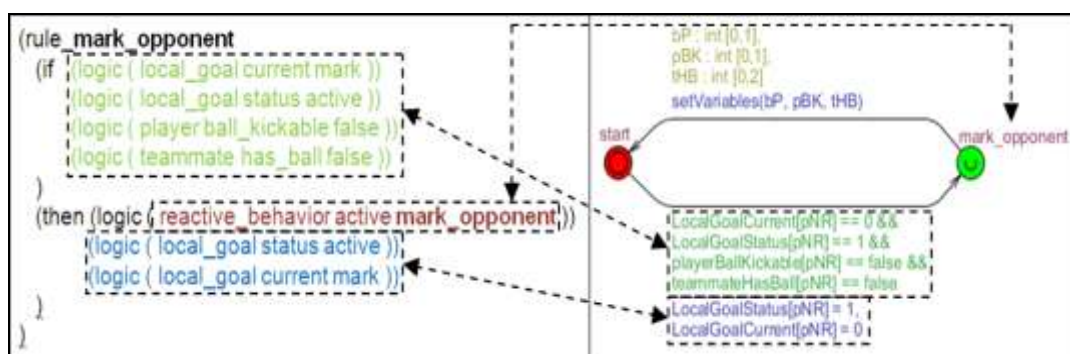
5.2.2.2 Modelagem e especificação das regras instintivas

A modelagem das regras instintivas representa as interações destas com o nível cognitivo, considerando a percepção do ambiente e o acionamento de comportamentos reativos em resposta à percepção obtida, o estado interno e o objetivo de cada jogador.

Tendo em vista a quantidade de regras instintivas, a especificação de todas estas regras encontra-se no Apêndice B. Contudo, de forma geral, existem 02 (dois) tipos de regras instintivas: regras que acionam comportamentos do nível reativo e regras que alteram o objetivo local e o *status* cognitivo de cada jogador.

As regras que acionam comportamentos do nível reativo representam as condições de execução de comportamentos de acordo com o objetivo local, o *status* deste objetivo local e da percepção do jogador em relação ao jogo em um determinado instante, conforme discutido no capítulo 4.

A percepção do jogador é representada pelos valores das variáveis apresentadas na seção anterior. Tais valores são representados como condições para que um comportamento do nível reativo seja acionado. As condições que devem ser satisfeitas para que um determinado comportamento reativo seja acionado foram representadas como guardas (*guards*). O quadro 11 apresenta um exemplo deste tipo de regra.



Quadro 11. Relação entre a regra *mark_opponent* e sua especificação no UPPAAL.

A transição do estado *Start* para o estado *mark_opponent* no autômato do quadro 11 apresenta a relação entre os guardas que condicionam a transição e as variáveis existentes na regra correspondente. Além disso, podem ser identificadas as representações das sentenças da regra que fazem as atualizações dos valores

das variáveis cognitivas *local_goal status* e *local_goal current*, quando a transição é realizada.

Apesar das sentenças (*logic(local_goal status active)*) e (*logic (local_goal current mark)*) serem, efetivamente, desnecessárias, tendo em vista que não alteram o valor das variáveis cognitivas *local_goal status* e *local_goal current* iniciais da regra, as mesmas foram mantidas em todas as especificações de regras que as continham. Esta decisão se deve ao fato de se procurar manter a fidelidade do modelo com o sistema real.

O autômato visto no quadro 11 apresenta a existência de uma transição que retorna do estado *mark_opponent* para o estado inicial (*start*). Este retorno representa a forma como o nível instintivo recebe informações do nível reativo através do ambiente, além de conduzir a regra para um estado inicial, de forma a permitir que a mesma seja utilizada novamente na próxima iteração com o ambiente.

```
1 void setVariables(int pBP, int pPBK, int pTHB)
2 {
3     if ( pBP == 1 )
4         { // if the parameter (pBP) is equals 1 then player knows where is the ball
5             ballPosition[pNR] = true; //player knows where is the ball
6             if (pTHB == 1) {
7                 teammateHasBall[pNR] = true;
8                 playerBallKickable[pNR] = false;
9                 playerFastesttoBall[pNR] = false;
10                if (pMPHB == 1) midplayersHasBall[pNR] = true;
11                else midplayersHasBall[pNR] = false;
12            }
13            else {
14                teammateHasBall[pNR] = false;
15                midplayersHasBall[pNR] = false;
16                if (pPBK == 1){
17                    playerBallKickable[pNR] = true;
18                    playerFastesttoBall[pNR] = true;
19                }
20                else{
21                    playerBallKickable[pNR] = false;
22                    if (pPFtB == 1) playerFastesttoBall[pNR] = true;
23                    else playerFastesttoBall[pNR] = false;
24                }
25            }
26        }
27        else{
28            ballPosition[pNR] = false;
29            teammateHasBall[pNR] = false;
30            midplayersHasBall[pNR] = false;
31            playerBallKickable[pNR] = false;
32            playerFastesttoBall[pNR] = false;
33        }
34    }
```

Quadro 12. Função *setVariables()* vinculada à regra *mark_opponent*.

Para representar o recebimento e tratamento de informações do nível reativo foram criadas variáveis de seleção e uma função *setVariables()* para

representar o dinamismo e o não determinismo do ambiente, de acordo com os atributos dos jogadores vinculados às regras que estão sendo acionadas em um determinado momento. Esta medida se justifica pelo fato dos jogadores, ao serem verificados individualmente, necessitarem de informações de como o ambiente se apresenta depois das suas atuações por meio das regras acionadas por eles mesmos.

As variáveis de seleção (*selection*) existentes no UPPAAL são variáveis inteiras cujo valor pode variar dentro de uma faixa de valores (*range*), definida na declaração das mesmas. A variação entre os valores que estas variáveis podem assumir ocorre de forma pseudo-randômica dentro do *range* declarado para cada variável.

A título de exemplo, as variáveis de seleção $bP : \text{int } [0,1]$, $pBK : \text{int } [0,1]$ e $tHB : \text{int } [0,2]$, apresentadas no quadro 11 representam as variáveis das regras de produção instintivas *ball position*, *player ball_kickable* e *teammate has_ball*, respectivamente. Estas variáveis têm os seus valores selecionados depois do acionamento do comportamento reativo *mark_opponent* e representam como as condições do ambiente podem variar, para cada uma destas variáveis, entre verdadeiro, caso o valor selecionado pseudo-aleatoriamente pelo UPPAAL seja 1, ou falso, caso o valor seja 0 ou 2, no caso específico da variável tHB.

Como as variáveis de seleção são do tipo inteiro e as variáveis das regras instintivas são do tipo booleanas, houve a necessidade de criar, para cada autômato destas regras instintivas, funções que traduzissem os valores inteiros pseudo-randômicos das variáveis *selection* para valores booleanos a serem gravados nas variáveis das regras instintivas. Essas funções, denominadas *setVariables()*, de cada regra, possuem a idéia básica de fazer com que cada jogador, após atingir um estado que representa o acionamento de um comportamento reativo, receba informações (tenha atualizadas suas variáveis) de acordo com a situação do jogo e o seu próprio estado interno.

O quadro 12 apresenta o código da função *setVariables()* relativo à regra apresentada no quadro 11.

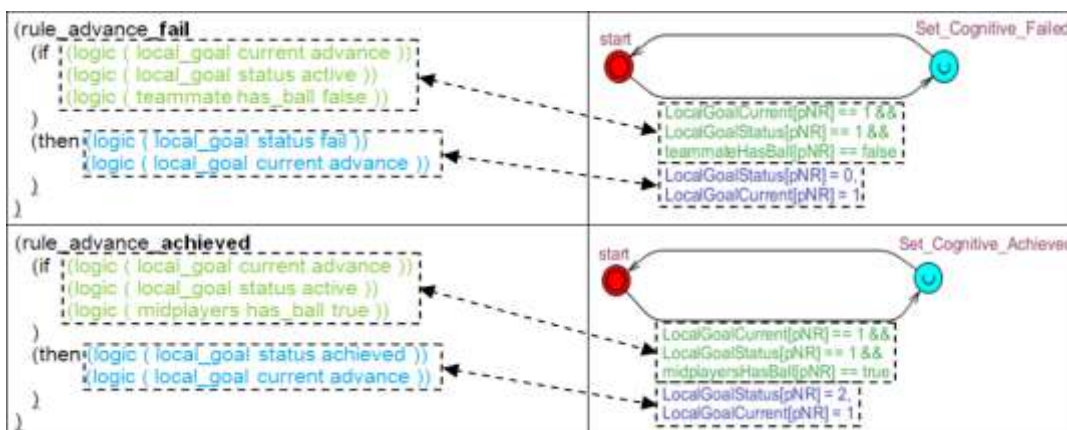
O código do quadro 12 mostra que a função *setVariables()* possui os parâmetros pBP , $pPBK$ e $pTHB$ na sua assinatura (linha1). Cada um destes corresponde, respectivamente, às variáveis de seleção bP , pBK e tHB , vistas no quadro 11. A partir do recebimento destes valores, por passagem de valor, a função

executa um teste lógico identificando se o valor recebido pelo parâmetro pBP é igual a 1 (linha 3). Se no teste lógico o valor de pBP for igual a 1, significa que o jogador recebeu a informação da localização da bola, logo a variável *ballPosition[pNR]* recebe o valor true (linha 4). Caso contrário, significa que o jogador não recebeu a informação da posição da bola e a variável *ballPosition[pNR]* recebe o valor false (linha 28). Como consequência, se o jogador não tem informações sobre a posição da bola, ele não pode identificar se algum companheiro está de posse da bola, se algum companheiro que joga como de meio de campo está de posse da bola, se o jogador está próximo da bola para chutá-la e se o jogador é o que está mais perto da bola para alcançá-la. Assim, conforme apresentado das linhas 29 a 32, as variáveis *ballPosition[pNR]*, *teammateHasBall[pNR]*, *midplayersHasBall[pNR]*, *playerBallKickable[pNR]* e *playerFastesttoBall[pNR]* recebem o valor lógico falso (*false*).

Por sua vez, as regras instintivas que alteram o *status* do objetivo local de cada jogador representam a modificação do *status* de um determinado estado cognitivo que o jogador se encontra para atingido (*achieved* = 2) ou falha (*fail* = 0), para que o nível cognitivo possa alterar o seu estado e *status* futuros. Quando tal situação ocorre não existe nenhum acionamento de comportamentos do nível reativo.

O quadro 13 apresenta a relação entre as regras e suas especificações em autômatos. A transição do estado que representa o nível instintivo (vermelho) simboliza a modificação dos estados das regras cognitivas (azul-claro).

Nas transições do quadro 13 que levam ao estado *start*, não existem variáveis do tipo *select* ou funções, pois, diferentemente das transições apresentadas no quadro 11 as estas não interagem com o nível reativo.



Quadro 13. Relação entre as regras *advance_achieved* e *advance_fail* e suas especificações no UPPAAL.

Por fim, definidas as especificações das regras instintivas e definidas as soluções adotadas para representar o comportamento dos jogadores, suas visões parciais, o dinamismo e o não determinismo do ambiente, estabeleceram-se as condições iniciais para a verificação das regras conforme planejado na seção 5.1.1.

5.3. APLICAÇÃO DE TÉCNICAS DE VERIFICAÇÃO COMPOSICIONAL DE MODELOS

O método de verificação definido se baseou em termos de organização em aspectos conceituais e abordagens adotadas pela verificação composicional de modelos, conforme visto nas seções 3.3.2 e 5.2. Além da observação aos conceitos e abordagens adotadas pela verificação composicional de modelos na definição do método, foram aplicadas algumas técnicas de verificação composicional durante todo o processo de verificação dos planos em cada uma das etapas do método empregado.

Dentre as técnicas de verificação composicional de modelos foram utilizadas as seguintes:

- *Assume – Guarantee*;
- *Lazy parallel composition*;
- *Partitioned transition relations*;
- *Interface process*.

A técnica *Assume – Guarantee* é uma técnica top-down e foi utilizada parcialmente na especificação e verificação em vários níveis do sistema. Considerando cada jogador da equipe como um componente do sistema (da equipe), pode ser dito que, em cada jogador, as regras do nível cognitivo e as regras do nível instintivo são componentes de cada um dos jogadores.

A técnica *Assume – Guarantee* foi utilizada na especificação e verificação das regras cognitivas e instintivas de cada grupo de jogadores. Isto se deu pela especificação das regras cognitivas e verificação de que estas satisfariam determinadas propriedades, ao se assumir que as regras instintivas iriam se comportar de uma determinada maneira.

Um exemplo desta situação é a garantia de que todos os estados do autômato das regras cognitivas seriam alcançados, assumindo que existem, no

conjunto de regras instintivas, regras que promoveriam modificações de valor da variável *local_goal_status* para cada valor da variável *local_goal_current*.

Esta mesma abordagem foi adotada, posteriormente, no nível instintivo ao se assumir que o ambiente e o nível reativo (conjuntamente) apresentariam informações sobre o jogo suficientes para garantir, por exemplo, que todos os estados dos autômatos que compõem o nível instintivo seriam alcançados.

Os exemplos apresentados acima foram empregados nas etapas 2 e 3 do método de verificação, as quais se caracterizaram pela verificação individual dos planos de cada jogador.

A técnica *Assume – Guarantee* foi também utilizada na etapa 4 para especificação e verificação do ambiente. Nesta etapa foi adotada uma abordagem *bottom-up* e *top-down*. A abordagem *bottom-up* se deu assumindo que se os autômatos que compõem os planos dos agentes se comunicassem com o Soccerserver de forma assíncrona, poderia se garantir que o ambiente promoveria: as transições entre todos os seus estados, as modificações no estado do jogo, a atuação da equipe adversária e a resposta síncrona do tipo *broadcast* para todos os jogadores da equipe verificada. A abordagem *top-down* considerou que, em se assumindo que o Soccerserver apresentaria informações suficientes sobre o estado do jogo para cada jogador, de forma independente, poderia se garantir que os autômatos que compõem as regras instintivas e cognitivas de cada jogador poderiam alcançar todos os estados de cada uma das suas regras.

As técnicas *Partitioned relation transition* e *Lazy parallel composition* foram utilizadas na especificação e verificação dos planos dos jogadores e da equipe.

A utilização destas técnicas nos jogadores se deu pela especificação e verificação das regras instintivas de cada agente. Como estas regras foram modeladas por autômatos individuais e o plano instintivo de cada jogador é composto de um conjunto destes tipos de regras, as quais foram modeladas como autômatos individuais, o resultado da verificação se deu pela composição paralela destes autômatos durante a verificação de cada jogador. Assim, foi possível verificar gradativamente partes do sistema, sem a necessidade de se especificar inicialmente todas as suas transições.

O que diferencia a aplicação das duas técnicas é o fato que, por definição, a técnica *Lazy parallel composition* possui um conjunto de restrições

comum para todos os componentes paralelizados. No caso da verificação dos jogadores, considerando que o comportamento das regras instintivas de um jogador fica condicionado ao conjunto de regras cognitivas do respectivo agente, pode ser dito que este conjunto de regras é restritor das regras instintivas, justificando assim, o emprego da referida técnica, conforme visto nas etapas 2 e 3 do método de verificação.

No nível da equipe, cada jogador possui um conjunto de regras próprio, composto de regras instintivas e cognitivas. Assim, como o ocorrido na etapa 5 do método de verificação, quando foram verificados os planos de um grupo de jogadores ao mesmo tempo e estes estavam condicionados a sua interação com o ambiente, estavam sendo aplicados os conceitos de *Lazy parallel composition*.

A técnica *Interface process* foi empregada na especificação somente de variáveis que fossem relevantes para a comunicação entre os componentes do sistema. Esta técnica foi utilizada em todas as etapas para minimizar o sistema eliminando eventos que não estivessem relacionados com a comunicação. Como exemplo, pode ser referenciado que nas etapas 2 e 3 de verificação individual dos jogadores não foram representados explicitamente os eventos de comunicação dos jogadores com o ambiente e nas etapas 4 e etapa 5 não foi representada explicitamente a comunicação do *Soccerserver* com a equipe adversária.

O emprego das técnicas de verificação composicional foram de fundamental importância para redução do modelo do sistema. Como a aplicação destas técnicas se deu durante a execução de cada etapa será possível identificar tal emprego nos capítulos que descrevem a verificação de cada uma destas etapas.

6. VERIFICAÇÃO DOS PLANOS DOS JOGADORES

Neste capítulo, são apresentados os procedimentos adotados para a execução das verificações e as verificações propriamente ditas dos planos dos agentes, com respeito aos jogadores individuais. São também descritos alguns problemas encontrados e as respectivas soluções adotadas.

A etapa de verificação das regras individuais dos agentes é dividida em duas subetapas: verificação dos jogadores que possuem o mesmo tipo de plano cognitivo; e verificação de cada jogador individualmente.

6.1. VERIFICAÇÃO DOS PLANOS POR TIPOS DE JOGADORES

A verificação dos jogadores por tipo levou em consideração um jogador de cada grupo, uma vez que os jogadores de um mesmo grupo possuem as mesmas regras cognitivas e instintivas. A tabela 9 identifica as regras instintivas comuns aos jogadores de um mesmo grupo e a tabela 10 apresenta os grupos de jogadores que têm o mesmo plano cognitivo, definidos nas seções 4.2.1 e 5.2.1, ou seja: Grupo Goleiro (jogador 1), Grupo Defesa (jogadores 2, 3, 4 e 5) e Grupo Meio de Campo – Ataque (jogadores 6, 7, 8, 9, 10 e 11).

Tabela 9. Resumo das regras cognitivas comuns por tipos de jogadores.

Nr	Regras Cognitivas	Jogadores											Total de jogadores por regras		
		1	2	3	4	5	6	7	8	9	10	11			
1	Cognitive_Keeper	x													1
2	Cognitive_Defender		x	x	x	x									4
3	Cognitive_Mid_Attack						x	x	x	x	x	x	x		6

Na figura 27 estão apresentados os nomes dos autômatos especificados no UPPAAL que correspondem às regras de cada grupo de jogadores. Estes autômatos são instanciados no UPPAAL por processos correspondentes a jogadores diferentes. Por exemplo, P1_MHB(1) é um processo que instancia o *template Mark_Hold_Ball*, passando como parâmetro o valor 1 que identifica o jogador 1. Os jogadores 2 e 7 também instanciam *Mark_Hold_Ball*, com *Mark_Hold_Ball(2)* e *Mark_Hold_Ball(7)*, respectivamente.

Tabela 10. Resumo das regras instintivas comuns por tipos de jogadores.

Nr	Regras Instintivas	Jogadores											Total de jogadores por regras
		1	2	3	4	5	6	7	8	9	10	11	
1	rule_mark_hold_ball	x	x	x	x	x	x	x	x	x	x	x	11
2	rule_mark_search_ball	x	x	x	x	x	x	x	x	x	x	x	11
3	rule_mark_opponent	x	x	x	x	x							5
4	rule_mark_interceptBall	x	x	x	x	x	x	x	x	x	x	x	11
5	rule_mark_strategic_position						x	x	x	x	x	x	6
6	rule_mark_achieved	x	x	x	x	x	x	x	x	x	x	x	11
7	rule_advance_search_ball		x	x	x	x							4
8	rule_advance_interceptBall	x	x	x	x	x							5
9	rule_advance_strategic_position	x	x	x	x	x							5
10	rule_advance_pass_ball_foward	x	x	x	x								4
...													
16	rule_advance_fail	x	x	x	x	x							5
17	rule_advance_achieved	x			x								2
18	rule_side_attack_search_ball	x			x		x	x	x	x	x	x	8
19	rule_side_attack_interceptBall	x			x		x	x	x	x	x	x	8
20	rule_mark_markOpponent	x			x								2
21	rule_side_attack_pass_ball_7	x			x								2
...													
31	rule_side_attack_fail	x			x		x	x	x	x	x	x	8
32	rule_side_attack_achieved	x			x								2
33	rule_ending_kick_ball	x											1
34	rule_ending_search_ball	x											1
35	rule_ending_interceptBall	x											1
36	rule_ending_fail	x											1
37	rule_ending_achieved	x											1

```

// Place template instantiations here.

//-----P1-----
// Cognitive
P1_C = Cognitive_Keeper(1);
// Instintive
P1_MHB = Mark_Hold_Ball(1); // Mark
P1_MSB = Mark_Search_Ball(1);
P1_MMO = Mark_Mark_Opponent(1);
P1_MIB = Mark_Intercept_Ball(1);
P1_MA = Mark_Achieved(1);
P1_AIB = Advance_Intercept_Ball(1); // Advance
P1_ASP = Advance_Strategic_Position(1);
P1_ADFW = Advance_Pass_Ball_Foward(1);
P1_AF = Advance_Fail(1);
P1_AA = Advance_Achieved(1);
P1_SASB = Side_Attack_Search_Ball(1); // Side Attack
P1_SAIIB = Side_Attack_Intercept_Ball(1);
P1_SAMO = Side_Attack_Mark_Opponent(1);
P1_SAF = Side_Attack_Fail(1);
P1_SAA = Side_Attack_Achieved(1);
P1_EKB = Ending_Kick_Ball(1); // Ending
P1_ESB = Ending_Search_Ball(1);
P1_EIB = Ending_Intercept_Ball(1);
P1_EF = Ending_Fail(1);
P1_EA = Ending_Achieved(1);
//-----P2-----
P2
// Cognitive
P2_C = Cognitive_Defender(2);
// Instintive
P2_MHB = Mark_Hold_Ball(2); // Mark
P2_MSB = Mark_Search_Ball(2);
P2_MMO = Mark_Mark_Opponent(2);
P2_MIB = Mark_Intercept_Ball(2);
P2_MA = Mark_Achieved(1);
P2_ASB = Advance_Search_Ball(2); // Advance
P2_AIB = Advance_Intercept_Ball(2);
P2_ASP = Advance_Strategic_Position(2);
P2_AF = Advance_Fail(2);
//-----P7-----
P7
// Cognitive
P7_C = Cognitive_Mid_Attack(7);
// Instintive
P7_MHB = Mark_Hold_Ball(7); // Mark
P7_MSB = Mark_Search_Ball(7);
P7_MIB = Mark_Intercept_Ball(7);
P7_SAF = Mark_Strategic_Position(7);
P7_MA = Mark_Achieved(7);
P7_SASB = Side_Attack_Search_Ball(7); // Side Attack
P7_SAIIB = Side_Attack_Intercept_Ball(7);
P7_SAF = Side_Attack_Fail(7);
//-----P7-----

```

Figura 27. Instância das regras de um jogador de cada grupo no UPPAAL.

6.1.1. Identificação das propriedades verificadas

Para a verificação do sistema foram consideradas propriedades *safety* e *liveness*. Dentre as propriedades *safety*, foi verificada a ausência de *deadlocks* e/ ou *livelocks*. Já em relação às propriedades *liveness*, foi verificada a alcançabilidade de todos os estados das regras cognitivas e instintivas.

Tabela 11. Descrição e fórmulas em CTL das propriedades verificadas.

Nr	Descrição	Fórmula / Tipo
1	Para todos os caminhos nunca ocorre <i>deadlock</i> .	$A[] \text{ not deadlock}$ Safety
2	Para todos os caminhos nunca ocorre que o valor da variável estado do jogador seja diferente de X e o estado do processo cognitivo equivalente ao estado esperado para o valor da variável estado do jogador seja igual a X.	$A[] \text{ not (LocalGoalCurrent[pNR] != X and Pn_C.Estado)}$ Safety Ex: $A[] \text{ not (LocalGoalCurrent[1] != 0 and P1_C.Mark)}$ $X = 0 \Leftrightarrow \text{Mark} \mid \dots \mid 4 \Leftrightarrow \text{Ending}; \text{pNR} = 1 \mid 2 \mid 7 \text{ e } n = 1 \mid 2 \mid 7$
3	Existirá, pelo menos, um caminho em que um determinado estado X seja alcançado a partir do estado inicial.	$E<> \text{ Pn_Regra.Estado}$ Ex: $E<> \text{ P1_C.Side_Attack}$ Liveness $n = 1 \mid 2 \mid 7$

A tabela 11 apresenta as propriedades verificadas e as respectivas fórmulas em TCTL. Das propriedades definidas na tabela, as de número 1 e 2 são relacionadas às regras cognitivas. A propriedade 3, que permite a verificação do alcance dos estados, pode ser aplicada tanto para verificação das regras cognitivas quanto das instintivas.

6.1.2. Verificação das regras do grupo goleiro

Em relação ao goleiro, todas as suas regras cognitivas e instintivas foram verificadas, com exceção da regra *rule_side_attack_pass_ball_7* que demandava interação com outros jogadores, conforme definição na seção 5.2.2.

Nesta verificação foi constatada a existência de *deadlock* (linha 1 da tabela), o qual está relatado na seção 6.1.5.. Em relação à alcançabilidade (tabela 12), tanto os estados cognitivos quanto os instintivos foram demonstrados como alcançáveis.

Tabela 12. Resultado das verificações das propriedades das regras cognitivas do jogador1 (grupo goleiro).

Nr	Fórmula / Tipo	Resultado
1	A[] not deadlock Safety	A[] not deadlock Verification/kernel/elapsed time used: 0,157s / 0s / 0,156s. Resident/virtual memory usage peaks: 8.580KB / 31.020KB. Property is not satisfied.
2	A[] not (LocalGoalCurrent[pNR] != X and Pn_C.Estado) Safety	
	2.1. A[] not (LocalGoalCurrent[1] != 0 and P1_C.Mark)	A[] not (LocalGoalCurrent[1] != 0 and P1_C.Mark) Verification/kernel/elapsed time used: 0,109s / 0s / 0,125s. Resident/virtual memory usage peaks: 8.732KB / 31.408KB. Property is not satisfied.
	2.2. A[] not (LocalGoalCurrent[1] != 1 and P1_C.Advance)	A[] not (LocalGoalCurrent[1] != 1 and P1_C.Advance) Verification/kernel/elapsed time used: 13,61s / 0,062s / 13,719s. Resident/virtual memory usage peaks: 10.508KB / 34.772KB. Property is satisfied.
	2.3. A[] not (LocalGoalCurrent[1] != 2 and P1_C.Side_Attack)	A[] not (LocalGoalCurrent[1] != 2 and P1_C.Side_Attack) Verification/kernel/elapsed time used: 2,547s / 0s / 2,547s. Resident/virtual memory usage peaks: 10.500KB / 34.772KB. Property is satisfied.
	2.4. A[] not (LocalGoalCurrent[1] != 3 and P1_C.Ending)	A[] not (LocalGoalCurrent[1] != 3 and P1_C.Ending) Verification/kernel/elapsed time used: 2,468s / 0s / 2,5s. Resident/virtual memory usage peaks: 10.508KB / 34.772KB. Property is satisfied.
	2.5. A[] not (LocalGoalCurrent[1] != 4 and P1_C.None)	A[] not (LocalGoalCurrent[1] != 4 and P1_C.None) Verification/kernel/elapsed time used: 2,516s / 0s / 2,563s. Resident/virtual memory usage peaks: 10.508KB / 34.772KB. Property is satisfied.
3	E<> Pn_Regra.Estado Liveness	
	3.1. E<> P1_C.Mark	E<> P1_C.Mark Verification/kernel/elapsed time used: 0,032s / 0s / 0,031s. Resident/virtual memory usage peaks: 10.816KB / 35.924KB. Property is satisfied.
	3.2. E<> P1_C.Advance	E<> P1_C.Advance Verification/kernel/elapsed time used: 0,14s / 0s / 0,14s. Resident/virtual memory usage peaks: 11.164KB / 36.568KB. Property is satisfied.
	3.3. E<> P1_C.Side_Attack	E<> P1_C.Side_Attack Verification/kernel/elapsed time used: 0,672s / 0,015s / 0,703s. Resident/virtual memory usage peaks: 20.148KB / 55.820KB. Property is satisfied.
	3.4. E<> P1_C.Ending	E<> P1_C.Ending Verification/kernel/elapsed time used: 0,313s / 0s / 0,312s. Resident/virtual memory usage peaks: 20.176KB / 55.868KB. Property is satisfied.
	3.5. E<> P1_C.None	Não foi verificada por ser trivial. Se None é estado inicial, logo, obrigatoriamente é alcançável.

Tabela 13. Propriedades de alcançabilidade de estados das regras instintivas do jogador 1 verificadas e satisfeitas.

3	E<> Pn_Regra.Estado Liveness	
	3.6. E<> P1_MHB.hold_ball	3.16. E<> P1_SASB.search_ball
	3.7. E<> P1_MSB.search_ball	3.17. E<> P1_SAIB.intercept_ball
	3.8. E<> P1_MMO.mark_opponent	3.18. E<> P1_SAMO.mark_opponent
	3.9. E<> P1_MIB.intercept_ball	3.19. E<> P1_SAF.Set_Cognitive_Failed
	3.10. E<> P1_MA.Set_Cognitive_Achieved	3.20. E<> P1_SAA.Set_Cognitive_Achieved
	3.11. E<> P1_AIB.intercept_ball	3.21. E<> P1_EKB.kick_to_goal
	3.12. E<> P1_ASP.strategic_position	3.22. E<> P1_ESB.search_ball
	3.13. E<> P1_ADBFW.pass_ball_forward	3.23. E<> P1_EIB.intercept_ball
	3.14. E<> P1_AF.Set_Cognitive_Failed	3.24. E<> P1_EF.Set_Cognitive_Failed
	3.15. E<> P1_AA.Set_Cognitive_Achieved	3.25. E<> P1_EA.Set_Cognitive_Achieved

A tabela 13 apresenta, de forma resumida, a relação de propriedades verificadas e satisfeitas.

6.1.3. Verificação das regras do grupo jogadores de defesa

Os resultados da verificação das regras dos jogadores de defesa, usando como representante do grupo do jogador 2, estão apresentados na tabela .

A propriedade especificada na linha 1 da tabela 14 não foi satisfeita, devido a um erro na regra *rule_mark_hold Ball*, que está reportado na seção 6.1.5.

Outra propriedade não satisfeita foi a propriedade de alcançabilidade do estado *Side_Attack* das regras cognitivas (vide propriedade 3.3.). Isto é devido às regras comuns dos jogadores de defesa não possuírem a regra *Advance_Achieved* do nível instintivo que mudam o estado futuro dos jogadores para *Side_Attack*.

Todas as regras instintivas são alcançáveis, conforme resumo apresentado na tabela 15.

Tabela 14. Resultado das verificações das propriedades das regras cognitivas do jogador2 (grupo defesa).

Nr	Fórmula / Tipo	Resultado
1	A[] not deadlock Safety	A[] not deadlock Verification/kernel/elapsed time used: 0,203s / 0s / 0,203s. Resident/virtual memory usage peaks: 6.748KB / 27.008KB. Property is not satisfied.
2	A[] not (LocalGoalCurrent[pNR] != X and Pn_C.Estado) Safety	
	2.1. A[] not (LocalGoalCurrent[2] != 0 and P2_C.Mark)	A[] not (LocalGoalCurrent[2] != 0 and P2_C.Mark) Verification/kernel/elapsed time used: 0,141s / 0s / 0,187s. Resident/virtual memory usage peaks: 6.960KB / 27.444KB. Property is not satisfied.
	2.2. A[] not (LocalGoalCurrent[2] != 1 and P2_C.Advance)	A[] not (LocalGoalCurrent[2] != 1 and P2_C.Advance) Verification/kernel/elapsed time used: 2,906s / 0,015s / 2,969s. Resident/virtual memory usage peaks: 7.244KB / 27.972KB. Property is satisfied.
	2.3. A[] not (LocalGoalCurrent[2] != 2 and P2_C.Side_Attack)	A[] not (LocalGoalCurrent[2] != 2 and P2_C.Side_Attack) Verification/kernel/elapsed time used: 0,547s / 0s / 0,593s. Resident/virtual memory usage peaks: 7.244KB / 27.972KB. Property is satisfied.
	2.5. A[] not (LocalGoalCurrent[2] != 4 and P2_C.None)	A[] not (LocalGoalCurrent[2] != 4 and P2_C.None) Verification/kernel/elapsed time used: 0,531s / 0s / 0,578s. Resident/virtual memory usage peaks: 7.244KB / 27.972KB. Property is satisfied.
3	E<> Pn_Regra.Estado Liveness	
	3.1. E<> P2_C.Mark	E<> P2_C.Mark Verification/kernel/elapsed time used: 0,016s / 0s / 0,016s. Resident/virtual memory usage peaks: 7.244KB / 27.972KB. Property is satisfied.
	3.2. E<> P2_C.Advance	E<> P2_C.Advance Verification/kernel/elapsed time used: 0,031s / 0s / 0,032s. Resident/virtual memory usage peaks: 7.244KB / 27.972KB. Property is satisfied.
	3.3. E<> P2_C.Side_Attack	E<> P2_C.Side_Attack Verification/kernel/elapsed time used: 56,625s / 0s / 56,781s. Resident/virtual memory usage peaks: 29.476KB / 75.796KB. Property is not satisfied.
	3.5. E<> P2_C.None	Não foi verificada por ser trivial. Se None é estado inicial, logo, obrigatoriamente é alcançável.

Tabela 15. Propriedades de alcançabilidade de estados das regras instintivas do jogador 2 verificadas e satisfeitas.

3	E<> Pn_Regra.Estado Liveness	
	3.6. E<> P2_MHB.hold_ball	3.10. E<> P2_MA.Set_Cognitive_Achieved
	3.7. E<> P2_MSB.search_ball	3.11. E<> P2_AIB.intercept_ball
	3.8. E<> P2_MMO.mark_opponent	3.12. E<> P2_ASP.strategic_position
	3.9. E<> P2_MIB.intercept_ball	3.14. E<> P2_AF.Set_Cognitive_Failed

6.1.4. Verificação das regras do grupo jogadores de meio de campo - ataque

Os resultados da verificação das regras dos jogadores de meio de campo - ataque, usando como representante o jogador 7, estão apresentados na tabela 16.

Na verificação das regras cognitivas do jogador 7 (grupo meio de campo – ataque) foram encontrados os mesmos resultados para a verificação das propriedades para as fórmulas 1 e 2.1. dos jogadores 1 e 2. Estes erros vinculados à regra *mark_hold_ball* estão reportados na seção 6.1.5..

Todas as propriedades de alcançabilidade de estados cognitivos e instintivos foram satisfeitas, conforme apresentado nas tabelas 16 e 17, respectivamente.

Tabela 16. Resultado das verificações das propriedades das regras cognitivas do jogador7 (grupo meio de campo - ataque).

Nr	Fórmula / Tipo	Resultado
1	A[] not deadlock Safety	A[] not deadlock Verification/kernel/elapsed time used: 0,078s / 0s / 0,079s. Resident/virtual memory usage peaks: 7.072KB / 27.616KB. <i>Property is not satisfied.</i>
2	A[] not (LocalGoalCurrent[pNR] != X and Pn_C.Estado) Safety	
	2.1. A[] not (LocalGoalCurrent[7] != 0 and P7_C.Mark)	A[] not (LocalGoalCurrent[7] != 0 and P7_C.Mark) Verification/kernel/elapsed time used: 0,094s / 0s / 0,094s. Resident/virtual memory usage peaks: 7.220KB / 27.948KB. <i>Property is not satisfied.</i>
	2.3. A[] not (LocalGoalCurrent[7] != 2 and P7_C.Side_Attack)	A[] not (LocalGoalCurrent[7] != 2 and P7_C.Side_Attack) Verification/kernel/elapsed time used: 3,984s / 0,015s / 4s. Resident/virtual memory usage peaks: 7.520KB / 28.436KB. <i>Property is satisfied.</i>
	2.5. A[] not (LocalGoalCurrent[7] != 4 and P7_C.None)	A[] not (LocalGoalCurrent[7] != 4 and P7_C.None) Verification/kernel/elapsed time used: 0,813s / 0s / 0,828s. Resident/virtual memory usage peaks: 7.548KB / 28.464KB. <i>Property is satisfied.</i>
3	E<> Pn_Regra.Estado Liveness	
	3.1. E<> P7_C.Mark	E<> P7_C.Mark Verification/kernel/elapsed time used: 0s / 0s / 0,016s. Resident/virtual memory usage peaks: 7.548KB / 28.464KB. <i>Property is satisfied.</i>
	3.3. E<> P7_C.Side_Attack	E<> P7_C.Side_Attack Verification/kernel/elapsed time used: 0,015s / 0s / 0,016s. Resident/virtual memory usage peaks: 7.548KB / 28.464KB. <i>Property is satisfied.</i>
	3.5. E<> P7_C.None	Não foi verificada por ser trivial. Se None é estado inicial, logo, obrigatoriamente, é alcançável.

Tabela 17. Propriedades de alcançabilidade de estados das regras instintivas do jogador 7 verificadas e satisfeitas.

3	E<> Pn_Regra.Estado Liveness	
	3.6. E<> P7_MHB.hold_ball	3.10. E<> P7_MA.Set_Cognitive_Achieved
	3.7. E<> P7_MSB.search_ball	3.11. E<> P7_SAlB.intercept_ball
	3.8. E<> P7_MSP.strategic_position	3.12. E<> P7_SASB.search_ball
	3.9. E<> P7_MIB.intercept_ball	3.14. E<> P7_SAF.Set_Cognitive_Failed

A tabela 17 apresenta, de forma resumida, a relação de propriedades verificadas e satisfeitas.

6.1.5. Ajustes das regras e resolução de problemas encontrados na verificação dos grupos de jogadores

As seguintes regras instintivas causaram problemas na verificação dos grupos de jogadores: a) *rule_mark_hold_ball* e b) *rule_ending_achieved*.

Para continuar com a verificação foi necessário analisar o problema de cada regra e proceder ajustes nas mesmas.

O problema com a regra *rule_mark_hold_ball* foi identificado ao se analisar as regras instintivas dos jogadores ainda durante a modelagem das mesmas. Este problema estava vinculado à existência de uma inconsistência na regra instintiva *rule_mark_hold_ball* que conduzia todos os jogadores para um estado (*local_goal current = kick_to_goal*) não encontrado nas regras cognitivas (vide quadro 14).

```
(rule_mark_hold_ball
  (if (logic ( local_goal current mark ))
      (logic ( local_goal status active ))
      (logic ( player ball_kickable true ))
    )
  (then (logic ( reactive_behavior active hold_ball ))
        (logic ( local_goal status achieved ))
        (logic ( local_goal current kick_to_goal ))
      )
  )
)
```

Quadro 14. Destaque da mudança de estado local do jogador via regra *rule_mark_hold_ball*.

Mesmo havendo indícios de que essa inconsistência implicava em situações indesejadas para os jogadores e a equipe, ficou definido que, para manter a fidelidade do modelo com o sistema, não seria criado um novo estado nos autômatos correspondentes às regras cognitivas, pois as tais regras não prevêm este estado. Desta forma, foi acrescentado ao conjunto de valores possíveis para a variável *local_goal current* o valor *kick_to_goal*, o qual foi representada com o valor inteiro 5 na especificação dos autômatos da regras instintivas.

As figuras 28 e 29 apresentam as convenções e modificações adotadas para essa situação.

```

Editor | Simulator | Verifier
Drag out
Project
  Declarations
    Cognitive_Keeper
    Cognitive_Defender
    Cognitive_Mid_Attack
    Mark_Hold_Ball
    Mark_Search_Ball
    Mark_Mark_Opponent
  Mark_Change_Position
// Place global declarations here.
// Cognitive Layer's Variables
int LocalGoalCurrent [12] = {-1,4,4,4,4,4,4,4,4,4,4,4}; /* Mark = 0, Advance = 1,
Side Attack = 2, Ending = 3,
None = 4, Kick_to_Goal = 5 */
int LocalGoalStatus [12] = {-1,3,3,3,3,3,3,3,3,3,3,3}; /* Fail = 0, Active = 1, Achieved = 2,
None = 3 */

```

Figura 28. Alteração das informações sobre os possíveis valores da variável *local_goal current*

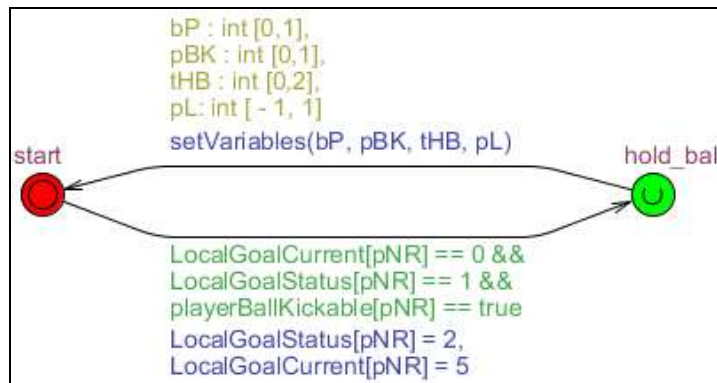


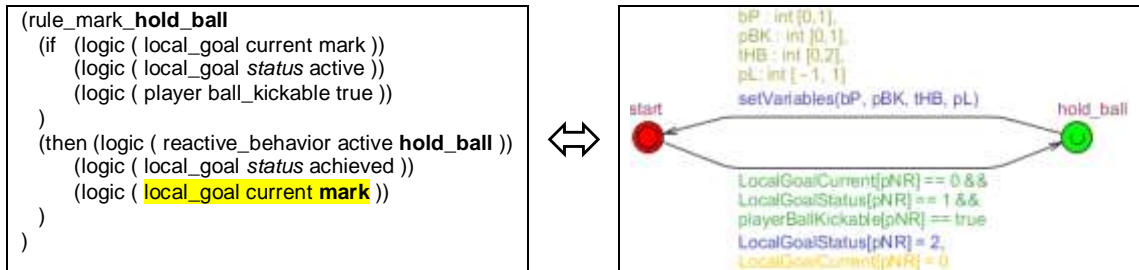
Figura 29. Especificação da regras *Mark_Hold_Ball* considerando o novo estado *kick_to_goal* ⇔ *LocalGoalCurrent[pNR] = 5*.

A figura 29 apresenta o autômato que representa a regra *rule_mark_hold_ball*, o qual foi especificado considerando o valor de *LocalGoalCurrent[pNR]* igual a 5 para a situação *kick_to_goal* vista no quadro 14.

De acordo com o que foi descrito na verificação das regras dos grupos de jogadores, a regra *rule_mark_hold_ball* fez com que todos os grupos de jogadores apresentassem a possibilidade de ocorrência de *deadlocks*. Esta situação impacta o sistema como um todo, pois isto inevitavelmente acarretará em um *deadlock* de todo o sistema, tendo em vista que em algum momento todas as variáveis *local_goal current* estarão com valor 5 e nas regras cognitivas não existem transições para as variáveis com este valor.

Para dar continuidade às próximas etapas da verificação, foi procedida uma modificação na referida regra tendo como objetivo não alterar o conceito de atuação dos jogadores e da equipe. Assim, considerando que o nome da regra faz referência à palavra *mark* e que na regra é acionado o comportamento reativo *hold_ball* (segurar a bola), o valor *kick_to_goal* foi removido, e na parte direita da

regra, o estado corrente do objetivo local passou a ser igual a *mark*, ou seja, o valor das variáveis *local_goal current* no autômato correspondente passou a ser 2, como representado no quadro 15.



Quadro 15. Relação entre a alteração da regra *rule_mark_hold_ball* e o autômato equivalente especificado no UPPAAL.

Depois de realizada a alteração, os problemas identificados para as verificações das regras 2.1. de todos os jogadores (vide tabelas 9, 11 e 13) não ocorreram mais, e as propriedades foram satisfeitas, conforme apresentado na tabela 18.

Tabela 18. Resultado da verificação de ocorrência de inconsistência entre os estados dos jogadores e o valor da variável *LocalGoalCurrent[]*, após correção da regra instintiva *rule_mark_hold_ball*.

Nr	Fórmula / Tipo	Grupo	Resultado
2	A[] not (LocalGoalCurrent[pNR] != X and Pn_C.Estado)		Safety
	2.1. A[] not (LocalGoalCurrent[1] != 0 and P1_C.Mark)	Goleiro / 1	A[] not (LocalGoalCurrent[1] != 0 and P1_C.Mark) Verification/kernel/elapsed time used: 24,375s / 0,047s / 24,547s. Resident/virtual memory usage peaks: 11,448KB / 36,508KB. Property is satisfied.
	2.1. A[] not (LocalGoalCurrent[2] != 0 and P2_C.Mark)	Defesa / 2	A[] not (LocalGoalCurrent[2] != 0 and P2_C.Mark) Verification/kernel/elapsed time used: 21,234s / 0,051s / 21,297s. Resident/virtual memory usage peaks: 10,248KB / 33,932KB. Property is satisfied.
	2.1. A[] not (LocalGoalCurrent[7] != 0 and P7_C.Mark)	Meio - Ataque / 7	A[] not (LocalGoalCurrent[7] != 0 and P7_C.Mark) Verification/kernel/elapsed time used: 9,963s / 0,015s / 9,994s. Resident/virtual memory usage peaks: 8,032KB / 29,504KB. Property is satisfied.

É importante destacar que a mudança da regra *rule_mark_hold_ball* não afetou o comportamento dos jogadores para as outras propriedades verificadas, de acordo com novas verificações realizadas.

Contudo, as regras verificadas via primeira fórmula das tabelas (A[] not deadlock), de todos os jogadores só obtiveram resultado satisfatório para os jogadores 2 e 7, pois o jogador 1 ainda apresentou a ocorrência de *deadlock* (vide tabela 19).

Foi identificado que a razão deste *deadlock* se encontrava na regra *rule_ending_achieved* (especificada no UPPAAL como P1_EA).

Tabela 19. Resultado da verificação de ocorrência de deadlocks, após correção da regra instintiva *rule_mark_hold_ball*.

Nr	Fórmula / Tipo	Grupo/Jogador	Resultado
1	A[] not deadlock Safety	Goleiro / 1	A[] not deadlock Verification(kernel)/elapsed time used: 29,407s / 0,062s / 29,547s. Resident/virtual memory usage peaks: 45.416KB / 109.656KB. Property is not satisfied.
		Defesa / 2	A[] not deadlock Verification(kernel)/elapsed time used: 99,5s / 0,172s / 100,062s. Resident/virtual memory usage peaks: 49.348KB / 118.064KB. Property is satisfied.
		Meio - Ataque / 7	A[] not deadlock Verification(kernel)/elapsed time used: 42,438s / 0,062s / 42,672s. Resident/virtual memory usage peaks: 36.988KB / 91.936KB. Property is satisfied.

A regra *rule_ending_achieved* (vide quadro 16) é uma regra instintiva vinculada ao estado *Ending* do plano cognitivo do jogador 1. Esta regra só é utilizada pelo jogador 1, o que, objetivamente, na verificação do sistema como um todo, só implica no *deadlock* do goleiro, podendo o restante da equipe continuar a ser verificada sem ser afetada por esta regra.

Entretanto, considerando o comportamento global da equipe e não dos jogadores individualmente, no ambiente de jogo e não de verificação, tal fato poderia contribuir negativamente para os objetivos coletivos, pois o goleiro é um elemento crítico para a equipe.

Esta regra possui uma descrição totalmente diferente das demais, pois aciona um comportamento do nível reativo (*reactive_behavior active kick_to_goal*) que não é comum para uma regra que altera o *status* do estado cognitivo (regra do tipo *fail* e *achieved*). Além disso, nas demais regras o acionamento dos comportamentos do nível reativo são descritos no lado direito das suas sentenças e não no lado esquerdo como ocorre neste caso (vide quadro 16 e regras do Apêndice A).

Portanto, considerando o exposto acima, a solução encontrada para este problema foi a de eliminar a referida regra do conjunto de regras do grupo goleiro (jogador 1). A remoção desta regra foi efetuada considerando que isto não traria impacto para o jogo.

Curiosamente, após as medidas para solucionar os problemas de *deadlocks* mencionados, a propriedade não pode ser verificada porque o verificador não conseguia finalizar tal tarefa. Analisando esta situação, foi identificado via MSC (*Message Sequence Chart*) do simulador do UPPAAL a existência de *livelock* para os estados *Side_Attack* e *Ending*, para esta regras.

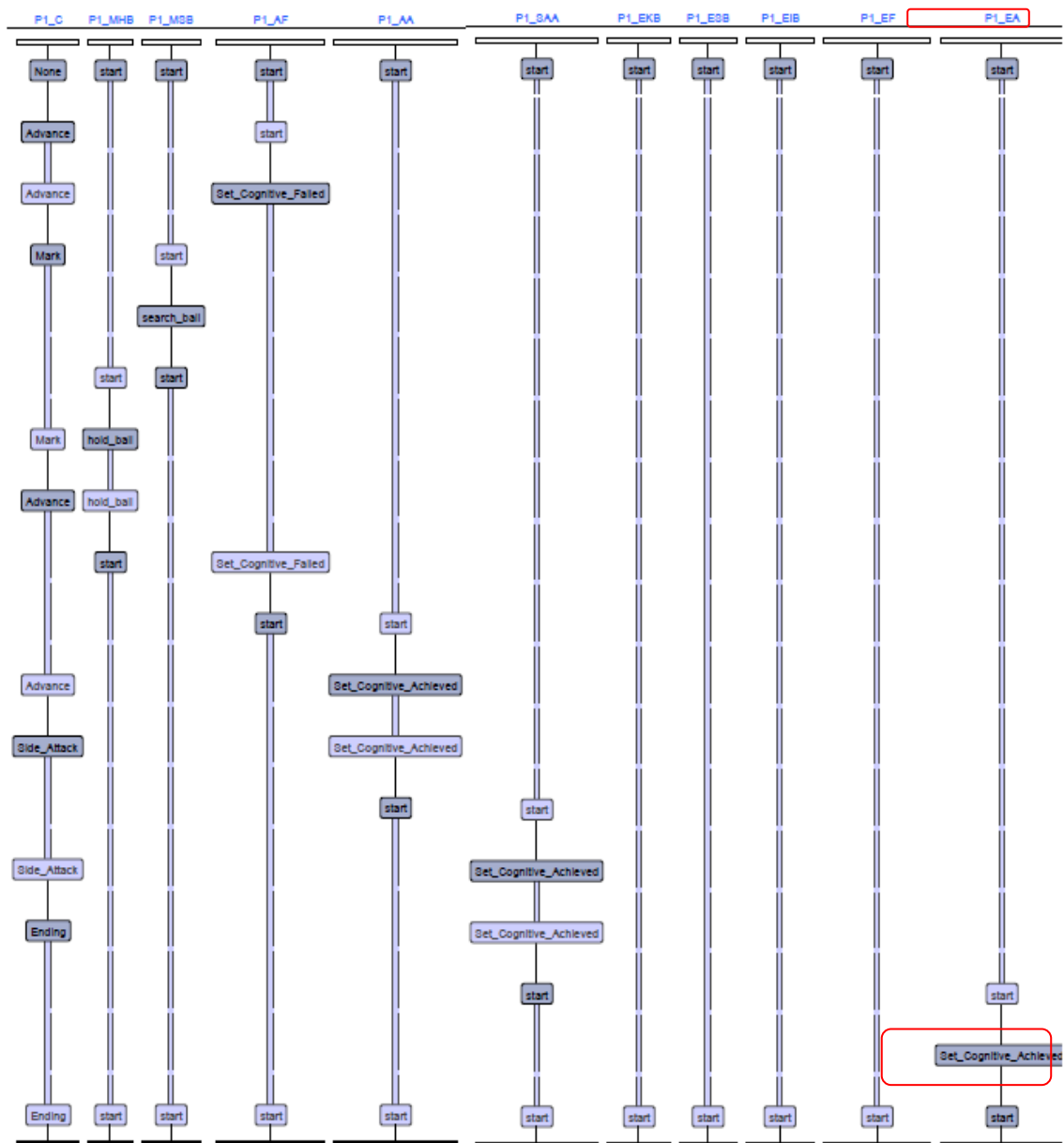


Figura 30. Caminho executado pela regras até o deadlock provocado pela regra *rule_ending_achieved* representada pelo *Message Sequence Chart* (MSC) do UPPPAAL.

```
(rule_ending_achieved
  (if (logic ( local_goal current ending ))
      (logic ( local_goal status active ))
      (logic ( reactive_behavior active kick_to_goal ))
    )
  (then (logic ( local_goal status achieved ))
        (logic ( local_goal current ending ))
    )
  )
)
```

Quadro 16. Regra *rule_ending_achieved* utilizada pelo goleiro

Nas duas situações o jogador 1 entrou nos respectivos estados do nível cognitivo e não existiam regras do nível instintivo em condições de promover a evolução do sistema do agente para outros estados, em nenhuma das configurações do ambiente.

Isto ocorreu pelo fato do agente estar sendo analisado isoladamente e não existirem modificações do ambiente promovidas pelos outros jogadores e reportados em momentos posteriores pelo simulador do jogo.

Deste modo, mesmo não sendo livre de *livelock* a verificação pôde continuar a ser realizada.

6.2. VERIFICAÇÃO DE CADA JOGADOR INDIVIDUALMENTE

Esta seção trata da verificação de todos os jogadores individualmente, considerando as regras já verificadas anteriormente e as regras adicionadas nesta etapa.

6.2.1. Verificação das regras do jogador 1

Tendo em vista que quase todas as regras do jogador 1 já tinham sido verificadas, foi necessário incluir somente a regra `rule_side_attack_pass_ball_7` (nomeada como `side_attack_pass_ball_uncond` no UPPAAL) para completar o seu conjunto de regras.

Como resultado, nenhuma regra do tipo *safety* pode ser verificada, devido ao *livelock* detectado anteriormente, mas todas as propriedades *liveness* de alcançabilidade de estados, descritas na seção 6.1., foram satisfeitas, incluindo a propriedade `E<> P1_SAPBU7.pass_ball`, conforme apresenta a figura 31.

```
E<> P1_SAPBU7.pass_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 16.024KB / 50.184KB.
Property is satisfied.
```

Figura 31. Resultado da verificação da propriedade da regra `Side_Attack_Pass_Ball` do jogador 1.

6.2.2. Verificação das regras de cada jogador de defesa

O jogador 2 teve adicionadas ao seu conjunto as seguintes regras com os respectivos modelos e processos no UPPAAL:

- rule_advance_pass_ball_forward ⇔ Advance_Pass_Ball_Forward ⇔ P2_ADBFW;
- rule_advance_pass_ball_7 ⇔ Advance_Pass_Ball_Uncond ⇔ P2_APBU7;
- rule_advance_pass_ball_8 ⇔ Advance_Pass_Ball_Cond ⇔ P2_APBC8.

Procedida a verificação, todas as regras, as já verificadas anteriormente e as recém incluídas, tiveram suas propriedades satisfeitas. A figura 32 apresenta o resultado da verificação das propriedades das regras incluídas para verificação nesta etapa.

```
E<> P2_ADBFW.pass_ball_foward
Verification/kernel/elapsed time used: 0,969s / 0s / 0,984s.
Resident/virtual memory usage peaks: 26.852KB / 68.152KB
Property is satisfied.

E<> P2_APBU7.pass_ball
Verification/kernel/elapsed time used: 0,047s / 0s / 0,078s.
Resident/virtual memory usage peaks: 12.492KB / 40.216KB.
Property is satisfied.

E<> P2_APBC8.pass_ball
Verification/kernel/elapsed time used: 0,062s / 0s / 0,094s.
Resident/virtual memory usage peaks: 18.956KB / 53.112KB.
Property is satisfied.
```

Figura 32. Resultado da verificação da propriedade das regras do jogador 2 inseridas nesta etapa.

Cabe destacar que com a inclusão das regras supramencionadas houve um aumento considerável em termos de tempo de resposta da verificação das propriedades. Como exemplo disto, pode ser destacada a verificação da não ocorrência de *deadlocks* que, na etapa anterior durou, conforme a tabela 18, 99,5 segundos (~1,66 minutos), e nesta etapa demorou 1.699,75 segundos (~28,33 minutos), conforme apresentado na figura 33.

```
A[] not deadlock
Verification/kernel/elapsed time used: 1.699,75s / 3,234s / 1.712,953s.
Resident/virtual memory usage peaks: 26.844KB / 68.144KB.
Property is satisfied.
```

Figura 33. Resultado da verificação da propriedade de não ocorrência de *deadlocks* nas regras do jogador 2.

Foi desnecessário verificar as regras do jogador 3, tendo em vista que as suas regras são um subconjunto das regras do jogador 2, não possuindo, em relação a este último, somente as regras *rule_advance_pass_ball_7* e *rule_advance_pass_ball_8*.

O jogador 4 possui o maior conjunto de regras instintivas dos jogadores de defesa. Considerando este aspecto e os problemas de *overhead* citados anteriormente e a não finalização de algumas verificações, a verificação a partir deste ponto foi realizada utilizando o microcomputador 02, descrito na seção 5.1.

O jogador 4 teve as seguintes regras adicionadas para verificação:

- rule_advance_pass_ball_forward ⇔ Advance_Pass_Ball_Forward ⇔ P4_ADBFW;
- rule_side_attack_search_ball ⇔ Side_Attack_Search_Ball ⇔ P4_SASB;
- rule_side_attack_intercept_ball ⇔ Side_Attack_Intercept_Ball ⇔ P4_SAIB;
- rule_side_attack_mark_opponent ⇔ Side_Attack_Mark_Opponent ⇔ P4_SAMO;
- rule_side_attack_pass_ball_7 ⇔ Side_Attack_Pass_Ball_Uncond ⇔ P4_SAPBU7;
- rule_side_attack_fail ⇔ Side_Attack_Fail ⇔ P4_SAF;
- rule_side_attack_achieved ⇔ Side_Attack_Achieved ⇔ P4_SAA.

```
E<> P4_SAA.Set_Cognitive_Achieved
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 31.712KB / 82.368KB.
Property is satisfied.
E<> P4_SAF.Set_Cognitive_Failed
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 31.720KB / 82.376KB.
Property is satisfied.
E<> P4_SAPBU7.pass_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 31.720KB / 82.376KB.
Property is satisfied.
E<> P4_SAMO.mark_opponent
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 31.720KB / 82.376KB.
Property is satisfied.
E<> P4_SAIB.intercept_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 31.720KB / 82.376KB.
Property is satisfied.
E<> P4_SASB.search_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 31.720KB / 82.376KB.
Property is satisfied.
E<> P4_APBFW.pass_ball_foward
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 31.720KB / 82.376KB.
Property is satisfied.
```

Figura 34. Resultado da verificação das propriedades das regras do jogador 4 inseridas nesta etapa.

Todas as regras acima e as já verificadas anteriormente tiveram suas propriedades satisfeitas. A figura 34 apresenta o resultado da verificação das propriedades das regras incluídas para verificação nesta etapa.

O jogador 5 teve as seguintes regras incorporadas ao seu conjunto de regras:

- rule_advance_pass_ball_6 ⇔ Advance_Pass_Ball_Uncond ⇔ P5_APBU6;
- rule_advance_pass_ball_8 ⇔ Advance_Pass_Ball_Cond ⇔ P5_APBC8.

As propriedades referentes a estas regras foram todas satisfeitas, conforme apresentado na figura 35.

```
E<> P5_APBC8.pass_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 46.684KB / 116.984KB.
Property is satisfied.
E<> P5_APBU6.pass_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 46.684KB / 116.984KB.
Property is satisfied.
```

Figura 35. Resultado da verificação da propriedade das regras do jogador 5 inseridas nesta etapa.

6.2.3. Verificação das regras de cada jogador de meio de campo e ataque

Em relação ao jogador 6, as regras específicas acrescentadas foram:

- rule_advance_drive_ball_forward ⇔ Advance_Drive_Ball_Forward ⇔ P6_A DBFW;
- rule_side_attack_pass_ball_8 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P6_SAPB C8;
- rule_side_attack_pass_ball_9 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P6_SAPB C9;
- rule_side_attack_pass_ball_11 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P6_SAPB C11;
- rule_side_attack_strategic_position_free_mark ⇔ Side_Attack_Strategic_Position_Free_Mark ⇔ P6_SASPFM.

As figuras 36 e 37 apresentam o resultado da verificação das propriedades relativas às regras acima. Destas regras, somente o estado correspondente ao comportamento *reactive_behavior_drive_ball_forward* não foi alcançado e a regra *rule_advance_pass_ball_forward* não foi satisfeita, conforme resultado da verificação da propriedade E<> P6_ADBFW. *drive_ball_forward* (vide

figura 37). Logo, considerando que o estado *drive_ball_forward* não é alcançado, esta regra pode ser descartada do conjunto de regras do jogador 6, pois não é utilizada em nenhum momento pelo referido agente.

```
E<> P6_SASPFM.free_mark
Verification/kernel/elapsed time used: 0,047s / 0s / 0,047s.
Resident/virtual memory usage peaks: 65.272KB / 143.744KB.
Property is satisfied.
E<> P6_SAPBC11.pass_ball
Verification/kernel/elapsed time used: 0,031s / 0s / 0,031s.
Resident/virtual memory usage peaks: 65.272KB / 143.744KB.
Property is satisfied.
E<> P6_SAPBC9.pass_ball
Verification/kernel/elapsed time used: 0,031s / 0s / 0,031s.
Resident/virtual memory usage peaks: 0KB / 0KB.
Property is satisfied.
E<> P6_SAPBC8.pass_ball
Verification/kernel/elapsed time used: 0,015s / 0s / 0,015s.
Resident/virtual memory usage peaks: 65.272KB / 143.744KB.
Property is satisfied.
```

Figura 36. Resultado da verificação da propriedade das regras do jogador 6 inseridas nesta etapa.

```
E<> P6_ADBFW.drive_ball_foward
Verification/kernel/elapsed time used: 6.669,953s / 1,094s / 25.166,328s.
Resident/virtual memory usage peaks: 69.008KB / 153.452KB.
Property is not satisfied.
```

Figura 37. Resultado da verificação da propriedade de alcançabilidade do estado *drive_ball_forward* da regra P6_ADBFW.

Em relação ao jogador 7 as seguintes regras foram incorporadas ao conjunto de regras verificadas anteriormente:

- rule_advance_drive_ball_fast ⇔ Advance_Drive_Ball_Fast ⇔ P7_ADBFS;
- rule_side_attack_pass_ball_8 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P7_SAPB C8;
- rule_side_attack_pass_ball_9 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P7_SAPB C9;
- rule_side_attack_pass_ball_10 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P7_SAPB C10;
- rule_side_attack_strategic_position_free_mark ⇔ Side_Attack_Strategic_Position_Free_Mark ⇔ P7_SASPFM.

As figuras 38 e 39 apresentam o resultado da verificação das propriedades relativas a estas regras. Destas regras, o comportamento *reactive_behavior drive_ball_fast* não é acionado, pois a regra *rule_advance_pass_ball_fast* não é alcançável, conforme resultado da verificação da propriedade da sentença E<> P7_ADBFS. *drive_ball_fast*, apresentado na figura 39. Deste modo, a referida regra foi retirada do conjunto de regras do jogador.

```

E<> P7_SASPFM.free_mark
Verification/kernel/elapsed time used: 0,047s / 0s / 0,047s.
Resident/virtual memory usage peaks: 15.396KB / 44.540KB.
Property is satisfied.
E<> P7_SAPBC10.pass_ball
Verification/kernel/elapsed time used: 0,031s / 0s / 0,031s.
Resident/virtual memory usage peaks: 15.404KB / 44.548KB.
Property is satisfied.
E<> P7_SAPBC9.pass_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 15.436KB / 44.612KB.
Property is satisfied.
E<> P7_SAPBC8.pass_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 15.436KB / 44.612KB.
Property is satisfied.

```

Figura 38. Resultado da verificação da propriedade das regras do jogador 7 inseridas nesta etapa.

```

E<> P7_ADBFS.drive_ball_fast
Established direct connection to local server.
(Academic) UPPAAL version 4.1.2 (rev. 4409), September 2009 -- server.
Verification/kernel/elapsed time used: 6.692,485s / 1s / 6.748,117s.
Resident/virtual memory usage peaks: 54.532KB / 129.692KB.
Property is not satisfied.

```

Figura 39. Resultado da verificação da propriedade de alcançabilidade do estado `drive_ball_forward` da regra do jogador 7 `P7_ADBFS`.

Como pode ser observado na figura 39, a verificação da regra *reactive_behavior drive_ball_fast* consumiu 6.692,485 segundos (~111,54 minutos ou ~1 hora, 51 minutos e 36 segundos).

O jogador 8 teve as seguintes regras incorporadas ao seu conjunto de regras:

- rule_side_attack_pass_ball_9 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P8_SAPB C9;
- rule_side_attack_pass_ball_10 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P8_SAPB C10;
- rule_side_attack_pass_ball_11 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P8_SAPB C11;
- rule_side_attack_drive_ball_forward ⇔ Side_Attack_Drive_Ball_forward ⇔ P8_SADBFW.
- rule_side_attack_strategic_position_free_mark ⇔ Side_Attack_Strategic_Position_Free_Mark ⇔ P8_SASPFM.

No caso deste jogador, também foram satisfeitas todas as propriedades de alcançabilidade de estados, sem alterar os resultados das regras verificadas anteriormente, conforme apresentado na figura 43.

```

E<> P8_SASPFM.free_mark
Verification/kernel/elapsed time used: 0,046s / 0s / 0,047s.
Resident/virtual memory usage peaks: 18.944KB / 51.652KB.
Property is satisfied.
E<> P8_SADBFW.drive_ball_foward
Verification/kernel/elapsed time used: 0,047s / 0s / 0,047s.
Resident/virtual memory usage peaks: 0KB / 0KB.
Property is satisfied.
E<> P8_SAPBC11.pass_ball
Verification/kernel/elapsed time used: 0,047s / 0s / 0,047s.
Resident/virtual memory usage peaks: 18.992KB / 51.732KB.
Property is satisfied.
E<> P8_SAPBC10.pass_ball
Verification/kernel/elapsed time used: 0,016s / 0s / 0,031s.
Resident/virtual memory usage peaks: 18.992KB / 51.732KB.
Property is satisfied.
E<> P8_SAPBC9.pass_ball
Verification/kernel/elapsed time used: 0,031s / 0s / 0,031s.
Resident/virtual memory usage peaks: 18.992KB / 51.732KB.
Property is satisfied.

```

Figura 40. Resultado da verificação da propriedade das regras do jogador 8 inseridas nesta etapa.

O jogador 9 teve as seguintes regras incorporadas:

- rule_side_attack_pass_ball_10 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P9_SAPB C10;
- rule_side_attack_pass_ball_11 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P9_SAPB C11;
- rule_side_attack_drive_ball_forward ⇔ Side_Attack_Drive_Ball_Foward ⇔ P9_SADBFW.
- rule_side_attack_kick_to_goal ⇔ Side_Attack_Kick_To_Goal ⇔ P9_SAKTG.
- rule_side_attack_strategic_position_free_mark ⇔ Side_Attack_Strategic_Position_Free_Mark ⇔ P9_SASPFM;
- rule_side_attack_strategic_position ⇔ Side_Attack_Strategic_Position ⇔ P9_SASP.

A figura 41 apresenta o resultado da verificação das propriedades vinculadas às regras deste jogador e que todas estas foram satisfeitas.

Os jogadores 10 e 11 possuem o mesmo conjunto de regras e, por esse motivo, foi considerado apenas o jogador 10 com as seguintes regras:

- rule_side_attack_pass_ball_9 ⇔ Side_Attack_Pass_Ball_Cond ⇔ P10_SAPB C9;
- rule_side_attack_drive_ball_forward ⇔ Side_Attack_Drive_Ball_Foward ⇔ P10_SADBFW.
- rule_side_attack_kick_to_goal ⇔ Side_Attack_Kick_To_Goal ⇔ P10_SADB FW.
- rule_side_attack_strategic_position ⇔ Side_Attack_Strategic_Position ⇔ P9_SASP

- rule_side_attack_drive_ball_fast ⇔ Side_Attack_Drive_Ball_Fast ⇔ P10_SADBFS;

```
E<> P9_SASP.strategic_position
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 15.164KB / 44.164KB.
Property is satisfied.
E<> P9_SASPFM.free_mark
Verification/kernel/elapsed time used: 0,047s / 0s / 0,047s.
Resident/virtual memory usage peaks: 15.196KB / 44.228KB.
Property is satisfied.
E<> P9_SAKTG.kick_to_goal
Verification/kernel/elapsed time used: 0,031s / 0s / 0,031s.
Resident/virtual memory usage peaks: 15.208KB / 44.240KB.
Property is satisfied.
E<> P9_SADBFW.drive_ball_foward
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 15.216KB / 44.248KB.
Property is satisfied.
E<> P9_SAPBC11.pass_ball
Verification/kernel/elapsed time used: 0,031s / 0s / 0,031s.
Resident/virtual memory usage peaks: 15.216KB / 44.248KB.
Property is satisfied.
E<> P9_SAPBC10.pass_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 15.216KB / 44.248KB.
```

Figura 41. Resultado da verificação da propriedade das regras do jogador 9 inseridas nesta etapa.

A figura 42 apresenta o resultado das verificações das propriedades relativas às regras inseridas e que todas estas propriedades foram satisfeitas.

```
E<> P10_SADBFS.drive_ball_fast
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 27.324KB / 70.228KB.
Property is satisfied.
E<> P10_SASP.strategic_position
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 27.324KB / 70.228KB.
Property is satisfied.
E<> P10_SAKTG.kick_to_goal
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 27.324KB / 70.228KB.
Property is satisfied.
E<> P10_SADBFW.drive_ball_foward
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 27.324KB / 70.228KB.
Property is satisfied.
E<> P10_SAPBC9.pass_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 27.324KB / 70.228KB.
Property is satisfied.
E<> P10_SAIB.intercept_ball
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 27.324KB / 70.228KB.
Property is satisfied.
```

Figura 42. Resultado da verificação da propriedade das regras do jogador 10 inseridas nesta etapa.

Como resultado do processo de verificação das regras individuais de cada jogador, algumas ações foram tomadas para permitir que o processo de verificação fosse efetivado para a equipe de jogadores. Como relatado acima, algumas regras cujas propriedades não foram satisfeitas foram descartadas e outras foram alteradas para evitar a ocorrência de *livelocks*, sem comprometer o modelo. Além disso, devido à existência de regras que dependem da atuação de todos os jogadores foi necessário estender o modelo existente de forma a incorporar as condições para a verificação conjunta da equipe, a qual será tratada nos capítulos seguintes.

7. ESTENDENDO A ESPECIFICAÇÃO PARA A VERIFICAÇÃO DA EQUIPE

Para a verificação das propriedades que dependem da interação coletiva dos jogadores foi adotado um processo de sucessivos refinamentos dos modelos previamente definidos. Todos os modelos especificados e verificados até então foram adaptados e complementados, e um novo modelo foi criado para representar a interação de todos os jogadores com o ambiente (*Soccerserver*) de forma a se obter uma visão centralizada do jogo, em contraposição às visões parciais de cada jogador.

7.1. ANÁLISE DO AMBIENTE

A modelagem do *Soccerserver* foi idealizada para, dentre outras atribuições, atender às necessidades de representação da atuação e percepção dos jogadores da equipe sem, no entanto, se ater aos detalhes do nível reativo e nem aos detalhes de implementação do *Soccerserver*.

As principais necessidades de representação da atuação e percepção dos jogadores são:

- identificar a própria localização e a localização da bola;
- promover deslocamentos entre as regiões do campo;
- identificar a posse de bola.

As principais funções executadas pelo *Soccerserver* estão representadas pelo fluxograma na figura 43, e constam de:

- controlar o início, reinício e fim da partida;
- configurar o ambiente e os jogadores para o início e reinício da partida;
- aguardar e controlar o recebimento das informações de atuação dos jogadores no ambiente;
- atualizar o ambiente após o recebimento das informações de atuação dos jogadores;
- enviar as informações para cada jogador sobre a nova situação do ambiente, após a atuação da equipe e dos adversários.

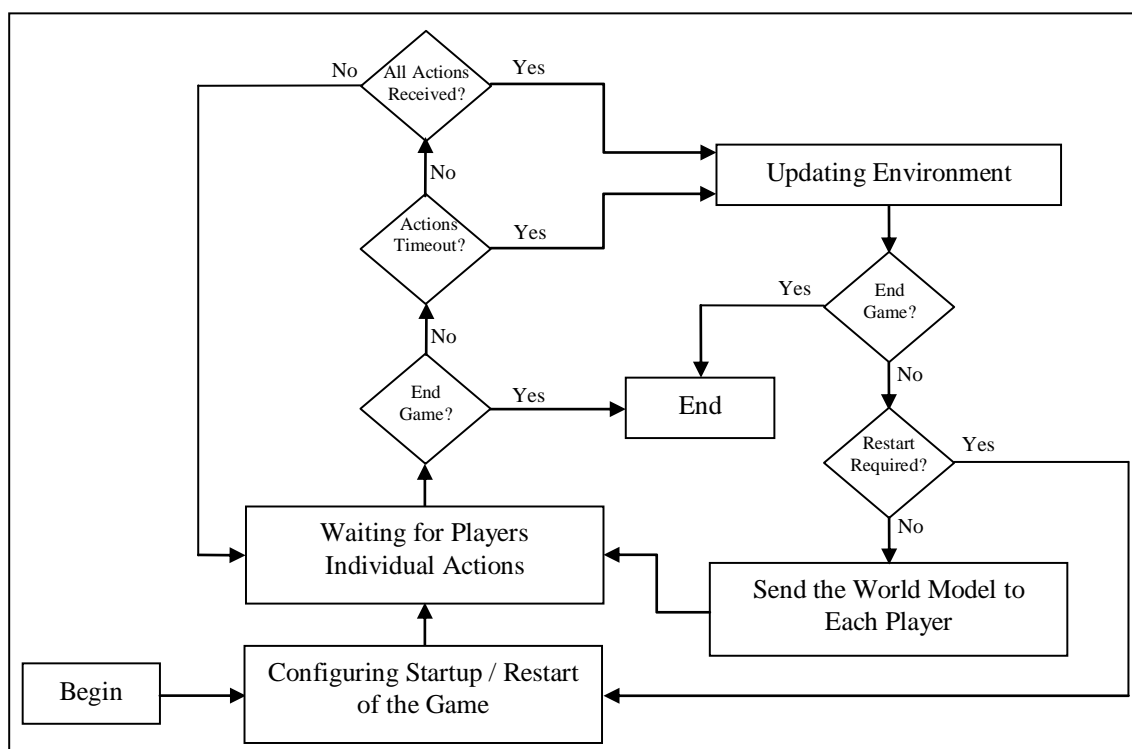


Figura 43. Representação das principais funções realizadas pelo *Soccerserver*.

Para iniciar ou reiniciar o jogo, o *Soccerserver* posiciona as equipes no campo e identifica a posse da bola de acordo com a situação da partida. No que diz respeito à disposição dos jogadores no campo, tanto no início como no reinício da partida, os jogadores de ambas as equipes devem estar posicionados nos seus respectivos lados do campo, sendo tal (re)posicionamento controlado e realizado de forma impositiva pelo *Soccerserver*.

A diferença entre o início e o reinício da partida se dá pela questão da posse de bola. No início do jogo, a posse de bola é definida por um sorteio feito pelo *Soccerserver* entre as equipes, ficando com a posse de bola aquela equipe que ganhar o sorteio. Já o reinício da partida ocorre quando uma das equipes marca gol ou quando se dá o início do segundo tempo da disputa. No primeiro caso, a posse de bola é dada à equipe que sofreu o gol. Na segunda situação, a posse de bola é dada a quem perdeu o sorteio do início do jogo.

Outra atribuição do *Soccerserver* é aguardar e controlar o recebimento das ações dos jogadores, nas quais estes apresentam informações de como pretendem atuar no ambiente. O simulador irá receber, durante toda a partida, as

informações de atuação de todos os jogadores ou daqueles que as conseguirem enviar em intervalos de 80ms.

No caso do envio de informações do jogador para o ambiente, a comunicação é síncrona, mas não garante que o servidor irá receber as mensagens de todos os jogadores, tendo em vista as restrições temporais existentes.

Depois de receber as informações dos jogadores o *Socccerserver* as confronta com a atuação da equipe adversária e, assim, atualiza o ambiente. Com isso o *Socccerserver* tem a verdadeira situação do jogo, como posicionamento dos jogadores, posição da bola, posse de bola, etc. e se comunica com todos os jogadores apresentando, para cada um, a cada 150ms, uma visão particular do jogo. Cada jogador tem uma visão parcial do ambiente, que é função da posição relativa do jogador em relação aos outros jogadores, à bola, aos marcos referenciais do campo, etc..

Para a especificação do *Socccerserver* buscou-se equacionar uma representação fiel do ambiente com o tamanho e complexidade do modelo. Para isto, algumas situações foram abstraídas, quais sejam:

- restrições temporais da comunicação dos jogadores com o *Socccerserver* e vice-versa: o intervalo de tempo para os jogadores enviarem suas ações e o intervalo de tempo para o *Socccerserver* atualizar os jogadores com as informações do jogo;
- restrições temporais da partida: tempo de duração do jogo e dos tempos da partida;
- situações de jogo: impedimentos, laterais, escanteios, faltas, etc.

O motivo de se ter abstraído as três situações apresentadas se deve ao fato de serem transparentes aos planos, tendo em vista que nenhuma das regras existentes fazem referência a elas.

Portanto, com base nessas premissas, a representação do ambiente da figura 43 foi redefinida conforme apresentado na figura 44. Além das atribuições inerentes ao ambiente *Socccerserver* apresentadas anteriormente, a atuação da equipe adversária foi modelada também como parte do ambiente, através da adição da funcionalidade "*Opposing Team's Perception and Action*", a qual executa as tarefas de percepção e atuação de uma equipe adversária.

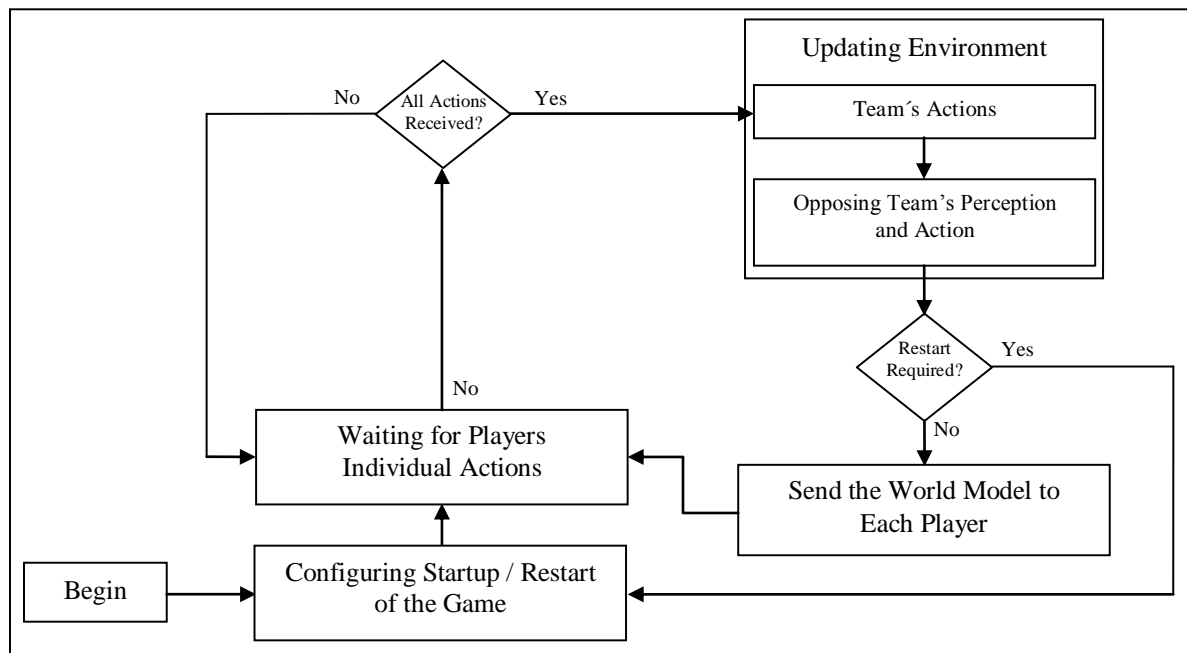


Figura 44. Representação do ambiente após sucessivos refinamentos.

7.2. MODELAGEM DO AMBIENTE

O autômato que representa o modelo do ambiente *Soccerserver* é apresentado na figura 45 e foi definido a partir da especificação apresentada na figura 44.

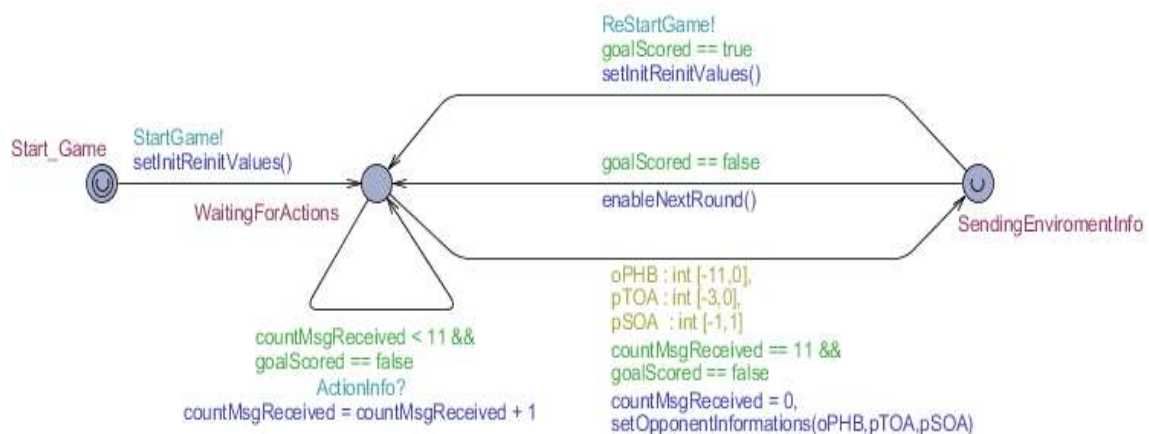


Figura 45. Autômato do ambiente *Soccerserver*.

O autômato é composto de 03(três) nós: *Start_Game*, *WaitingForActions* e *SendingEnvironmentInfo*. O nó *Start_Game* representa o início do sistema. Ele se sincroniza com os autômatos que representam as regras cognitivas de cada jogador, para que os mesmos iniciem a partida.

O nó *WaitingForActions* é o estado de início / reinício de jogo. Este estado representa a situação em que uma nova visão do ambiente é disponibilizada para cada jogador. Além disso, o sistema continuará no referido estado, enquanto não forem enviadas as informações de todos os jogadores no canal *ActionInfo* ou não for identificado que nenhuma das equipe marcou gol ($(goalScored \neq 0) == false$).

A quantidade de mensagens encaminhadas ao ambiente é controlada pela variável contadora de mensagens recebidas (*countMsgReceived*), a qual é incrementada de uma unidade a cada nova sincronização recebida. Assim, quando a quantidade de mensagens recebidas for igual à quantidade de jogadores da equipe, ocorrerá a transição para *SendingEnvironmentInfo*. Esta transição encerra a funcionalidade denominada *setOpponentInformations()*, que percebe o ambiente e o atualiza sob a ótica da equipe adversária, além de sincronizar o ambiente com todos os jogadores, via vetor *EnvironmentInfo[]*.

O nó *SendingEnvironmentInfo* fará a transição para o nó *WaitingForActions* a cada nova interação ou quando houver necessidade de reinício da partida pela ocorrência de gols.

Ainda em se tratando da especificação do ambiente, foi necessário criar as funções *setInitReinitValues()* e *enableNextRound()*. A função *setInitReinitValues()* realiza a configuração ou reconfiguração do ambiente, respectivamente, no início da partida ou quando da detecção de gol realizado por uma das equipes. A chamada da função ocorre na transição do nó *Start_Game* para o nó *WaitingForActions*, no início da partida ou na transição do nó *SendingEnvironmentInfo* para o nó *WaitingForActions*, para então permitir o início ou reinício da partida e identificar e definir a posse e posicionamento da bola e de todos os jogadores nas suas posições iniciais.

Conjuntamente com a modelagem do ambiente foi necessário implementar o registro de informações globais do sistema e mecanismos de comunicação entre os jogadores e o ambiente, e *vice-versa*.

Como cada jogador possui uma visão parcial do ambiente, eles podem ter informações diferentes e nenhuma delas corresponder à verdadeira situação do jogo. Portanto, o ambiente deve registrar o verdadeiro cenário da partida em cada novo ciclo do jogo para que, se necessário, confrontar a visão de cada jogador com este cenário real.

Com base nas definições do sistema foram criadas variáveis e funções de controle do ambiente, canais de comunicação e uma função para representação do comportamento da equipe adversária. A implementação de uma função para representar a atuação de uma equipe adversária, ao invés de modelar e instanciar os planos de cada jogador desta, visa reduzir a complexidade do modelo e respectivo espaço de estados.

7.2.1. Variáveis e funções de controle do ambiente

As variáveis globais *SoccerServerBallPosition*, *SoccerServerPlayer HoldingTheBall*, *goalScored*, o vetor *scoreBoard[]* e o vetor *EnvironmentInfo[]* e os canais de sincronização *StartGame*, *ReStartGame* e *ActionInfo* representam, respectivamente, informações gerais do sistema e a comunicação do ambiente com os jogadores e vice-versa, conforme quadro 17.

```
// SoccerServer's Variables and channels
int SoccerServerBallPosition = initialBallPosition;
int SoccerServerPlayerHoldingTheBall = initialPlayerHoldingTheBall;
bool goalScored = false;
int scoreboard[2] = {0,0};
bool EnvironmentInfo[12] = {false,false,false,false,false,false,false,false,false,false,false,false};
/* {0} = Not used, {1} = Keeper, {2}{3}{4}{5} = Defenders,
   {6}{7}{8}{9}{10}{11} = Mid - attack players
   values: true = environment info passed to player || false = environment info not passed to player */

urgent broadcast chan StartGame;
urgent broadcast chan ReStartGame;
urgent chan ActionInfo;
```

Quadro 17. Declaração das variáveis globais e canais do ambiente.

A variável *SoccerServerBallPosition* armazena um valor inteiro no intervalo de [0,5] para representar as posições da bola dentro do campo (vide seção 5.2.2.1). A variável *SoccerServerPlayerHoldingTheBall* armazena o valor do número do jogador que realmente está controlando a bola. Esta variável deve apresentar um dos seguinte valores:

- [1,11] para representar quando um jogador da equipe verificada está com o controle da bola;
- [-11,-1] para representar quando um jogador da equipe adversária está com a posse da bola;
- {0} para representar quando nenhum jogador está com a posse da bola.

A variável *goalScored* e o vetor *scoreBoard[]* são utilizados respectivamente para assinalar a ocorrência de gol por parte de uma das equipes e registrar o placar da partida. A variável *goalScored* poderá assumir os valores: 0 (quando não houver gol registrado ou após a atualização do placar), -1 (quando a equipe oponente tiver realizado um gol) e 1 (quando a equipe verificada tiver realizado um gol). No vetor *scoreBoard[]* são registrados os gols efetuados pelas equipes, considerando que se a equipe que está sendo verificada tiver feito o gol, então o valor da posição 0 do vetor (*scoreBoard[0]++*) será incrementado em uma unidade. Se a equipe adversária fizer um gol, então o valor da posição 1 do vetor (*scoreBoard[1]--*) será decrementado em uma unidade.

O vetor *EnvironmentInfo[]* é do tipo booleano e serve como um semáforo para cada jogador de forma a garantir que este, após informar ao *Soccerserver* que houve o acionamento de um comportamento reativo, aguarde até o simulador apresente a nova situação do ambiente para todos os jogadores.

Assim, quando o jogador informa ao ambiente o acionamento de um comportamento reativo, ele altera o valor da sua posição no vetor *EnvironmentInfo[]* para true (*EnvironmentInfo[] = true*) e fica bloqueado até que o *Soccerserver* receba o acionamento de todos os jogadores e os processe.

Depois que o *Soccerserver* processa tudo para aquela rodada do jogo, ele desbloqueia as regras dos jogadores alterando o valor de todas as posições do vetor para *false*.

7.2.2. Canais de sincronização

Os canais de sincronização *StartGame* e *ReStartGame* são do tipo “*urgent broadcast chan*” e foram concebidos para permitir que o *Soccerserver* informe a todos os jogadores, ao mesmo tempo, que o jogo foi iniciado ou reiniciado. O canal *StartGame* é sincronizado nos autômatos dos jogadores, referentes aos seus planos cognitivos, na transição do nó inicial para o nó sucessor. O canal *ReStartGame* é responsável pela inicialização dos autômatos referentes às regras cognitivas e instintivas quando ocorre um gol por uma das equipes.

O canal *ActionInfo* é responsável pela sincronização dos jogadores com o *Soccerserver*, informando a existência de ações a serem executadas. Desta maneira, como cada jogador se comunica individualmente com o *Soccerserver*,

numa relação 1 para 1, o canal *ActionInfo* foi declarado como “*urgent chan*”, para garantir a prioridade no envio da mensagem em relação a outras transições possíveis e pelo fato deste canal ser confiável no sistema real.

7.2.3. Representação do comportamento da equipe adversária

A função *setOpponentInformations()* (vide Apêndice C) tem por tarefa verificar a situação do ambiente a cada instante, primeiramente identificando se o jogador que possui a bola é da equipe oponente ou não através da variável de ambiente *SoccerserverPlayerHoldingTheBall*, conforme visto na seção 7.1.1. Esta identificação definirá a adoção de um entre dois tipos de comportamentos:

- a) Comportamento ofensivo - Caso seja identificado que a equipe adversária está com a posse de bola ($-11 \leq \text{SoccerserverPlayerHoldingTheBall} < 0$), a função promoverá a adoção de uma postura ofensiva caracterizada pelo avanço em direção ao lado do campo aonde encontra-se o gol da equipe que está sendo verificada.
- b) Comportamento defensivo - Caso não possua a posse de bola ($0 \leq \text{SoccerserverPlayerHoldingTheBall} \leq 11$), a equipe adversária adotará uma postura defensiva de marcação para recuperar a posse de bola.

A abordagem adotada para representar o comportamento ofensivo da equipe oponente não representa uma estratégia ou plano elaborado, muito pelo contrário. O seu objetivo é permitir submeter as regras da equipe que está sendo verificada a situações de jogo onde a posição e posse de bola alternem-se e que a situação da equipe possa ser cambiada entre as regras ofensivas e defensivas.

Para poder atuar ofensivamente, a equipe adversária deve identificar em que posição (região) do campo ela está com a posse da bola, através da variável *SoccerserverBallPosition*, vista na seção 7.1.1, pois terá ações específicas a depender da região do campo aonde esteja: regiões 0 ou 1 ($\text{SoccerserverBallPosition} \leq 1$), regiões 2 ou 3 ou 4 ($\text{SoccerserverBallPosition} < 5$) ou na região 5 ($\text{SoccerserverBallPosition} == 5$).

Caso a bola esteja nas regiões 0 ou 1 o jogador adversário poderá tentar chutar a gol. Já nos demais casos, a equipe adversária tentará avançar conduzindo ou trocando passes entre os jogadores até as regiões 0 e 1 para, posteriormente,

tentar chutar a gol. Em todos os casos foi considerado o não determinismo do *Soccerserver*, e por este motivo foi utilizado o verbo “tentar“, considerando que, neste tipo de ambiente, a execução de uma ação pode ser bem sucedida ou não. Para gerar resultados não deterministas, foram usadas as seguintes variáveis do tipo *selection* do UPPAAL, cujas descrições e valores e descrições são apresentados na tabela 20: oPHB (*opponent Player Holding Ball*), pTOA (*player Type Of Action*) e pSOA (*player Succeed Of Action*).

Tabela 20. Resumo dos valores definidos para as variáveis.

Variáveis	Valores	Descrição
oPHB	{0}	- Nenhum jogador terá a posse da bola.
	[-11,-1]	- O jogador passará a bola para o jogador -1 ou -2 ou ou -11.
pTOA	{0}	- Se oPHB diferente de zero, o jogador ficará parado com a bola; - Se oPHB igual a zero, o jogador perderá a bola e esta ficará a na mesma região do jogador.
	{-1}	- O jogador executará um toque curto para um outro jogador definido por oPHB a uma posição de distância; - No caso de oPHB igual a zero, o jogador perderá a posse da bola que ficará a uma posição de distância.
	{-2}	-O jogador fará um passe de média distância para um jogador definido por oPHB a duas posições de distância; - No caso de oPHB igual a zero, o jogador perderá a posse da bola que ficará a duas posições de distância.
	{-3}	- O jogador fará um passe longo para um outro jogador definido por oPHB a uma distância de três posições em relação à sua. - No caso de oPHB igual a zero, o jogador perderá a posse da bola que ficará a três posições de distância.
pSOA	[-1,0]	- A ação definida em pTOA não é bem sucedida.
	{1}	- A ação definida em pTOA é bem sucedida.

Como exemplos da atuação da equipe adversária são apresentados alguns cenários possíveis, conforme as figuras 46 e 47.

A figura 46 apresenta o cenário em que a equipe oponente possui a bola (*SoccerserverPlayerHoldingTheBall == -9*) e está na região 01 do campo. Neste cenário, serão descritas algumas das possibilidades resultantes dos valores obtidos pelas variáveis *selection* oPHB, pTOA e pSOA.

A letra (a) da figura 47 representa o cenário inicial já apresentado na figura 46. A situação demonstrada na letra (b) corresponde ao caso do jogador ter tentado chutar ao gol e o alcance da bola ter excedido a posição correspondente a este. Isto acontece quando o valor da variável oPHB for menor que -6 ([-11, -6[). Como consequência desta situação, será forçada a situação de tiro de meta para a equipe que está sendo verificada. Na situação (c), o jogador chutou e não atingiu ao

gol e perdeu a posse de bola. Isto acontece quando o valor da variável oPHB for maior que -6 ($[-6, 0]$), o valor da posição atual da bola ($SoccerserverBallPosition + oPHB$) for maior ou igual a 0 (alcance curto) e a tentativa de passe não for bem sucedida. Já a situação (d) representa o gol realizado, na qual o alcance calculado para a variável oPHB, somado à posição da bola foi igual a -1 e a tentativa de chute foi bem sucedida ($sOA == 1$).

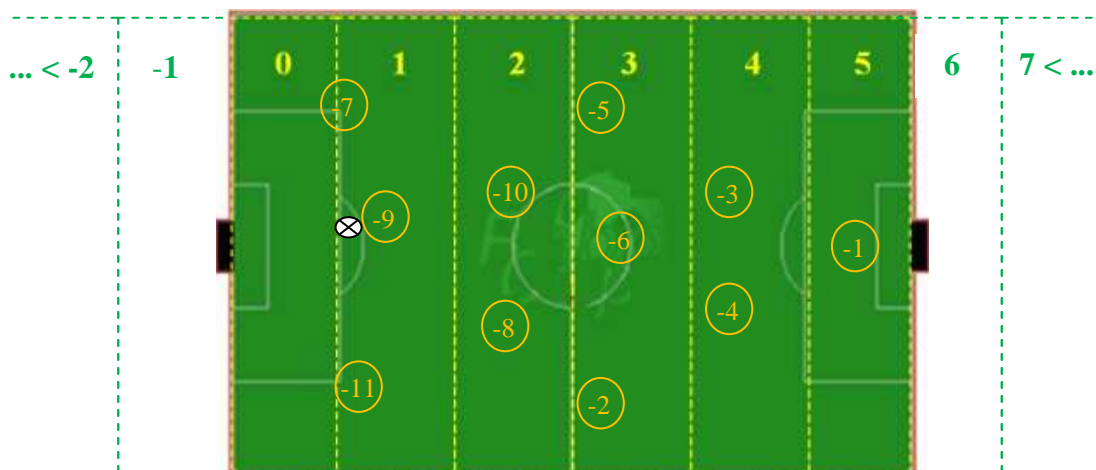


Figura 46. Cenário ofensivo da equipe oponente com $SoccerserverBallPosition \leq 1$.

Nas demais possibilidades de atuação ofensiva, nas quais a posição da bola é diferente de 0 e 1, a equipe adversária irá tentar promover a passagem da bola, entre os seus jogadores, para as posições mais ofensivas podendo obter sucesso ou não, similarmente ao que foi apresentado anteriormente. No caso particular de não haver sucesso, isto significará que a equipe perdeu a posse de bola.

No caso da bola não estar de posse da equipe oponente esta pode estar em duas situações: posse da equipe que está sendo verificada ou sem estar sob a posse de nenhum dos jogadores. Em ambos os casos a função irá identificar a posição no campo aonde a bola está para tentar retomá-la identificando, também, qual jogador está mais propenso a realizar tal tarefa.

Para realizar a marcação também foram consideradas as questões relativas ao não determinismo do ambiente. Para isto, foram utilizadas somente as variáveis pSOA e oPHB apresentadas na tabela 19 e seus respectivos intervalos de valores. Além disso, o comportamento associado a cada valor e a lógica envolvida

na execução da marcação foram modificados para atender as peculiaridades desta tarefa. A tabela 21 apresenta as variáveis, seus valores e descrição.

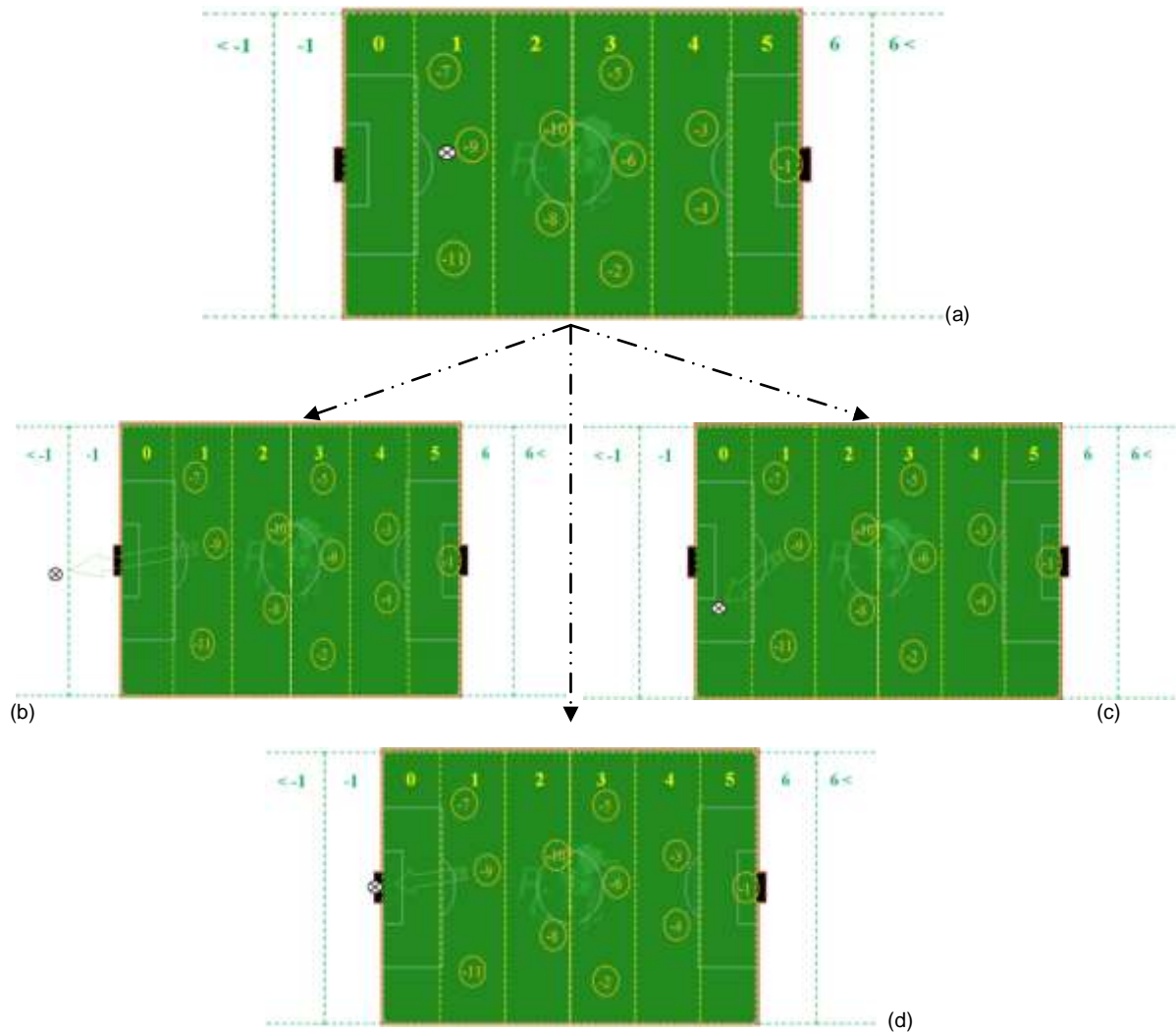


Figura 47. Representação de possíveis evoluções do (a) cenário ofensivo da equipe oponente com *SoccerserverBallPosition* ≤ 1 , (b) chute além do gol, (c) jogador mantém posse de bola e (d) gol realizado.

Tabela 21. Resumo dos valores definidos para as variáveis utilizadas na marcação.

Variáveis	Valores	Descrição
pSOA	[-1,0]	- A ação de marcação não é bem sucedida.
	{1}	- A ação de marcação é bem sucedida.
oPHB	{0}	- Nenhum jogador terá a posse da bola caso pSOA seja igual a 1; - Caso pSOA seja diferente de 1, nenhuma equipe terá a posse de bola.
	[-11,-1]	- O jogador -1 ou -2 ou ou -11 terá a posse de bola caso pSOA seja igual a 1.

No que diz respeito à estratégia de marcação promovida pela equipe oponente, é importante frisar que a mesma é realizada por uma marcação por zona, onde cada região do campo é “marcada” por um ou mais jogadores, conforme a figura 48.

A estratégia de marcação é fixa e cada região do campo sempre será “marcada” pelo(s) mesmo(s) jogador(es). Assim, se a bola estiver localizada na posição (região) 2, por exemplo, somente os jogadores -7 e -8 participaram da tentativa de retomada da posse da bola.

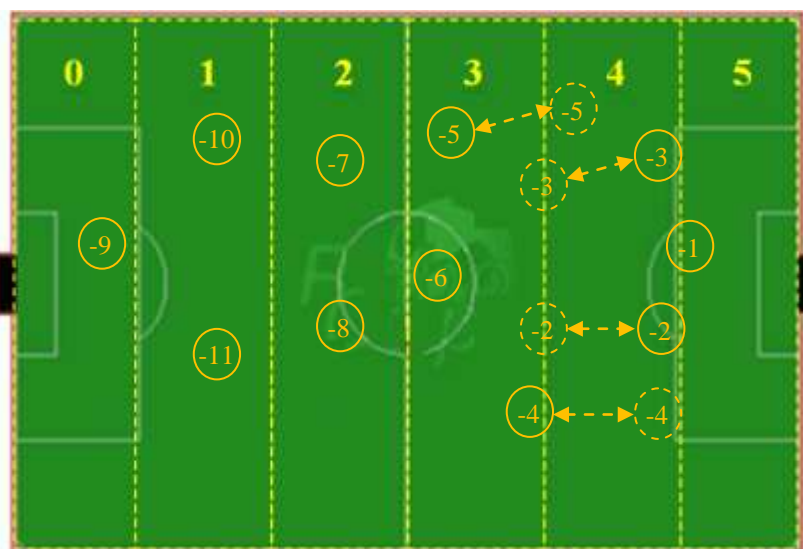


Figura 48. Representação esquemática do sistema de marcação por zona da equipe oponente.

No caso do exemplo mencionado acima, primeiro será verificado se a variável pSOA tem valor 1 e o resultado da ação de tentar retomar a bola é positivo. Se isto ocorrer então será verificado o valor da variável oPHB para identificar qual dos jogadores -7 ou -8 irá obter a posse de bola. Assim, se o valor de oPHB estiver no intervalo de $[-6,0]$ a posse de bola será do jogador -7 e se o valor estiver no intervalo $[-11, -7]$, então o jogador -8 terá a posse de bola.

Cabe destacar que a regra acima é análoga às regras de marcação das regiões 1 e 3, alterando-se somente os jogadores que concorrem pela disputa pela retomada da bola.

A marcação das regiões 4 e 5 são similares às demais apresentadas, variando-se somente na quantidade de jogadores que concorrem pela disputa da bola de 02(dois) para 04(quatro) jogadores.

Já a marcação da região 0 é caracterizada por só possuir um jogador para efetuar-la, o jogador -9. Neste caso específico, como não existem outros jogadores, só há necessidade de definir se a ação de marcação foi bem sucedida ($pSOA == 1$) ou não ($pSOA == 0$ ou $pSOA == -1$).

7.3. EVOLUÇÃO DOS MODELOS DAS REGRAS COGNITIVAS DOS JOGADORES

As regras do nível cognitivo foram reespecificadas e remodeladas com muito poucas alterações. Basicamente, as alterações promovidas nestas regras foram duas: inserir os canais de sincronização *StartGame* e *ReStartGame* para receber as informações do ambiente de início ou reinício da partida e criar uma função de inicialização do modelo do ambiente para o início da partida (*setInitialValues()*). Ambas as modificações foram implementadas na transição do nó inicial do autômato para o seu sucessor. A sincronização via *StartGame* e *ReStartGame* é utilizada para garantir que todos os jogadores comecem a “jogar” depois que o ambiente informar o início ou reinício da partida, conforme apresentado na figura 45

Por sua vez, a função *setInitialValues()* de cada jogador (vide Apêndice D) é responsável por identificar a equipe que está iniciando a partida para, depois, poder inicializar as variáveis de cada jogador. Caso a equipe com a posse de bola inicial seja a própria equipe verificada, os jogadores terão suas variáveis inicializadas com os valores apresentados na tabela 22.

De acordo com o que foi estipulado para a definição de valores das variáveis para cada jogador no início da partida, tal situação é regida pela questão da equipe estar ou não com a posse de bola neste instante. No caso da tabela 21, são apresentados de forma resumida os respectivos valores de cada variável dos jogadores, o que representa o modelo do ambiente no qual os mesmos se encontram no início da partida. Desta forma, conforme mostrado nesta tabela, no início da partida os jogadores possuem uma visão real do ambiente, o que faz com que todos possuam os mesmos valores para a maioria das variáveis, quais sejam: *playerHoldingTheBall*, *ballPosition == true*, *teammateHasBall == true*, *midplayersHasBall == true* e *playerBallLocalization == true*.

No entanto, existem variáveis como *playerBallKickable* e *playerFastesttoBall* em que todos os jogadores, exceto um, possuem valor falso

(false). Isto é devido ao fato que, no início do jogo, somente um jogador está de posse da bola e, portanto, somente ele pode chutar a bola por estar mais próximo desta.

Tabela 22. Resumo dos valores definidos por jogador para as variáveis locais quando a equipe tem a posse de bola no início a partida.

Função		Goleiro	Defesa		Meio de Campo / Ataque			
Jogadores ⇔ [pNR]		1	2 e 3	4 e 5	6 e 7	8, 9, 10 e 11		
Variáveis	playerHoldingTheBall	>= 8						
	ballPosition	true						
	teammateHasBall	true						
	midplayersHasBall	true						
	playerBallLocalization	2						
	playerBallKickable	false		Se pNR == SoccerserverPlayerHoldingTheBall true Se pNR != SoccerserverPlayerHoldingTheBall false				
	playerFastesttoBall	false		Se pNR == SoccerserverPlayerHoldingTheBall true Se pNR != SoccerserverPlayerHoldingTheBall false				
	playerLocalization	0	1	2				

Ocorrem ainda diferenças quando se consideram, por exemplo, as variáveis de localização dos jogadores, pois estes devem estar distribuídos taticamente em seu próprio lado do campo quando do início do jogo, variando suas posições de 0 a 2.

Tabela 23. Resumo dos valores por jogador para as variáveis locais quando equipe não tem a posse de bola no início a partida.

Função		Goleiro	Defesa		Meio de Campo / Ataque		
Jogadores ⇔ [pNR]		1	2 e 3	4 e 5	6 e 7	8, 9, 10 e 11	
Variáveis	playerHoldingTheBall	<= -8					
	ballPosition	true					
	teammateHasBall	false					
	midplayersHasBall	false					
	playerBallLocalization	3					
	playerBallKickable	false					
	playerFastesttoBall	false					
	playerLocalization	0	1	2			

Contudo, no caso da equipe verificada não possuir a posse de bola no momento inicial, os jogadores devem ter suas variáveis iniciadas com os valores apresentados na tabela 23.

A título de exemplo, a figura 49 apresenta o autômato de um plano cognitivo com as devidas modificações.

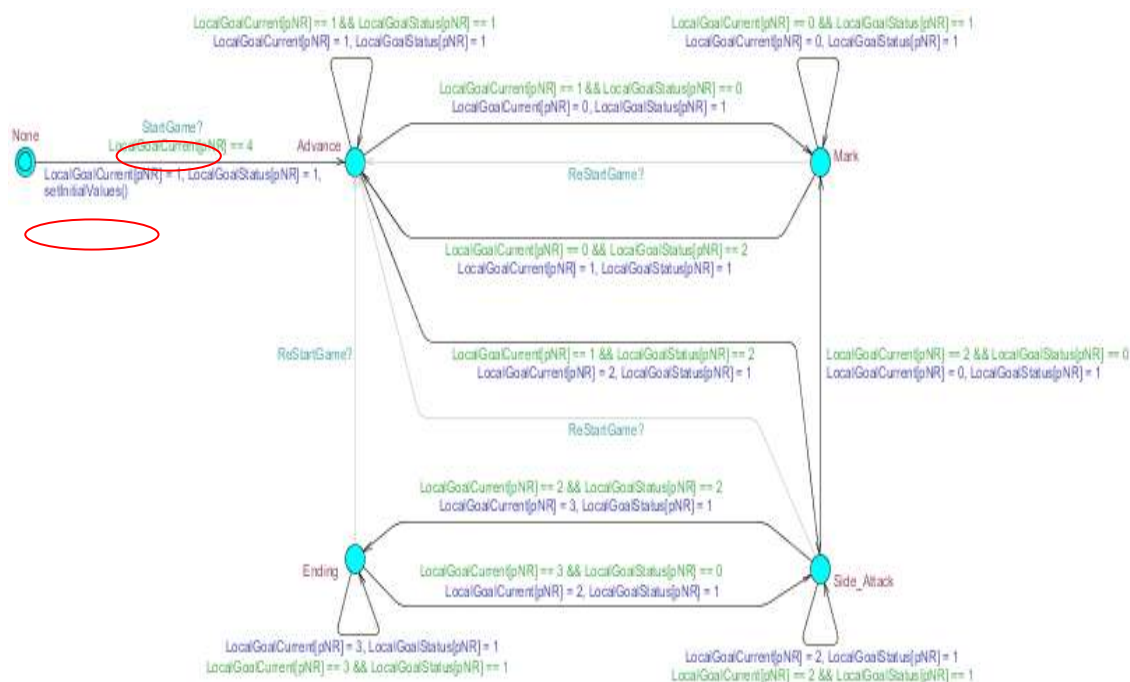


Figura 49. Modelos de regras cognitivas goleiro com destaque para o canal StartGame? e a função SetInitialValues().

7.4. EVOLUÇÃO DOS MODELOS DAS REGRAS INSTINTIVAS DOS JOGADORES

Devido às mudanças realizadas na especificação do ambiente e das regras cognitivas, as regras instintivas também foram modificadas para se adequarem aos novos modelos dos demais componentes do sistema. As modificações realizadas nas regras instintivas foram as seguintes:

- adição dos canais de comunicação *ReStartGame*, *ActionInfo* e *EnvironmentInfo*;
- adição de uma variável *tA* do tipo *selection*;
- alteração da função *setAction()*;
- alteração da função *setVariables()*.

A adição dos canais de comunicação teve como objetivo permitir a sincronização dos jogadores com o ambiente e *vice-versa*, de acordo com a explicação apresentada na especificação do ambiente do presente capítulo.

A adição da variável *tA* e da função *setAction()* está vinculada à necessidade de se definir a realização da(s) ação(ões) vinculada(s) ao comportamento reativo correspondente, acionado pela regra instintiva. No caso da variável *tA*, a mesma é utilizada para se obter um valor pseudorrandômico que irá ser utilizado pela função *setAction()* para definir a efetividade da(s) ação(ões) do comportamento reativo, ou seja, definir se o jogador efetivamente irá alcançar o objetivo vinculado ao comportamento acionado ou não.

A modificação da função *setVariables()* é motivada pela necessidade de que qualquer das regras instintivas que seja acionada permita uma percepção geral dos atributos de cada jogador e não mais uma percepção parcial dos atributos vinculados às regras, como na verificação individual de jogadores.

Todas as modificações das regras instintivas foram aplicadas nas regras que acionam comportamentos do nível reativo. No entanto, as regras do nível instintivo que promovem as mudanças de estado do nível cognitivo só tiveram adicionados os canais de comunicação. Esta diferença se deve ao fato destas últimas não terem interação com o acionamento das ações do nível reativo.

A título de exemplo, a figura 50 apresenta a regras instintivas *Mark_Hold_Ball*, antes e depois de modificada.

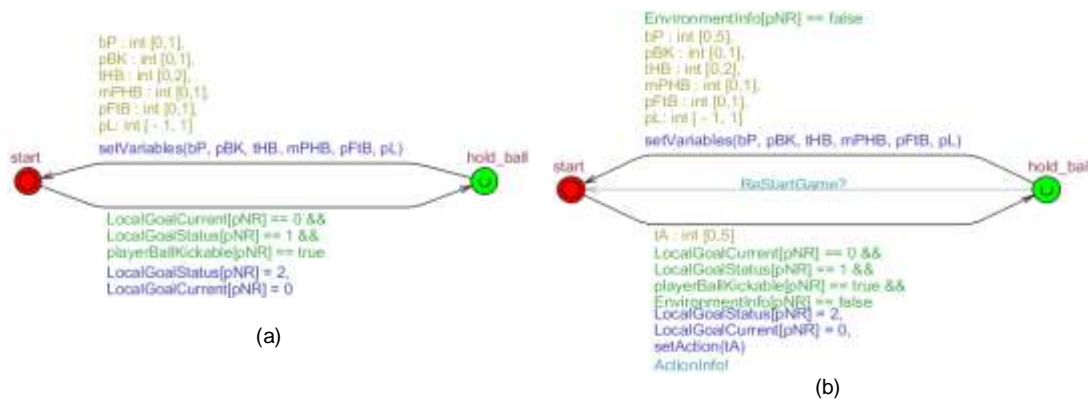


Figura 50. Comparativo entre as versões da regra *Mark_Hold_Ball* (a) utilizada na verificação das regras dos jogadores individualmente e (b) modificada para a verificação da equipe.

7.4.1. Adição da variável *tA* e Alteração das funções *setAction()*

A fim de implementar uma atuação não determinista ao comportamento acionado, foi utilizada a variável *tA*, para que a efetividade da ação ficasse condicionada a uma faixa de valores dentro do intervalo [0,5], cujo tratamento é realizado na função *setAction*.

O quadro 18 demonstra que cada grupo de jogadores possui um tratamento próprio quanto à capacidade de efetivamente retomar o controle da bola. Esta capacidade foi definida de forma a refletir o comportamento de uma equipe de futebol real, considerando que os jogadores de defesa e o goleiro (em algumas condições) possuem maior chance de serem bem sucedidos ao tentar recuperar a bola.

Considerando que a variável *tA* pode assumir valores inteiros de 0 a 5 (*tA* == 0 || 1 || 2 || 3 || 4 || 5), foi definido que os jogadores de defesa, mais o jogador 6, possuem 3 chances em 6 (50%) de conseguir efetivar a captura da bola (linhas 10 e 11 do quadro 18), caso o valor (pseudo)randomicamente atribuído à variável *tA* seja *tA* <= 2. Já na situação dos jogadores de meio de campo e ataque (jogadores 7, 8, 10 e 11) foi definido que as chances de capturar a bola são de 2 em 6 (≈33 %), conforme as linhas 12 e 13 do quadro 18. Por sua vez, foi definido que o jogador 9 (jogador de ataque conhecido como centroavante) possui a menor capacidade de captura da bola, obtendo 1 em 6 (≈17 %) chance de capturar a bola, conforme as linhas 14 e 15 do quadro 18.

```
1 void setAction(int tA) // Try to hold ball
2 {
3     if(pNR == 1) //Goalkeeper
4     {
5         if(playerBallLocalization[pNR] == 0)
6             if(tA <= 3) SoccerserverPlayerHoldingTheBall = pNR;
7         else
8             if(tA <= 1) SoccerserverPlayerHoldingTheBall = pNR;
9     }
10    if((pNR >= 2)&&(pNR <= 6)) //Defensive players + midfielder 6
11        if(tA <= 2) SoccerserverPlayerHoldingTheBall = pNR;
12    if((pNR == 7)||(pNR == 8)||(pNR == 10)||(pNR == 11)) //Mid-Attack players
13        if(tA <= 1) SoccerserverPlayerHoldingTheBall = pNR;
14    if(pNR == 9) //Attack player
15        if(tA == 0) SoccerserverPlayerHoldingTheBall = pNR;
16    playerHoldingTheBall[pNR] = SoccerserverPlayerHoldingTheBall;
17 }
```

Quadro 18. Código da função *setAction()* da regra *Mark_Hold_Ball*.

Ainda, no que diz respeito à capacidade de retomar a bola, é importante destacar a situação especial do goleiro (jogador 1). Para este jogador, as chances de capturar a bola são maiores enquanto o mesmo estiver na região da sua área. Se isto ocorrer o jogador terá 5 chances em 6 (83%) de capturar a bola (linhas 10 e 11 do quadro 18). Isto se justifica, observando o futebol humano, pelas características inerentes ao jogador de poder utilizar as mãos e os pés na sua área.

Por outro lado, se o goleiro está fora da região de sua área, este fica mais vulnerável, o que implica em aplicar, nesta situação, um valor de sucesso na captura da bola de 2 em 6 ($\approx 33\%$).

Esta definição da capacidade de retomar a posse de bola foi definida considerando o comportamento de equipes de futebol humano. Isto se deve ao fato de ser notório o entendimento de que os jogadores de defesa e o goleiro (em certas condições) possuem maior capacidade para executar esta tarefa com sucesso. Os percentuais adotados foram definidos para cada grupo de jogadores de forma intuitiva, mas podem ser perfeitamente alterados quando necessário.

As modificações apresentadas para a regra *Mark_Hold_Ball* foram aplicadas para as demais regras mantendo os novos modelos isomorfos aos seus originais, conforme pode ser identificado comparando os autômatos apresentados no Apêndice B com os do Apêndice C.

Assim, analisando as demais regras instintivas, estas podem ser agrupadas pelos comportamentos reativos acionados pelas mesmas, independente do estado ao qual estejam associadas, quais sejam:

- 1) Regras que acionam o comportamento *Search_Ball*: *Mark_Search_Ball*, *Advance_Search_Ball*, *Side_Attack_Search_Ball* e *Ending_Search_Ball*;
- 2) Regras que acionam o comportamento *Intercept_Ball*: *Mark_Intercept_Ball*, *Advance_Intercept_Ball*, *Side_Attack_Intercept_Ball* e *Ending_Intercept_Ball*;
- 3) Regras que acionam o comportamento *Strategic_Position*: *Mark_Strategic_Position*, *Advance_Strategic_Position* e *Side_Attack_Strategic_Position*;
- 4) Regras que acionam o comportamento *Mark_Opponent*: *Mark_Mark_Opponent* e *Side_Attack_Mark_Opponent*;
- 5) Regras que acionam o comportamento *Pass_Ball* (incondicional): *Advance_Pass_Ball_Uncond* e *Side_Attack_Pass_Ball_Uncond*;
- 6) Regras que acionam o comportamento *Pass_Ball* (condicional): *Advance_Pass_Ball_Cond* e *Side_Attack_Pass_Ball_Cond*;
- 7) Regras que acionam o comportamento *Drive_Ball_Forward*: *Advance_Drive_Ball_Forward* e *Side_Attack_Drive_Ball_Forward*;
- 8) Regras que acionam comportamentos específicos não abrangidos nos demais grupos e não agrupáveis entre si: *Advance_Pass_Ball_Forward*,

Advance_Drive_Ball_Fast, Side_ Attack_Drive_Ball_Fast,
Side_Attack_Kick_to_Goal, Side_ Attack_Strategic_Position_Free_Mark e
Ending_Kick_Ball.

Excetuando o último grupo de regras supramencionado, cada grupo de regras aciona um mesmo comportamento reativo e, por esse motivo, cada agrupamento teve a função *setAction()* implementada da mesma maneira em cada uma das suas regras.

O detalhamento da implementação de cada grupo de regras está detalhado no Apêndice F.

7.4.2. Alteração da função *setVariables()*

A função *setVariables()* necessitou ser modificada, pois a percepção do ambiente depende agora da atuação de todos os jogadores e da equipe adversária (vide Apêndice E).

Além disso, com a dinâmica de movimentação dos jogadores pelas regiões do campo, já descrita anteriormente, foi implementada uma capacidade de percepção do ambiente de acordo com a distância do jogador em relação à bola. Assim, quanto mais distante da bola, menor a chance do jogador obter informações mais precisas sobre a bola e sobre quem está com a posse da mesma, entre outros atributos.

Seguindo esta linha de raciocínio a função *setVariables()* verifica inicialmente se o jogador em questão encontra-se na mesma posição do campo que a bola, segundo a visão do ambiente (vide linha 3 do Apêndice E). Caso seja verdade, é verificado se o jogador está de posse da bola (linha 4) o que implicará que o mesmo tem visão total do jogo.

Por outro lado, se o jogador não estiver na mesma posição que a bola, o mesmo terá menos chances de obter informações precisas do ambiente. Esta precisão da informação varia a medida que a distância entre ele e a bola aumenta, podendo ser de 1 até 5, sendo esta última a maior distância possível para a configuração de regiões do campo adotada nesta modelagem.

8. VERIFICAÇÃO DO AMBIENTE E DA EQUIPE

A verificação do ambiente e dos planos da equipe é a etapa na qual são realizadas as verificações das propriedades relativas ao comportamento do ambiente, da equipe como um todo e da interação entre estes.

Para executar esta tarefa, primeiramente foi realizada a verificação do ambiente, seguida da verificação dos planos individuais de cada jogador conjuntamente com o ambiente e, posteriormente, de múltiplos jogadores e o ambiente.

Neste capítulo, serão descritos os procedimentos adotados para a execução das verificações, bem como a descrição dos problemas encontrados e as soluções adotadas para atingir os objetivos deste estudo de caso.

8.1. VERIFICAÇÃO DO AMBIENTE

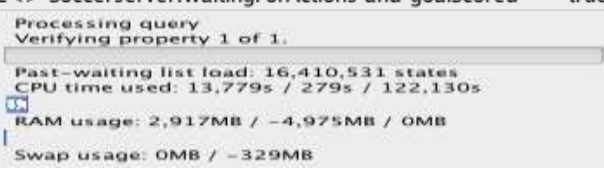
Para executar a verificação do ambiente foram utilizados os modelos das regras de cada um dos jogadores para representar efetivamente a comunicação entre o *Soccerserver* e os membros da equipe, com foco na capacidade do ambiente de receber e enviar informações para cada agente. Além disso, foram consideradas as propriedades de alcançabilidade de estados, transições e valores de variáveis definidas pelas funções inerentes ao controle do jogo e ao comportamento do ambiente.

As propriedades especificadas para verificação foram as seguintes:

- 1) Não existência de deadlocks: $A[]$ not deadlock;
- 2) Alcançabilidade dos estados *Start_Game*, *WaitingforActions* e *SendingEnvironmentInfo*: $E \langle \rangle \text{Soccer_Serve.Start_Game}$, $E \langle \rangle \text{Soccerserver.WaitingForActions}$ e $E \langle \rangle \text{Soccerserver.SendingEnvironmentInfo}$;
- 3) Propriedade de validação entre valores de variáveis e estados: $E \langle \rangle \text{Soccerserver.SendingEnvironmentInfo and goalScored == true}$; $E \langle \rangle \text{Soccerserver.SendingEnvironmentInfo and goalScored == false}$; e $E \langle \rangle \text{Soccerserver.WaitingForActions and goalScored == false}$;

A tabela 24 apresenta as propriedades e os respectivos resultados da verificação.

Tabela 24. Propriedades e resultados da verificação do ambiente.

Propriedade	Resultado
A[] not deadlock	A[] not deadlock Verification/kernel/elapsed time used: 0.217s / 0.004s / 0.22s. Resident/virtual memory usage peaks: 42,708KB / 2,521,320KB. Property is satisfied.
E<> Soccerserver.WaitingForActions	E<> Soccerserver.WaitingForActions Verification/kernel/elapsed time used: 0.213s / 0.002s / 0.285s. Resident/virtual memory usage peaks: 86,348KB / 2,579,208KB. Property is satisfied.
E<> Soccerserver.SendingEnvironmentInfo	E<> Soccerserver.SendingEnvironmentInfo Verification/kernel/elapsed time used: 0.21s / 0.003s / 0.211s. Resident/virtual memory usage peaks: 152,552KB / 2,666,296KB. Property is satisfied.
E<> Soccer_Serve.Start_Game	Não foi verificada por ser trivial, já que o referido estado é inicial.
E<> Soccerserver.SendingEnvironmentInfo and goalScored == true	E<> Soccerserver.SendingEnvironmentInfo and goalScored == true Verification/kernel/elapsed time used: 0.326s / 0.009s / 0.335s. Resident/virtual memory usage peaks: 185,896KB / 2,710,868KB. Property is satisfied.
E<> Soccerserver.SendingEnvironmentInfo and goalScored == false	E<> Soccerserver.SendingEnvironmentInfo and goalScored == false Verification/kernel/elapsed time used: 0.214s / 0.003s / 0.217s. Resident/virtual memory usage peaks: 211,380KB / 2,744,676KB. Property is satisfied.
E<> Soccerserver.WaitingForActions and goalScored == false	E<> Soccerserver.WaitingForActions and goalScored == false Verification/kernel/elapsed time used: 0.213s / 0.003s / 0.215s. Resident/virtual memory usage peaks: 229,412KB / 2,768,756KB. Property is satisfied.
E<> Soccerserver.WaitingForActions and goalScored == true	E<> Soccerserver.WaitingForActions and goalScored == true 

De todas as propriedades verificadas não foi possível obter o resultado da última apresentada na tabela, pois houve travamento do computador por falta de memória quando a verificação alcançou valores próximos a 17 milhões de estados.

O fato da grande maioria das propriedades terem sido satisfeitas permitiu dar seguimento à verificação da equipe.

8.2. VERIFICAÇÃO DA EQUIPE

A verificação da equipe foi realizada em duas subetapas: nova verificação das propriedades identificadas para cada agente individualmente juntamente com o ambiente, e a verificação de todos os jogadores conjuntamente e o ambiente.

As propriedades verificadas em ambas as subetapas foram:

- 1) Não existência de *deadlocks*;
- 2) Alcançabilidade dos estados das regras cognitivas;
- 3) Alcançabilidade dos estados da regras instintivas.

8.2.1. Nova verificação das propriedades dos jogadores com o ambiente

Esta subetapa teve por objetivo identificar se as propriedades verificadas na etapa de verificação do capítulo 6 se mantém quando da adição do ambiente ao modelo.

Como resultado foi constatado que todas as propriedades verificadas no capítulo 6 mantiveram seus resultados quando da verificação com o ambiente, exceto pela verificação da não ocorrência de *deadlock* que não pode ser apurada devido à ocorrência de explosão de estados.

As propriedades e os resultados desta subetapa são apresentados no apêndice G.

8.2.2. Verificação da equipe

A verificação da equipe foi realizada instanciando as regras de todos os jogadores e do ambiente. Algumas propriedades verificadas até então nas etapas anteriores não apresentaram resultado, devido à explosão de estados.

Como aconteceu na subetapa anterior não foi possível verificar a não ocorrência de *deadlock* para todo o sistema, devido ao travamento do computador por falta de memória.

A tabela 25 apresenta o resumo dos resultados da verificação desta etapa por jogadores e os resultados completos estão no apêndice H. É importante destacar que todas as propriedades que apresentaram resultado foram satisfeitas.

A tabela 26 apresenta as propriedades que não apresentaram resultado devido ao travamento do computador antes da conclusão da verificação.

Tabela 25. Propriedades que apresentaram resultados e foram satisfeitas na verificação da equipe com o ambiente.

E<> P1_C.Advance E<> P1_C.Mark E<> P1_MHB.hold_ball E<> P1_MSB.search_ball E<> P1_MMO.mark_opponent	E<> P1_MIB.intercept_ball E<> P1_AIB.intercept_ball E<> P1_ASP.strategic_position E<> P1_APBFW.pass_ball_forward E<> P1_AF.Set_Cognitive_Failed
E<> P2_C.Advance E<> P2_C.Mark E<> P2_MHB.hold_ball E<> P2_MSB.search_ball E<> P2_MMO.mark_opponent E<> P2_MIB.intercept_ball	E<> P2_APBFW.pass_ball_forward E<> P2_APBC8.pass_ball E<> P2_ASB.search_ball E<> P2_AIB.intercept_ball E<> P2_ASP.strategic_position E<> P2_AF.Set_Cognitive_Failed
E<> P3_C.Advance E<> P3_C.Mark E<> P3_MHB.hold_ball E<> P3_MSB.search_ball E<> P3_MMO.mark_opponent E<> P3_MIB.intercept_ball	E<> P3_APBFW.pass_ball_forward E<> P3_ASB.search_ball E<> P3_AIB.intercept_ball E<> P3_ASP.strategic_position E<> P3_AF.Set_Cognitive_Failed
E<> P4_C.Advance E<> P4_C.Mark E<> P4_MHB.hold_ball	E<> P4_MSB.search_ball E<> P4_MMO.mark_opponent E<> P4_MIB.intercept_ball
E<> P5_C.Advance E<> P5_C.Mark E<> P5_MHB.hold_ball E<> P5_MSB.search_ball E<> P5_MMO.mark_opponent E<> P5_MIB.intercept_ball	E<> P5_APBU6.pass_ball E<> P5_APBC8.pass_ball E<> P5_ASB.search_ball E<> P5_AIB.intercept_ball E<> P5_ASP.strategic_position E<> P5_AF.Set_Cognitive_Failed
E<> P6_C.Side_Attack E<> P6_C.Mark E<> P6_MHB.hold_ball E<> P6_MSB.search_ball	E<> P6_MIB.intercept_ball E<> P6_MSP.strategic_position E<> P6_SAF.Set_Cognitive_Failed
E<> P7_C.Side_Attack E<> P7_C.Mark E<> P7_MHB.hold_ball E<> P7_MSB.search_ball	E<> P7_MIB.intercept_ball E<> P7_MSP.strategic_position E<> P7_SAF.Set_Cognitive_Failed
E<> P8_C.Side_Attack E<> P8_C.Mark E<> P8_MHB.hold_ball	E<> P8_MSB.search_ball E<> P8_MIB.intercept_ball E<> P8_MSP.strategic_position
E<> P9_C.Side_Attack E<> P9_C.Mark E<> P9_MHB.hold_ball E<> P9_MSB.search_ball E<> P9_MIB.intercept_ball E<> P9_MSP.strategic_position E<> P9_SASB.search_ball	E<> P9_SAIB.intercept_ball E<> P9_SAPBC10.pass_ball E<> P9_SAPBC11.pass_ball E<> P9_SADBFW.drive_ball_forward E<> P9_SASPFM.free_mark E<> P9_SASP.strategic_position E<> P9_SAF.Set_Cognitive_Failed
E<> P10_C.Side_Attack E<> P10_C.Mark E<> P10_MHB.hold_ball E<> P10_MSB.search_ball E<> P10_MIB.intercept_ball E<> P10_MSP.strategic_position	E<> P10_SASB.search_ball E<> P10_SAIB.intercept_ball E<> P10_SAPBC9.pass_ball E<> P10_SADBFW.drive_ball_forward E<> P10_SASP.strategic_position E<> P10_SAF.Set_Cognitive_Failed
E<> P11_C.Side_Attack E<> P11_C.Mark E<> P11_MHB.hold_ball E<> P11_MSB.search_ball E<> P11_MIB.intercept_ball E<> P11_MSP.strategic_position	E<> P11_SASB.search_ball E<> P11_SAIB.intercept_ball E<> P11_SAPBC9.pass_ball E<> P11_SADBFW.drive_ball_forward E<> P11_SASP.strategic_position E<> P11_SAF.Set_Cognitive_Failed
E<> Soccerserver.SendingEnvironmentInfo E<> Soccerserver.WaitingForActions E<> Soccerserver.SendingEnvironmentInfo and goalScored == true	E<> Soccerserver.SendingEnvironmentInfo and goalScored == false E<> Soccerserver.WaitingForActions and goalScored == false

Tabela 26. Propriedades que não apresentaram resultados na verificação da equipe.

E<> P1_C.Side_Attack E<> P1_MA.Set_Cognitive_Achieved E<> P1_AA.Set_Cognitive_Achieved E<> P1_SASB.search_ball E<> P1_SAIB.intercept_ball	E<> P1_SAMO.mark_opponent E<> P1_SAPBU7.pass_ball E<> P1_SAF.Set_Cognitive_Failed E<> P1_SAA.Set_Cognitive_Achieved
E<> P2_MA.Set_Cognitive_Achieved	E<> P3_MA.Set_Cognitive_Achieved
E<> P4_C.Side_Attack E<> P4_MA.Set_Cognitive_Achieved E<> P4_APBFW.pass_ball_forward E<> P4_ASB.search_ball E<> P4_AIB.intercept_ball E<> P4_ASP.strategic_position E<> P4_AF.Set_Cognitive_Failed	E<> P4_AA.Set_Cognitive_Achieved E<> P4_SASB.search_ball E<> P4_SAIB.intercept_ball E<> P4_SAMO.mark_opponent E<> P4_SAPBU7.pass_ball E<> P4_SAF.Set_Cognitive_Failed E<> P4_SAA.Set_Cognitive_Achieved
E<> P5_C.Side_Attack	E<> P5_MA.Set_Cognitive_Achieved
E<> P6_MA.Set_Cognitive_Achieved E<> P6_ADBFW.drive_ball_forward E<> P6_SASB.search_ball E<> P6_SAIB.intercept_ball	E<> P6_SAPBC8.pass_ball E<> P6_SAPBC9.pass_ball E<> P6_SAPBC11.pass_ball E<> P6_SASPFM.free_mark
E<> P7_MA.Set_Cognitive_Achieved E<> P7_ADBFS.drive_ball_fast E<> P7_SASB.search_ball E<> P7_SAIB.intercept_ball	E<> P7_SAPBC8.pass_ball E<> P7_SAPBC9.pass_ball E<> P7_SAPBC10.pass_ball E<> P7_SASPFM.free_mark
E<> P8_MA.Set_Cognitive_Achieved E<> P8_SASB.search_ball E<> P8_SAIB.intercept_ball E<> P8_SAPBC9.pass_ball E<> P8_SAPBC10.pass_ball	E<> P8_SAPBC11.pass_ball E<> P8_SADBFW.drive_ball_forward E<> P8_SASPFM.free_mark E<> P8_SAF.Set_Cognitive_Failed
E<> P9_MA.Set_Cognitive_Achieved	E<> P9_SAKTG.kick_to_goal
E<> P10_MA.Set_Cognitive_Achieved E<> P10_SAKTG.kick_to_goal	E<> P10_SADBFS.drive_ball_fast
E<> P11_MA.Set_Cognitive_Achieved E<> P11_SAKTG.kick_to_goal	E<> P11_SADBFS.drive_ball_fast
A[] not deadlock	E<> Soccerserver.WaitingForActions and goalScored == true

A ocorrência de tal problema pode estar associada à explosão de estados, pois o computador utilizado começou a apresentar travamento ao atingir quantidades próximas a 20 milhões de estados. Esta suspeita se reforça pelo fato de todas as regras relativas aos jogadores terem anteriormente sido verificadas individualmente, sem ou na presença do modelo do *Soccerserver* e, nestas situações, terem apresentado resultado.

8.2.3. Considerações sobre a verificação

A verificação da equipe é a última etapa do processo de especificação e verificação do modelo do sistema. Durante este processo foram verificados os 03 (três) tipos de regras cognitivas e os 37 (trinta e sete) tipos de regras instintivas dos planos utilizados pelos jogadores.

As regras modeladas geraram na etapa de verificação de toda a equipe a instanciação de 11 (onze) autômatos para as regras cognitivas e 149 (cento e quarenta e nove) autômatos para as regras instintivas, somando-se mais um autômato do *Soccerserver*, num total de 161 (cento e sessenta e um) autômatos.

Em termos de quantidade de verificações realizadas tem-se um total de 715 (setecentos e quinze), distribuídas da seguinte maneira:

- 73 (setenta e três) na etapa 2;
- 191 (cento e noventa e um) na etapa 3;
- 7 (sete) na etapa 4;
- 444 (quatrocentas e quarenta e quatro) na etapa 5.

O total de propriedades verificadas foi 204 (duzentas e quatro), somando-se as propriedades correspondentes às regras de todos os jogadores e as propriedades correspondentes ao ambiente. Entretanto, enquanto nas etapas 2 e 3, todas as propriedades verificadas tiveram os resultados apresentados pelo verificador de modelos, nas etapas 4 e 5, 01 (uma) e 62 (sessenta e duas) propriedades não puderam ser verificadas devido, provavelmente, à explosão de estados.

9. CONCLUSÃO

Neste trabalho foi realizado um estudo de caso sobre a verificação de planos de agentes autônomos e sistemas multiagentes, o futebol de robôs Mecateam (UFBA). Esta equipe possui uma arquitetura híbrida em multicamadas, com ambiente não determinista e de visão parcial. Foi realizada uma análise detalhada dos planos dos agentes para modelar e verificar estes planos utilizando *model checking* com o verificador de modelos UPPAAL.

Embora este trabalho tenha considerado o problema de verificação de planos para o futebol de robôs, o estudo feito nesta dissertação poderá ser aplicado em outros domínios que envolvam agentes autônomos e sistemas multiagentes físicos ou em software, tanto para ambientes não deterministas e com visão parcial, como outros menos complexos.

As principais contribuições deste trabalho foram:

- 1) a criação de um método de verificação incremental e evolutivo para orientar o processo de especificação e verificação de planos de AA's e SMA's;
- 2) o emprego de abstrações e técnicas baseadas na verificação composicional de modelos para contornar situações de explosão de estado;
- 3) a modelagem e verificação dos planos de uma equipe de futebol de robôs simulado utilizando autômatos.

9.1. CONSIDERAÇÕES SOBRE AS SOLUÇÕES ADOTADAS

Devido às características do futebol de robôs, como o não determinismo e a visão parcial do ambiente, algumas soluções foram propostas para a modelagem tais como: a utilização de variáveis pseudo randômicas e a representação do posicionamento dos jogadores em regiões do campo de futebol, para simular o não determinismo das ações e percepções sobre o ambiente, permitindo assim a implementação da visão parcial do jogo.

Devido ao tamanho significativo do conjunto de regras dos planos e levando em consideração a estrutura em camadas dos agentes, foi necessário considerar algumas soluções para contornar o problema de explosão de estados tais como: abstrações e verificação composicional de modelos. Em relação às abstrações adotadas, destacam-se: a não representação dos comportamentos da

camada reativa, a representação unidimensional do campo de futebol, a não representação das restrições temporais do ambiente e a não representação explícita de uma equipe adversária.

As técnicas de verificação composicional de modelos utilizadas (*assume-guarantee*, *lazy parallel composition / partitioned transition relations* e *interface processes*) permitiram, entre outros resultados, que todos os estados das regras cognitivas durante a verificação fossem alcançados e também a verificação gradativa de partes do sistema, sem a necessidade de se especificar inicialmente todas as suas transições.

Para guiar o processo de modelagem e verificação foi criado um método cuja idéia geral foi adotar um processo incremental e evolutivo de verificação, integrado ao emprego das técnicas de abstração e verificação composicional de modelos. De acordo com este método, cada agente foi modelado e verificado individualmente, seguido do ambiente e, depois, todos os agentes da equipe com o ambiente, partindo previamente de uma análise do domínio. Isto permitiu que o sistema fosse especificado e verificado de forma escalonada, partindo dos componentes mais elementares até o sistema como um todo.

Em particular, em se tratando da modelagem do ambiente (a qual não era o foco do presente trabalho), é importante assinalar que o presente modelo foi fruto de uma série de soluções para equacionar a representação fiel do comportamento do sistema e o tamanho do modelo, este último fator preponderante para a ocorrência de problemas de explosões de estados durante a verificação.

Como resultado do experimento foi possível verificar uma considerável quantidade de propriedades para a quase totalidade dos planos dos jogadores individualmente e boa parte destes planos com toda a equipe e o ambiente representados no modelo. Apesar de terem sido realizadas as verificações de uma grande quantidade de propriedades, foram detectadas no processo de verificação algumas inconsistências no conjunto das regras da equipe, tanto nas regras cognitivas quanto nas instintivas. Após análise meticulosa, essas regras foram corrigidas ou excluídas do rol de regras original, a fim de permitir continuar o estudo de caso. Assim, foi possível especificar todas as 160 (cento e sessenta) regras e 01 (um) autômato do ambiente e verificar 204 (duzentas e quatro) propriedades definidas para estas regras. Dentre as propriedades 63 (sessenta e três) delas não puderam ser verificadas devido, provavelmente, à explosão de estados.

9.2. TRABALHOS FUTUROS

Em relação a trabalhos futuros, um ambiente para apoio a modelagem e verificação de modelos deste tipo de sistema poderá ser desenvolvido com base nesta experiência. Este ambiente teria etapas automáticas, como a tradução de regras para autômatos e vice-versa e semi-automáticas guiadas pelo usuário na escolha, por exemplo, de se verificar um único agente, sem ou com a representação do ambiente ou de grupos ou subgrupos de agentes do SMA.

Devido aos problemas que impossibilitaram a verificação de algumas propriedades na etapa final, poderia ser repetido o trabalho utilizando equipamentos mais robustos, aliados ou não a uma reavaliação e possível simplificação ou redução do modelo atual para se buscar uma verificação de toda a equipe.

Outra possibilidade de trabalhos futuros seria a melhoria e evolução do método para definir uma metodologia de verificação de AA's e SMA's. Essa tarefa se daria pela melhor definição dos processos e técnicas de modelagem e verificação utilizadas, integradas com a verificação da camada reativa dos AA's e SMA's, considerando a possibilidade de emprego de outras técnicas e ferramentas de verificação de modelos de acordo com a camada ou tipos de agentes, sejam eles físicos ou em *software*.

Seria possível ainda a utilização da modelagem adotada neste projeto para a aplicação de verificação probabilística de modelos. Essa possibilidade permitiria ir além da verificação da consistência das regras entre si, podendo-se verificar e mensurar a efetividade de cada regra ou conjunto de regras.

REFERÊNCIAS

- [1] RUSSEL, Stuart; NORVIG, Peter. **Inteligência Artificial**. 2ª edição. Rio de Janeiro : Elsevier, 2004. 1040 p.
- [2] CLARKE, E. M.; WING J. M. **Formal Methods: State of the Art and Future Directions**. ACM Computing Surveys, Vol. 28, No. 4, dezembro 1996.
- [3] HALL, A. **Seven Myths of Formal Methods**. IEEE Software, setembro de 1990, pp. 11-20.
- [4] PISTORE, M.; TRAVERSO, P. **Planning as Model Checking for Extended Goals in Non-deterministic Domains**. International Joint Conference on Artificial Intelligence. Vol 17. Ed 1. Seattle: Lawrence Erlbaum Ass. Ltd., 2001. p479-486.
- [5] CIMATTI, A.; ROVERI, M. **Conformant Planning via Model Checking**. Journal of Artificial Intelligence Research, 13, pp. 305-338.
- [6] BÉRARD, B. et al. **Systems and Software Verification: Model-Checking Techniques and Tools**. Berlim : Springer - Verlag, 2001. 202 p.
- [7] BERTOLI, P. et al. **Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking**, In Reasoning, Action and Interaction in AI Theories and Systems Lecture Notes in Computer Science Volume 4155, 2006, pp 213-228.
- [8] EL FALLAH-SEGHROUCHNI, A. et al. **Modelling, Control and Validation of Multi-Agent Plans in Dynamic Context**, AAMAS, vol. 1, pp.44-51, Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS'04), 2004.
- [9] MARC, F. **Planification Multi-Agent sous Contraintes dans un Contexte Dynamique: Application Aux Simulations Aériennes**. Thèse (Doctorat en Informatique), École Doctorale d'Informatique, Télécommunication et Électronique de Paris, Université Pierre et Marie Curie, 2005.
- [10] HENZINGER, T. A. **The Theory of Hybrid Automata**. 28p. Electronics Research Laboratory, College of Engineering, University of California : Berkeley, 1996.

- [11] KHATIB, L; MUSCETTOLA, N.; HAVELUND, K. **Verification of plan models using UPPAAL**. 1st International Workshop on Formal Approches to Agent-Based Systems. Maryland, 2000.
- [12] KHATIB, L; MUSCETTOLA, N.; HAVELUND, K. **Mapping temporal planning constraints into timed automata**. In: Proceeding of 8th International Syposium on Temporal Representation and Reasoning. 249p. IEEE Computer Society : Cividale Del Friuli, 2001.
- [13] COSTA, George Souza. **Utilização da Verificação de Modelos para o Planejamento de Missões de Veículos Aéreos não-Tripulados**. Dissertação (Mestrado em Engenharia Elétrica) - Instituto Militar de Engenharia, Rio de Janeiro, 2008.
- [14] EARLE, C. B. et al. **Verifying Robocup Teams**. In: Proceeding of MoChArt 2008, 5th International Workshop on Model Checking and Artificial Intelligence.189 p. Patras, 2008.
- [15] WOOLDRIDGE, M.; JENNINGS, N.R.. **Agents Theories, Architectures, and Languages: A Survey in Intelligent Agents**. In: Proceedings of ECAI-94, Workshop on Agent Theories Architectures and Languages, Lecture Notes in Artificial Intelligence.890p. Willey : Amsterdan, 1994.
- [16] FRANKLIN, S.; GRAESSER, A. **Is it a agent or just a Program?: A Taxonomy for Autonomous Agents**. In: Proceeding of the Third International Workshop on Agent Theories, Architectures and Languages. Springer – Verlag : Berlim, 1997.p.21-35.
- [17] FERBER, J.; GASSER, L.. **Intelligence artificielle distribuée**. Tutorial Notes of the 11th Conference on Expert Systems and their Applications. Avignon, 1991.
- [18] ZINI, F. **A Multi-Agent Specification Environment for Complex Software Applications**. Thesis (Dottorado in Informática), Depatimento di Informatica e Scienze dell'Informazione, Univercità degli Studi di Genova,1998.
- [19] LOUREIRO DA COSTA, Augusto. **ExpertCoop: Um Ambiente para Desenvolvimento de Sistemas Multiagentes Cognitivos**. Dissertação (Mestrado em Engenharia Elétrica), Laboratório de Controle e Micro Informática, Universidade Federal de Santa Catarina, 1997.

- [20] MENDES, M. et al. **Agents Skills And Their Roles In Mobile Computing And Personal Communications**. IFIP, 14th World Computer Congress, World Conference on Mobile Communications. Canberra, 1996.
- [21] MAES, Patty. **Design Autonomous Agents: Theory and Practice from Biology to Engineering and Back**. MIT Press : Cambridge, 1990. 200p.
- [22] LOUREIRO DA COSTA, Augusto. **Conhecimento Social Dinâmico: Uma Estratégia de Cooperação para Sistemas Multiagentes Cognitivos**. Tese (Doutorado em Engenharia Elétrica), Laboratório de Controle e Micro Informática, Universidade Federal de Santa Catarina, 2001.
- [23] COSTA, Augusto L.; BITTENCOURT, Guilherme. **From a concurrent architecture to a concurrent autonomous agents architecture**. IJCAI'99, Third International Workshop in Robocup, pages 85-90, Lecture Notes in Artificial Intelligence. Springer : Utrecht, 2000.
- [24] JUNIOR, Orivaldo V. de Santana. **MECATEAM FRAMEWORK: Uma infra-estrutura para desenvolvimento de agentes de futebol de robôs simulado**. Monografia (Graduação em Ciência da Computação) - Universidade Federal da Bahia, Salvador, 2007.
- [25] BITTENCOURT, Guilherme. **In the quest of the missing link**. In: Proceedings of IJCAI. p.310-315. Nagoya, 1997.
- [26] BITTENCOURT, Guilherme. **Inteligência Artificial: Ferramentas e Teorias**. 2^a ed. Ed. da UFSC : Florianópolis, 2001. 362p.
- [27] REZENDE, Solange O.. **Sistemas Inteligentes: Fundamentos e Aplicações**. Manole : Barueri, 2003. 550p.
- [28] SILVA, Rui Carlos Botelho Almeida da. et al. **Construção e análise do comportamento de robôs móveis aplicados na solução do problema do labirinto**. Revista Cientifico, Salvador - Bahia, v. II, p. 109-115, 2004.
- [29] GANNON, John D. et al. **SOFTWARE SPECIFICATION: A Comparison of Formal Methods**. University of Maryland : Maryland, 2001.
- [30] BAYER, Christel; KATOEN, Joost-Pieter. **Principles of Model Checking**. MIT Press : Massachusetts, 2008. 975p.
- [31] CLARKE, E. M.; LONG D. E.; McMILLAN K. L.; **Compositional Model Checking**, Proceedings of the Fourth Annual Symposium on Logic in computer science, pp.353-362, IEEE Press : Piscataway, 1989.

- [32] Behrmann G.; David A.; Larsen K. G.. **A Tutorial on Uppaal**. In: Lectures Notes in Computer Science, Formal Methods for the Design of Real-Time Systems. Springer : Berlim, 2004. 38p.
- [33] BEREZIN S.; CAMPOS S.; CLARKE, E. M.; **Compositional Reasoning in Model Checking**, International Symposium, COMPOS'97 Bad Malente, p 81-102, September 8–12, Springer-Verlag : Berlin-Heidelberg, 1998.
- [34] MELLER, Yael. **Multi-Valued Abstraction and Compositional Model Checking**. Research Thesis (Master of Science in Computer Science) - Israel Institute of Technology, Haifa, 2009.
- [35] LADEIRA, Marcelo. **Representação do Conhecimento e Rede de Decisão**. CPGCC – Universidade Federal do Rio Grande do Sul : Porto Alegre, 1997. 150p.

Apêndice A – Conjunto de Regras Instintivas e Jogadores que as utilizam

Nr	Regras Instintivas	Jogadores
1	<pre>(rule_mark_hold_ball (if (logic (local_goal current mark)) (logic (local_goal status active)) (logic (player ball_kickable true))) (then (logic (reactive_behavior active hold_ball)) (logic (local_goal status achieved)) (logic (local_goal current kick_to_goal)))))</pre>	1,2,3,4,5,6,7,8,9,10,11
2	<pre>(rule_mark_search_ball (if (logic (local_goal current mark)) (logic (local_goal status active)) (logic (ball position unknown)) (logic (player ball_kickable false))) (then (logic (reactive_behavior active search_ball)) (logic (local_goal status active)) (logic (local_goal current mark)))))</pre>	1,2,3,4,5,6,7,8,9,10,11
3	<pre>(rule_mark_mark_opponent (if (logic (local_goal current mark)) (logic (local_goal status active)) (logic (player ball_kickable false)) (logic (teammate has_ball false))) (then (logic (reactive_behavior active mark_opponent)) (logic (local_goal status activ)) (logic (local_goal current mark)))))</pre>	1,2,3,4,5
4	<pre>(rule_mark_intercept_ball (if (logic (local_goal current mark)) (logic (local_goal status active)) (logic (player ball_kickable false)) (logic (ball position known)) (logic (player fastest_to_ball true))) (then (logic (reactive_behavior active intercept_ball)) (logic (local_goal status active)) (logic (local_goal current mark)))))</pre>	1,2,3,4,5,6,7,8,9,10,11
5	<pre>(rule_mark_strategic_position (if (logic (local_goal current mark)) (logic (local_goal status active)) (logic (player ball_kickable false)) (logic (player fastest_to_ball false)) (logic (ball position known))) (then (logic (reactive_behavior active strategic_position)) (logic (local_goal status active)) (logic (local_goal current side_attack)))))</pre>	6,7,8,9,10,11
6	<pre>(rule_mark_achieved (if (logic (local_goal current mark)) (logic (local_goal status active)) (logic (teammate has_ball true))) (then (logic (local_goal status achieved)) (logic (local_goal current mark)))))</pre>	1,2,3,4,5,6,7,8,9,10,11

7	<pre>(rule_advance_search_ball (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (ball position unknown)) (logic (player ball_kickable false))) (then (logic (reactive_behavior active search_ball)) (logic (local_goal status active)) (logic (local_goal current advance)))))</pre>	2,3,4,5
8	<pre>(rule_advance_intercept_ball (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (player ball_kickable false)) (logic (ball position known)) (logic (player fastest_to_ball true))) (then (logic (reactive_behavior active intercept_ball)) (logic (local_goal status active)) (logic (local_goal current advance)))))</pre>	1,2,3,4,5
9	<pre>(rule_advance_strategic_position (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (player ball_kickable false)) (logic (player fastest_to_ball false)) (logic (ball position known))) (then (logic (reactive_behavior active strategic_position)) (logic (local_goal status active)) (logic (local_goal current advance)))))</pre>	1,2,3,4,5
10	<pre>(rule_advance_pass_ball_forward (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_forward)) (logic (local_goal status active)) (logic (local_goal current advance)))))</pre>	1, 2, 3, 4
11	<pre>(rule_advance_pass_ball_6 (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_6)) (logic (local_goal status active)) (logic (local_goal current advance)))))</pre>	5
12	<pre>(rule_advance_pass_ball_7 (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_7)) (logic (local_goal status active)) (logic (local_goal current advance)))))</pre>	2
13	<pre>(rule_advance_pass_ball_8 (if (logic (local_goal current advance))</pre>	2,5

	<pre> (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (pass_condition teammate_eight true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_8)) (logic (local_goal status active)) (logic (local_goal current advance)))) </pre>	
14	<pre> (rule_advance_drive_ball_forward (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (player ball_kickable true))) (then (logic (local_goal current advance)) (logic (local_goal status active)) (logic (reactive_behavior active drive_ball_forward))))) </pre>	6
15	<pre> (rule_advance_drive_ball_fast (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (player ball_kickable true))) (then (logic (local_goal current advance)) (logic (local_goal status active)) (logic (reactive_behavior active drive_ball_fast))))) </pre>	7
16	<pre> (rule_advance_fail (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (teammate has_ball false))) (then (logic (local_goal status fail)) (logic (local_goal current advance))))) </pre>	1,2,3,4,5
17	<pre> (rule_advance_achieved (if (logic (local_goal current advance)) (logic (local_goal status active)) (logic (midplayers has_ball true))) (then (logic (local_goal status achieved)) (logic (local_goal current advance))))) </pre>	1,4
18	<pre> (rule_side_attack_search_ball (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (ball position unknown)) (logic (player ball_kickable false))) (then (logic (reactive_behavior active search_ball)) (logic (local_goal status active)) (logic (local_goal current side_attack))))) </pre>	1,4,6,7,8,9,10,11
19	<pre> (rule_side_attack_intercept_ball (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player ball_kickable false)) (logic (ball position known)) (logic (player fastest_to_ball true))) (then (logic (reactive_behavior active intercept_ball)) (logic (local_goal status active)) (logic (local_goal current side_attack))))) </pre>	1,4,6,7,8,9,10,11

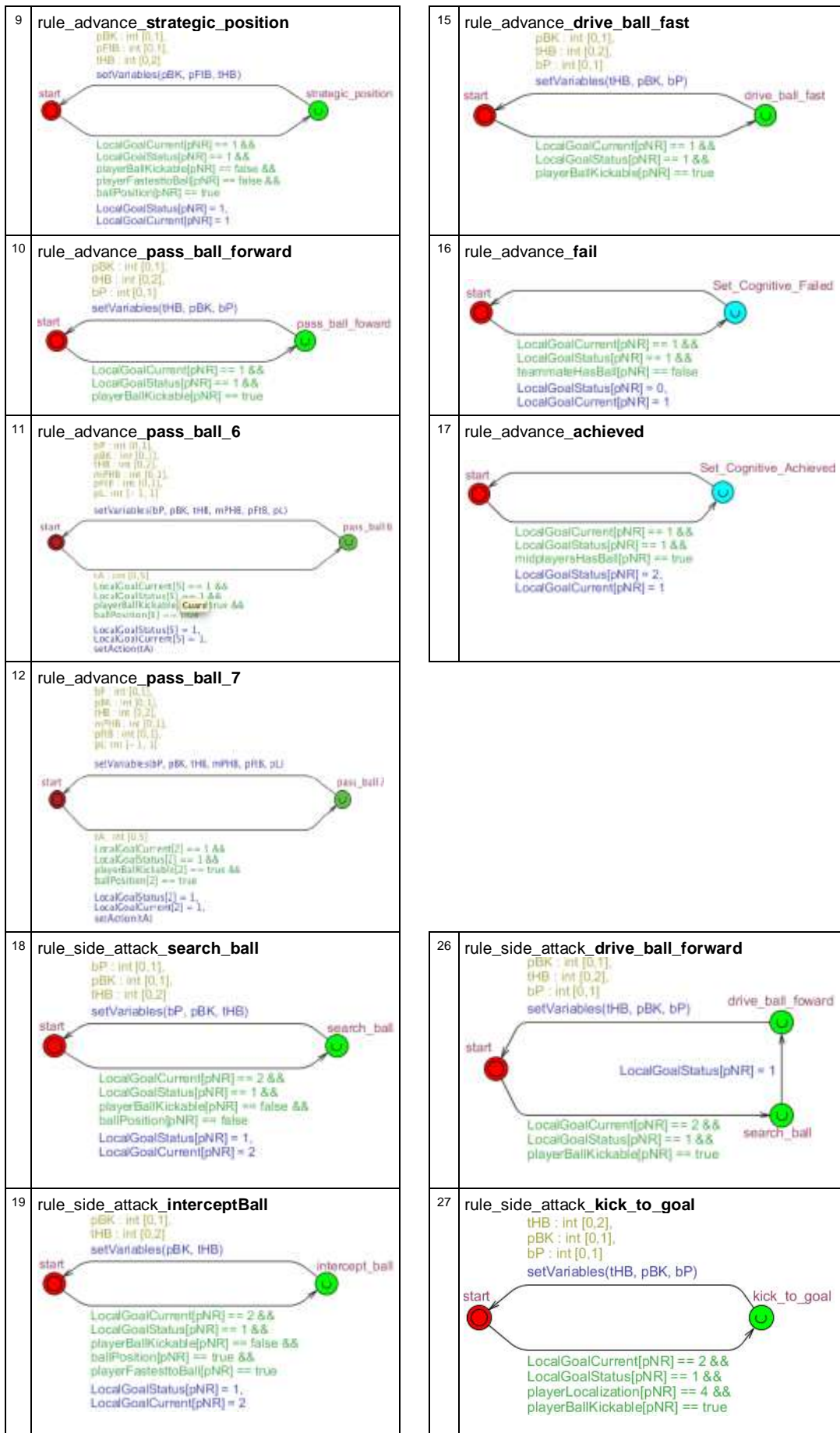
))	
20	(rule_side_attack_mark_opponent (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player ball_kickable false)) (logic (player fastest_to_ball false)) (logic (ball position known))) (then (logic (reactive_behavior active mark_opponent)) (logic (local_goal status active)) (logic (local_goal current side_attack)))))	1,4
21	(rule_side_attack_pass_ball_7 (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_7)) (logic (local_goal status active)) (logic (local_goal current side_attack)))))	1,4
22	(rule_side_attack_pass_ball_8 (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (pass_condition teammate_eight true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_8)) (logic (local_goal status active)) (logic (local_goal current side_attack)))))	6,7
23	(rule_side_attack_pass_ball_9 (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (pass_condition teammate_nine true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_9)) (logic (local_goal status active)) (logic (local_goal current side_attack)))))	6, 7, 8, 10, 11
24	(rule_side_attack_pass_ball_10 (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (pass_condition teammate_ten true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_10)) (logic (local_goal status active)) (logic (local_goal current side_attack)))))	7, 8, 9
25	(rule_side_attack_pass_ball_11 (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (pass_condition teammate_eleven true)) (logic (ball position known))) (then (logic (reactive_behavior active pass_ball_11)) (logic (local_goal status active))))	6, 8, 9

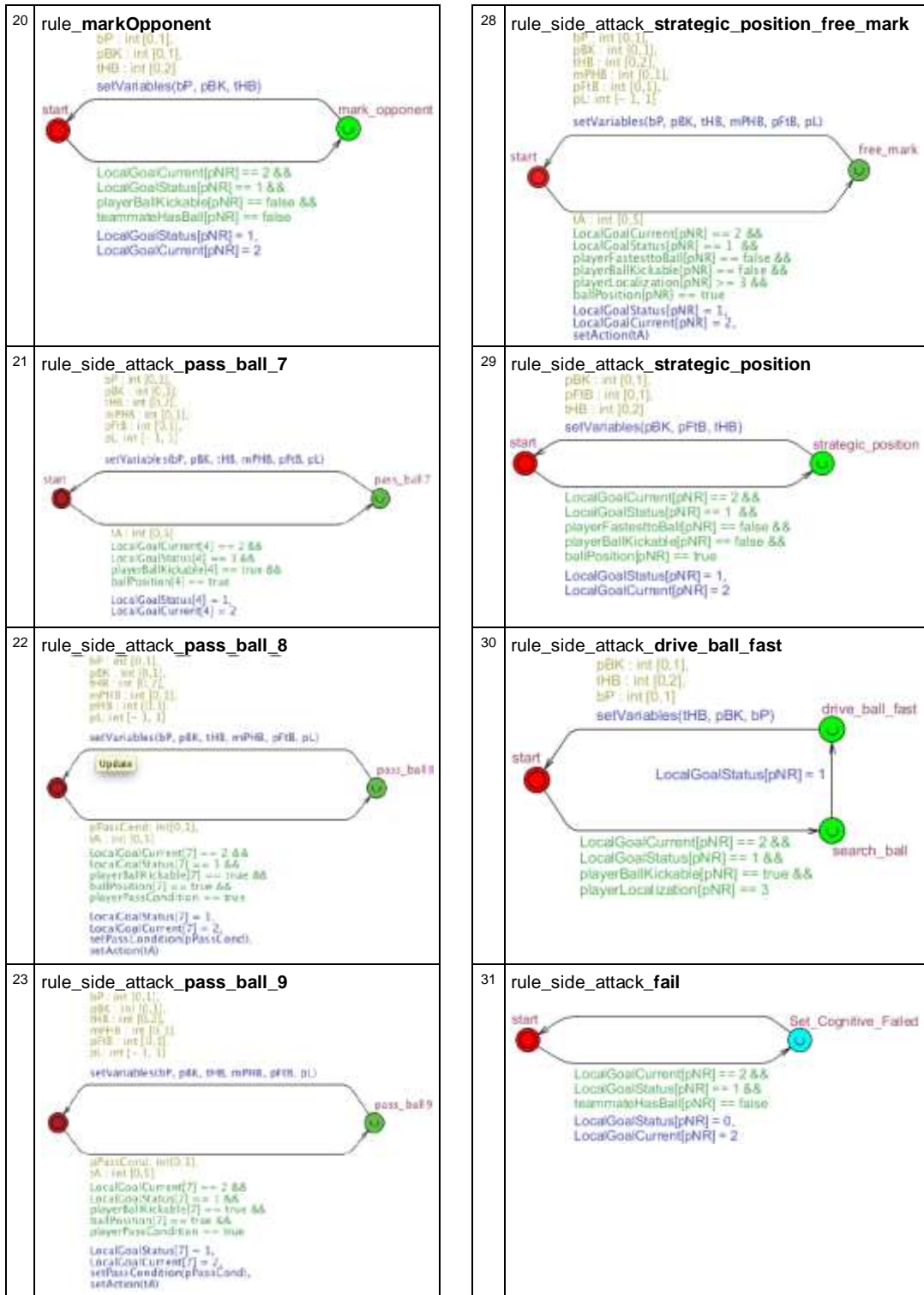
	(logic (local_goal current side_attack))))	
26	(rule_side_attack_drive_ball_forward (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player ball_kickable true))) (then (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (reactive_behavior active drive_ball_forward)))))	8, 9, 10, 11
27	(rule_side_attack_kick_to_goal (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player localization AREA_ATTACK)) (logic (player ball_kickable true))) (then (logic (reactive_behavior active kick_to_goal)))))	9,10, 11
28	(rule_side_attack_strategic_position_free_mark (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player fastest_to_ball false)) (logic (player ball_kickable false)) (logic (player localization AREA_ATTACK)) (logic (ball position known))) (then (logic (reactive_behavior active free_mark)) (logic (local_goal status active)) (logic (local_goal current side_attack)))))	6,7,8,9
29	(rule_side_attack_strategic_position (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player fastest_to_ball false)) (logic (player ball_kickable false)) (logic (ball position known))) (then (logic (reactive_behavior active strategic_position)) (logic (local_goal status active)) (logic (local_goal current side_attack)))))	9,10,11
30	(rule_side_attack_drive_ball_fast (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (player localization AREA_SIDE_ATTACK)) (logic (player ball_kickable true))) (then (logic (reactive_behavior active search_ball)) (logic (local_goal status active)) (logic (reactive_behavior active drive_ball_fast)))))	10,11
31	(rule_side_attack_fail (if (logic (local_goal current side_attack)) (logic (local_goal status active)) (logic (teammate has_ball false))) (then (logic (local_goal status fail)) (logic (local_goal current side_attack)))))	1,4,6,7,8,9,10,11
32	(rule_side_attack_achieved (if (logic (local_goal current side_attack)) (logic (local_goal status active))))	1,4

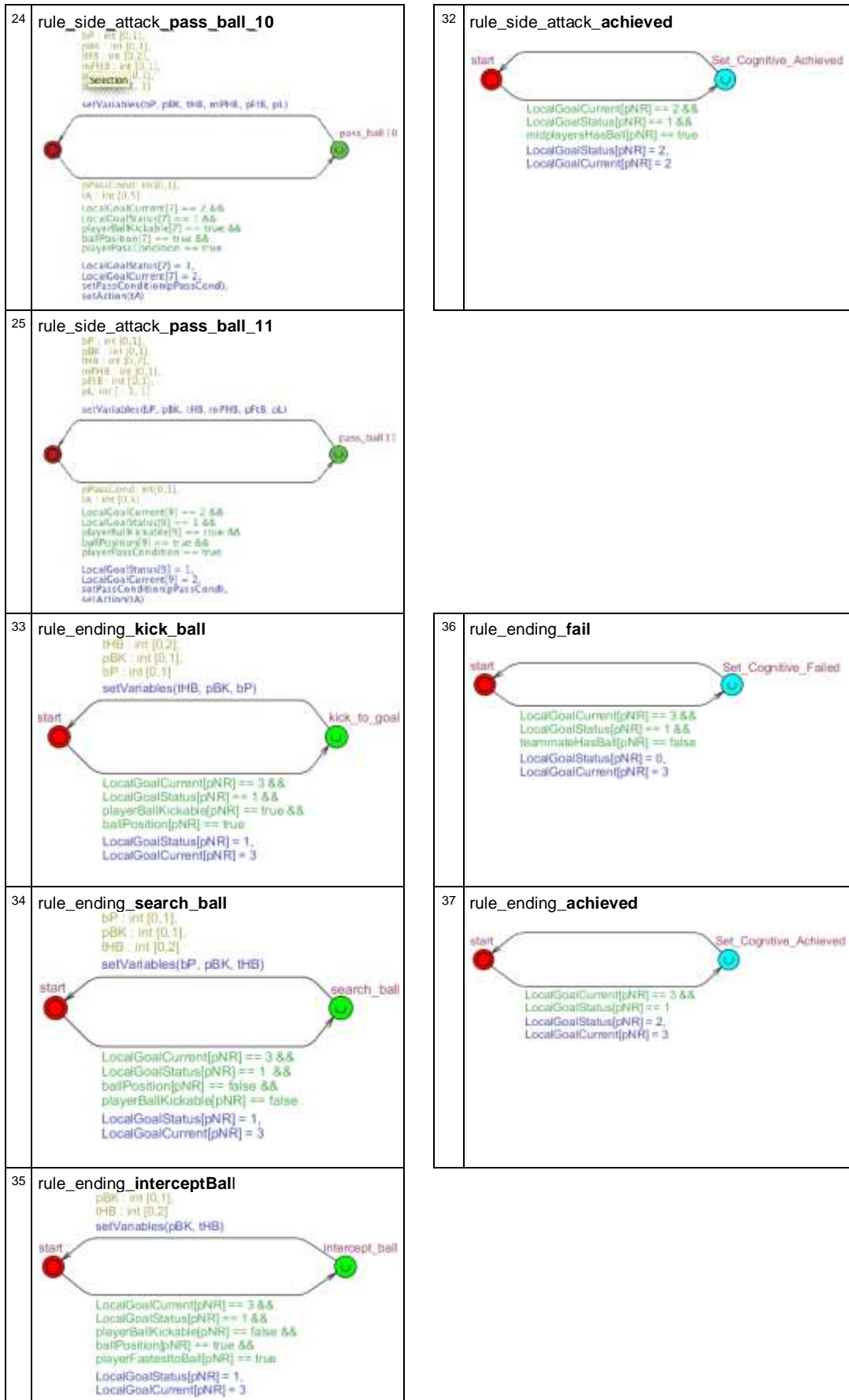
	<pre> (logic (midplayers has_ball true))) (then (logic (local_goal status achieved)) (logic (local_goal current side_attack)))) </pre>	
33	<pre> (rule_ending_kick_ball (if (logic (local_goal current ending)) (logic (local_goal status active)) (logic (player ball_kickable true)) (logic (ball position known))) (then (logic (reactive_behavior active kick_to_goal)) (logic (local_goal status active)) (logic (local_goal current ending))))) </pre>	1
34	<pre> (rule_ending_search_ball (if (logic (local_goal current ending)) (logic (local_goal status active)) (logic (ball position unknown)) (logic (player ball_kickable false))) (then (logic (reactive_behavior active search_ball)) (logic (local_goal status active)) (logic (local_goal current ending))))) </pre>	1
35	<pre> (rule_ending_intercept_ball (if (logic (local_goal current ending)) (logic (local_goal status active)) (logic (player ball_kickable false)) (logic (ball position known)) (logic (player fastest_to_ball true))) (then (logic (reactive_behavior active intercept_ball)) (logic (local_goal status active)) (logic (local_goal current ending))))) </pre>	1
36	<pre> (rule_ending_fail (if (logic (local_goal current ending)) (logic (local_goal status active)) (logic (teammate has_ball false))) (then (logic (local_goal status fail)) (logic (local_goal current ending))))) </pre>	1
37	<pre> (rule_ending_achieved (if (logic (local_goal current ending)) (logic (local_goal status active)) (logic (reactive_behavior active kick_to_goal))) (then (logic (local_goal status achieved)) (logic (local_goal current ending))))) </pre>	1

Apêndice B – Especificação das Regras Instintivas para Verificação Individual de Planos

Nr	Regras Instintivas	Nr	Regras Instintivas
1	<p>rule_mark_hold_ball</p> <p>bP : int [0, 1], pBK : int [0, 1], tHB : int [0, 2], pL : int [- 1, 1]</p> <p>start → setVariables(bP, pBK, tHB, pL) → hold_ball</p> <p>LocalGoalCurrent[pNR] == 0 && LocalGoalStatus[pNR] == 1 && playerBallKickable[pNR] == true && LocalGoalStatus[pNR] = 1, LocalGoalCurrent[pNR] = 0</p>	4	<p>rule_mark_interceptBall</p> <p>pBK : int [0, 1], tHB : int [0, 2]</p> <p>start → setVariables(pBK, tHB) → intercept_ball</p> <p>LocalGoalCurrent[pNR] == 0 && LocalGoalStatus[pNR] == 1 && playerBallKickable[pNR] == false && ballPosition[pNR] == true && playerFastesttoBall[pNR] == true LocalGoalStatus[pNR] = 1, LocalGoalCurrent[pNR] = 0</p>
2	<p>rule_mark_search_ball</p> <p>bP : int [0, 1], pBK : int [0, 1], tHB : int [0, 2], pL : int [- 1, 1]</p> <p>start → setVariables(bP, pBK, tHB, pL) → search_ball</p> <p>LocalGoalCurrent[pNR] == 0 && LocalGoalStatus[pNR] == 1 && playerBallKickable[pNR] == false && ballPosition[pNR] == false LocalGoalStatus[pNR] = 1, LocalGoalCurrent[pNR] = 0</p>	5	<p>rule_mark_strategic_position</p> <p>pBK : int [0, 1], pFB : int [0, 1], tHB : int [0, 2]</p> <p>start → setVariables(pBK, pFB, tHB) → strategic_position</p> <p>LocalGoalCurrent[pNR] == 0 && LocalGoalStatus[pNR] == 1 && playerBallKickable[pNR] == false && playerFastesttoBall[pNR] == false && ballPosition[pNR] == true LocalGoalStatus[pNR] = 1, LocalGoalCurrent[pNR] = 0</p>
3	<p>rule_mark_opponent</p> <p>bP : int [0, 1], pBK : int [0, 1], tHB : int [0, 2]</p> <p>start → setVariables(bP, pBK, tHB) → mark_opponent</p> <p>LocalGoalCurrent[pNR] == 0 && LocalGoalStatus[pNR] == 1 && playerBallKickable[pNR] == false && teammateHasBall[pNR] == false LocalGoalStatus[pNR] = 1, LocalGoalCurrent[pNR] = 0</p>	6	<p>rule_mark_achieved</p> <p>start → Set Cognitive Achieved</p> <p>LocalGoalCurrent[pNR] == 0 && LocalGoalStatus[pNR] == 1 && teammateHasBall[pNR] == true LocalGoalStatus[pNR] = 2, LocalGoalCurrent[pNR] = 0</p>
7	<p>rule_advance_search_ball</p> <p>bP : int [0, 1], pBK : int [0, 1], tHB : int [0, 2]</p> <p>start → setVariables(bP, pBK, tHB) → search_ball</p> <p>LocalGoalCurrent[pNR] == 1 && LocalGoalStatus[pNR] == 1 && playerBallKickable[pNR] == false && ballPosition[pNR] == false LocalGoalStatus[pNR] = 1, LocalGoalCurrent[pNR] = 1</p>	13	<p>rule_advance_pass_ball_8</p> <p>bP : int [0, 1], pBK : int [0, 1], tHB : int [0, 2], mPHB : int [0, 1], pFB : int [0, 1], pL : int [- 1, 1]</p> <p>start → setVariables(bP, pBK, tHB, mPHB, pFB, pL) → pass_ball_8</p> <p>pPassCond : int[0, 1], tA : int [0, 1]</p> <p>LocalGoalCurrent[2] == 1 && LocalGoalStatus[2] == 1 && playerBallKickable[2] == true && ballPosition[2] == true && playerPassCondition == true LocalGoalStatus[2] = 1, LocalGoalCurrent[2] = 1, setPassCondition(pPassCond), setAction(tA)</p>
8	<p>rule_advance_interceptBall</p> <p>pBK : int [0, 1], tHB : int [0, 2]</p> <p>start → setVariables(pBK, tHB) → intercept_ball</p> <p>LocalGoalCurrent[pNR] == 1 && LocalGoalStatus[pNR] == 1 && playerBallKickable[pNR] == false && ballPosition[pNR] == true && playerFastesttoBall[pNR] == true LocalGoalStatus[pNR] = 1, LocalGoalCurrent[pNR] = 1</p>	14	<p>rule_advance_drive_ball_forward</p> <p>bP : int [0, 1], pBK : int [0, 1], tHB : int [0, 2], mPHB : int [0, 1], pFB : int [0, 1], pL : int [- 1, 1]</p> <p>start → setVariables(bP, pBK, tHB, mPHB, pFB, pL) → drive_ball_forward</p> <p>LocalGoalCurrent[pNR] == 1 && LocalGoalStatus[pNR] == 1 && playerBallKickable[pNR] == true LocalGoalStatus[pNR] = 1, LocalGoalCurrent[pNR] = 1</p>







Apêndice C – Funções *setOpponentInformations()* e *setInitReinitValues()* do ambiente

```

1 void setOpponentInformations(int oHB, int typeOfAction, int succeedOfAction)
2 {
3     if(SoccerServerPlayerHoldingTheBall < 0) //Verify if opponent team has ball
4     {
5         if(SoccerServerBallPosition <= 1) //Verify if the ball is at 0 ou 1 field areas
6         {
7             if(oHB > -6) // Using the random variable oHB's value to verify if opponent is trying to kick to goal
8             {
9                 if((SoccerServerBallPosition + oHB) < -1) // Goal kick fail
10                {
11                    SoccerServerPlayerHoldingTheBall = 1;
12                    SoccerServerBallPosition = 0;
13                    playerBallKickable [1] = true;
14                    ballPosition [1] = true;
15                    playerLocalization [1] = 0;
16                    playerBallLocalization [1] = 0;
17                    playerHoldingTheBall [1] = 0;
18                }
19                else
20                {
21                    if((SoccerServerBallPosition + oHB) == -1) // Goal Scored
22                    {
23                        goalScored = -1;
24                    }
25                    else
26                    {
27                        if((SoccerServerPlayerHoldingTheBall + succeedOfAction) >= -11)
28                        {
29                            SoccerServerPlayerHoldingTheBall = SoccerServerPlayerHoldingTheBall +
30                            succeedOfAction;
31                            if((SoccerServerBallPosition + oHB) >= 0)
32                            {
33                                SoccerServerBallPosition = SoccerServerBallPosition + oHB;
34                            }
35                        }
36                        else
37                        {
38                            SoccerServerPlayerHoldingTheBall = 0;
39                        }
40                    }
41                }
42            }
43        }
44    }
45    else
46    {
47        if(SoccerServerBallPosition < 5)
48        {
49            if(succeedOfAction > 0)
50            {
51                SoccerServerBallPosition = SoccerServerBallPosition + typeOfAction;
52                SoccerServerPlayerHoldingTheBall = SoccerServerPlayerHoldingTheBall + typeOfAction;
53            }
54            else
55            {
56                SoccerServerBallPosition = SoccerServerBallPosition + succeedOfAction;
57                SoccerServerPlayerHoldingTheBall = SoccerServerPlayerHoldingTheBall + succeedOfAction;
58            }
59        }
60        else
61        {
62            if(succeedOfAction > 0)
63            {
64                SoccerServerBallPosition = SoccerServerBallPosition + typeOfAction;
65                SoccerServerPlayerHoldingTheBall = SoccerServerPlayerHoldingTheBall + typeOfAction;
66            }
67            else
68            {
69                SoccerServerBallPosition = SoccerServerBallPosition + typeOfAction;
70                SoccerServerPlayerHoldingTheBall = 0;
71            }
72        }
73    }
74 }

```

```

71     }
72   }
73 }
74 else
75 {
76   if (SoccerServerBallPosition == 0)
77   {
78     if(succeedOfAction > 0) SoccerServerPlayerHoldingTheBall = -9;
79   }
80
81   if(SoccerServerBallPosition == 1)
82   {
83     if(succeedOfAction > 0)
84     {
85       if(oHB < -6) SoccerServerPlayerHoldingTheBall = -11;
86       else SoccerServerPlayerHoldingTheBall = -10;
87     }
88   }
89
90   if(SoccerServerBallPosition == 2)
91   {
92     if(succeedOfAction > 0)
93     {
94       if(oHB < -6) SoccerServerPlayerHoldingTheBall = -8;
95       else SoccerServerPlayerHoldingTheBall = -7;
96     }
97   }
98
99   if(SoccerServerBallPosition == 3)
100  {
101    if(succeedOfAction > 0)
102    {
103      if(oHB < -6) SoccerServerPlayerHoldingTheBall = -6;
104      else SoccerServerPlayerHoldingTheBall = -5;
105    }
106  }
107
108  if(SoccerServerBallPosition == 4)
109  {
110    if(succeedOfAction > 0)
111    {
112      if(oHB < -9) SoccerServerPlayerHoldingTheBall = -5;
113      else
114      {
115        if(oHB < -6) SoccerServerPlayerHoldingTheBall = -4;
116        else
117        {
118          if(oHB < -4) SoccerServerPlayerHoldingTheBall = -3;
119          else SoccerServerPlayerHoldingTheBall = -2;
120        }
121      }
122    }
123  }
124
125  if(SoccerServerBallPosition == 5)
126  {
127    if(succeedOfAction > 0)
128    {
129      if(oHB < -10) SoccerServerPlayerHoldingTheBall = -4;
130      else
131      {
132        if(oHB < -7) SoccerServerPlayerHoldingTheBall = -3;
133        else
134        {
135          if(oHB < -5) SoccerServerPlayerHoldingTheBall = -2;
136          else SoccerServerPlayerHoldingTheBall = -1;
137        }
138      }
139    }
140  }
141 }
142 }

```

```

1 void setInitReinitValues()
2 {
3     goalScored = false;
4     playerHoldingTheBall [1] = playerHoldingTheBall [2] = playerHoldingTheBall [3] = playerHoldingTheBall
5 [4] = SoccerServerPlayerHoldingTheBall;
6     playerHoldingTheBall [5] = playerHoldingTheBall [6] = playerHoldingTheBall [7] = playerHoldingTheBall
7 [8] = SoccerServerPlayerHoldingTheBall;
8     playerHoldingTheBall [9] = playerHoldingTheBall [10] = playerHoldingTheBall [11] =
9 SoccerServerPlayerHoldingTheBall;
10
11     ballPosition [1] = ballPosition [2] = ballPosition [3] = ballPosition [4] = ballPosition [5] = ballPosition [6] =
12 true;
13     ballPosition [7] = ballPosition [8] = ballPosition [9] = ballPosition [10] = ballPosition [11] = true;
14
15     playerLocalization [1] = playerLocalization [2] = playerLocalization [3] = 0;
16     playerLocalization [4] = playerLocalization [5] = 1;
17     playerLocalization [6] = playerLocalization [7] = playerLocalization [8] = 2;
18     playerLocalization [9] = playerLocalization [10] = playerLocalization [11] = 2;
19
20     playerFastesttoBall [1] = playerFastesttoBall [2] = playerFastesttoBall [3] = playerFastesttoBall [4] =
21 playerFastesttoBall [5] = false;
22     playerFastesttoBall [6] = playerFastesttoBall [7] = playerFastesttoBall [8] = playerFastesttoBall [9] =
23 playerFastesttoBall [11] = false;
24
25     playerBallKickable [1] = playerBallKickable [2] = playerBallKickable [3] = playerBallKickable [4] =
26 playerBallKickable [5] = false;
27     playerBallKickable [6] = playerBallKickable [7] = playerBallKickable [8] = playerBallKickable [9] =
28 playerBallKickable [11] = false;
29
30     if(SoccerServerPlayerHoldingTheBall > 0) //Our team is initiating or reinitiating the game
31     {
32         playerFastesttoBall [10] = true;
33         playerBallKickable [10] = true;
34
35         teammateHasBall [1] = teammateHasBall [2] = teammateHasBall [3] = teammateHasBall [4] =
36 teammateHasBall [5] = teammateHasBall [6] = true;
37         teammateHasBall [7] = teammateHasBall [8] = teammateHasBall [9] = teammateHasBall [10] =
38 teammateHasBall [11] = true;
39
40         midplayersHasBall [1] = midplayersHasBall [2] = midplayersHasBall [3] = midplayersHasBall [4] =
41 midplayersHasBall [5] = true;
42         midplayersHasBall [6] = midplayersHasBall [7] = midplayersHasBall [8] = midplayersHasBall [9] =
43 midplayersHasBall [10] = true;
44         midplayersHasBall [11] = true;
45
46         playerBallLocalization [1] = playerBallLocalization [2] = playerBallLocalization [3] =
47 playerBallLocalization [4] = 2;
48         playerBallLocalization [5] = playerBallLocalization [6] = playerBallLocalization [7] =
49 playerBallLocalization [8] = 2;
50         playerBallLocalization [9] = playerBallLocalization [10] = playerBallLocalization [11] = 2;
51
52         if(gameStarted == true)
53         {
54             LocalGoalCurrent [1] = LocalGoalCurrent [2] = LocalGoalCurrent [3] = 1;
55             LocalGoalCurrent [4] = LocalGoalCurrent [5] = 1;
56             LocalGoalCurrent [6] = LocalGoalCurrent [7] = LocalGoalCurrent [8] = 2;
57             LocalGoalCurrent [9] = LocalGoalCurrent [10] = LocalGoalCurrent [11] = 2;
58
59             LocalGoalStatus [1] = LocalGoalStatus [2] = LocalGoalStatus [3] = 1;
60             LocalGoalStatus [4] = LocalGoalStatus [5] = LocalGoalStatus [6] = 1;
61             LocalGoalStatus [7] = LocalGoalStatus [8] = LocalGoalCurrent [9] = 1;
62             LocalGoalStatus [10] = LocalGoalStatus [11] = 1;
63         }
64     }
65     else
66     {
67         playerFastesttoBall [10] = true;
68         playerBallKickable [10] = false;
69
70         teammateHasBall [1] = teammateHasBall [2] = teammateHasBall [3] = teammateHasBall [4] =
71 teammateHasBall [5] = teammateHasBall [6] = false;

```

```

72     teammateHasBall [7] = teammateHasBall [8] = teammateHasBall [9] = teammateHasBall [10] =
73     teammateHasBall [11] = false;
74
75     midplayersHasBall [1] = midplayersHasBall [2] = midplayersHasBall [3] = midplayersHasBall [4] =
76     midplayersHasBall [5] = false;
77     midplayersHasBall [6] = midplayersHasBall [7] = midplayersHasBall [8] = midplayersHasBall [9] =
78     midplayersHasBall [10] = false;
79     midplayersHasBall [11] = false;
80
81     playerBallLocalization [1] = playerBallLocalization [2] = playerBallLocalization [3] =
82     playerBallLocalization [4] = 3;
83     playerBallLocalization [5] = playerBallLocalization [6] = playerBallLocalization [7] =
84     playerBallLocalization [8] = 3;
85     playerBallLocalization [9] = playerBallLocalization [10] = playerBallLocalization [11] = 3;
86
87     if(gameStarted == true)
88     {
89
90         LocalGoalCurrent [1] = LocalGoalCurrent [2] = LocalGoalCurrent [3] = 1;
91         LocalGoalCurrent [4] = LocalGoalCurrent [5] = 1;
92         LocalGoalCurrent [6] = LocalGoalCurrent [7] = LocalGoalCurrent [8] = 2;
93         LocalGoalCurrent [9] = LocalGoalCurrent [10] = LocalGoalCurrent [11] = 2;
94
95         LocalGoalStatus [1] = LocalGoalStatus [2] = LocalGoalStatus [3] = 1;
96         LocalGoalStatus [4] = LocalGoalStatus [5] = LocalGoalStatus [6] = 1;
97         LocalGoalStatus [7] = LocalGoalStatus [8] = LocalGoalStatus [9] = 1;
98         LocalGoalStatus [10] = LocalGoalStatus [11] = 1;
99         SoccerServerBallPosition = 2;
100        SoccerServerPlayerHoldingTheBall = 10;
101    }
102
103 }
104 EnvironmentInfo[1]=false;
105 EnvironmentInfo[2]=false;
106 EnvironmentInfo[3]=false;
107 EnvironmentInfo[4]=false;
108 EnvironmentInfo[5]=false;
109 EnvironmentInfo[6]=false;
110 EnvironmentInfo[7]=false;
111 EnvironmentInfo[8]=false;
112 EnvironmentInfo[9]=false;
113 EnvironmentInfo[10]=false;
114 EnvironmentInfo[11]=false;
115 gameStarted = true;
116 }

```

Apêndice D – Função setInitialValues() das regras cognitivas

```
1 void setInitialValues()
2 {
3     playerHoldingTheBall [pNR] = SoccerServerPlayerHoldingTheBall;
4     ballPosition [pNR] = true;
5     playerLocalization [pNR] = 0;
6     playerFastesttoBall [pNR] = false;
7     playerBallKickable [pNR] = false;
8     if(SoccerServerPlayerHoldingTheBall > 0) //Our team is initiating the game
9     {
10        teammateHasBall [pNR] = true;
11        midplayersHasBall [pNR] = true;
12        playerBallLocalization [pNR] = 2;
13    }
14    else
15    {
16        teammateHasBall [pNR] = false;
17        midplayersHasBall [pNR] = false;
18        playerBallLocalization [pNR] = 3;
19    }
20 }
21
```


Apêndice E – Função setVariables() das regras instintivas

```
1 void setVariables(int pBP, int pPBK, int pTHB, int pMPHB, int pPFtB, int pL)
2 {
3   if( playerLocalization[pNR] == SoccerServerBallPosition ){
4     if( pNR == SoccerServerPlayerHoldingTheBall ){
5       ballPosition[pNR] = true; //player knows where is the ball
6       teammateHasBall[pNR] = true;
7       playerBallKickable[pNR] = true;
8       playerFastesttoBall[pNR] = true;
9       if(( pNR >= 6 ) && ( pNR <= 8 ) midplayersHasBall[pNR] = true;
10      else midplayersHasBall[pNR] = false;
11    }
12    else{
13      if( pBP >= 1 ){
14        ballPosition[pNR] = true; //It's true that player knows where is the ball
15        playerHoldingTheBall [pNR] = SoccerServerPlayerHoldingTheBall;
16        if( SoccerServerPlayerHoldingTheBall >= 1 ){ // Team is holding the ball
17          teammateHasBall[pNR] = true;
18          playerBallKickable[pNR] = false;
19          playerFastesttoBall[pNR] = false;
20          if(( pNR >= 6 ) && ( pNR <= 8 )) midplayersHasBall[pNR] = true;
21          else{
22            if(( SoccerServerPlayerHoldingTheBall >= 6 ) && ( SoccerServerPlayerHoldingTheBall <= 8
23            )) midplayersHasBall[pNR] = true;
24            else midplayersHasBall[pNR] = false;
25          }
26        }
27        else{ // Team isn't holding the ball
28          teammateHasBall[pNR] = false;
29          midplayersHasBall[pNR] = false;
30          if(pPBK == 1){
31            playerBallKickable[pNR] = true;
32            playerFastesttoBall[pNR] = true;
33          }
34          else{
35            playerBallKickable[pNR] = false;
36            if (pPFtB == 1) playerFastesttoBall[pNR] = true;
37            else playerFastesttoBall[pNR] = false;
38          }
39        }
40      }
41      else{ // Player doesn't know anything about the game
42        ballPosition[pNR] = false;
43        teammateHasBall[pNR] = false;
44        midplayersHasBall[pNR] = false;
45        playerBallKickable[pNR] = false;
46        playerFastesttoBall[pNR] = false;
47      }
48    }
49  }
50  else{
51    if( ((SoccerServerBallPosition - playerLocalization[pNR] ) == 1 ) || ((SoccerServerBallPosition -
52    playerLocalization[pNR] ) == -1 ) ){
53      if( pBP >= 2 ){
54        ballPosition[pNR] = true; //player knows where is the ball
55        if(pTHB == 1){
56          teammateHasBall[pNR] = true;
57          playerBallKickable[pNR] = false; // player can't kick the ball
58          playerFastesttoBall[pNR] = false;
59          if (pMPHB == 1) midplayersHasBall[pNR] = true;
60          else midplayersHasBall[pNR] = false;
61        }
62        else{
63          teammateHasBall[pNR] = false;
64          midplayersHasBall[pNR] = false;
65          if(pPBK == 1){
66            playerBallKickable[pNR] = true;
67            playerFastesttoBall[pNR] = true;
68          }
69          else{
70            playerBallKickable[pNR] = false;
71            if (pPFtB == 1) playerFastesttoBall[pNR] = true;
72            else playerFastesttoBall[pNR] = false;

```

```

73     }
74     }
75     }
76     else{
77         ballPosition[pNR] = false;
78         teammateHasBall[pNR] = false;
79         midplayersHasBall[pNR] = false;
80         playerBallKickable[pNR] = false; // player can't kick the ball
81         playerFastesttoBall[pNR] = false;
82     }
83 }
84
85 if( ((SoccerServerBallPosition - playerLocalization[pNR] ) == 2 ) || ((SoccerServerBallPosition -
86 playerLocalization[pNR] ) == -2 ) ){
87     if( pBP >= 3 ){
88         ballPosition[pNR] = true; //player knows where is the ball
89         if(pTHB == 1){
90             teammateHasBall[pNR] = true;
91             playerBallKickable[pNR] = false;
92             playerFastesttoBall[pNR] = false;
93             if (pMPHB == 1) midplayersHasBall[pNR] = true;
94             else midplayersHasBall[pNR] = false;
95         }
96         else{
97             teammateHasBall[pNR] = false;
98             midplayersHasBall[pNR] = false;
99             if(pPBK == 1){
100                playerBallKickable[pNR] = true;
101                playerFastesttoBall[pNR] = true;
102            }
103            else{
104                playerBallKickable[pNR] = false;
105                if (pPFtB == 1) playerFastesttoBall[pNR] = true;
106                else playerFastesttoBall[pNR] = false;
107            }
108        }
109    }
110    else{
111        ballPosition[pNR] = false;
112        teammateHasBall[pNR] = false;
113        midplayersHasBall[pNR] = false;
114        playerBallKickable[pNR] = false;
115        playerFastesttoBall[pNR] = false;
116    }
117 }
118
119 if( ((SoccerServerBallPosition - playerLocalization[pNR] ) == 3 ) || ((SoccerServerBallPosition -
120 playerLocalization[pNR] ) == -3 ) ){
121     if( pBP >= 3 ){
122         ballPosition[pNR] = true; //player knows where is the ball
123         if(pTHB == 1){
124             teammateHasBall[pNR] = true;
125             playerBallKickable[pNR] = false; // player can't kick the ball
126             playerFastesttoBall[pNR] = false;
127             if (pMPHB == 1) midplayersHasBall[pNR] = true;
128             else midplayersHasBall[pNR] = false;
129         }
130         else{
131             teammateHasBall[pNR] = false;
132             midplayersHasBall[pNR] = false;
133             if(pPBK == 1){
134                 playerBallKickable[pNR] = true;
135                 playerFastesttoBall[pNR] = true;
136             }
137             else{
138                 playerBallKickable[pNR] = false;
139                 if (pPFtB == 1) playerFastesttoBall[pNR] = true;
140                 else playerFastesttoBall[pNR] = false;
141             }
142         }
143     }
144     else{
145         ballPosition[pNR] = false;
146         teammateHasBall[pNR] = false;
147         midplayersHasBall[pNR] = false;
148         playerBallKickable[pNR] = false; // player can't kick the ball

```

```

149     playerFastesttoBall[pNR] = false;
150     }
151     }
152
153     if( ((SoccerServerBallPosition - playerLocalization[pNR] ) >= 4 ) || ((SoccerServerBallPosition -
154 playerLocalization[pNR] ) <= -4 ) ){
155         if( pBP >= 4 ){ // if the parameter (pBP) is equals 1 then player knows where is the ball
156             ballPosition[pNR] = true; //player knows where is the ball
157             if(pTHB == 1){
158                 teammateHasBall[pNR] = true;
159                 playerBallKickable[pNR] = false; // player can't kick the ball
160                 playerFastesttoBall[pNR] = false;
161                 if (pMPHB == 1) midplayersHasBall[pNR] = true;
162                 else midplayersHasBall[pNR] = false;
163             }
164             else{
165                 teammateHasBall[pNR] = false;
166                 midplayersHasBall[pNR] = false;
167                 if(pPBK == 1){
168                     playerBallKickable[pNR] = true;
169                     playerFastesttoBall[pNR] = true;
170                 }
171                 else{
172                     playerBallKickable[pNR] = false;
173                     if (pPFtB == 1) playerFastesttoBall[pNR] = true;
174                     else playerFastesttoBall[pNR] = false;
175                 }
176             }
177         }
178     else{
179         ballPosition[pNR] = false;
180         teammateHasBall[pNR] = false;
181         midplayersHasBall[pNR] = false;
182         playerBallKickable[pNR] = false; // player can't kick the ball
183         playerFastesttoBall[pNR] = false;
184     }
185 }
186 }
187 }

```

Apêndice F – Função `setAction()` das regras instintivas com comportamentos agrupáveis e não agrupáveis entre si

1) Implementação das funções `setAction()` dos grupos de regras que possuem comportamentos agrupáveis entre si.

As regras que acionam o comportamento *Search_Ball* tiveram suas funções `setAction()` implementadas com o mesmo algoritmo. Este algoritmo tem como principal objetivo fazer com que cada jogador, que utiliza uma destas regras, seja capaz de identificar a posição da bola no campo a partir da sua posição relativa a esta. Assim, considerando que quanto mais distante da bola, menor a precisão na determinação da sua real posição no campo, a função utiliza a variável *selection tA* para determinar a capacidade de cada jogador, efetivamente, identificar a posição da bola, de acordo com o código da função apresentado no quadro 20.

```
1 void setAction(int tA) // Try to search ball
2 {
3     if( playerLocalization[pNR] == SoccerserverBallPosition )
4         if( tA >=1 )playerBallLocalization [pNR] = SoccerserverBallPosition;
5     else
6     {
7         if( ((SoccerserverBallPosition - playerLocalization[pNR] ) == 1 ) || ((SoccerserverBallPosition -
8 playerLocalization[pNR] ) == -1 ) )
9             if( tA >= 2 ) playerBallLocalization [pNR] = SoccerserverBallPosition;
10
11         if( ((SoccerserverBallPosition - playerLocalization[pNR] ) == 2 ) || ((SoccerserverBallPosition -
12 playerLocalization[pNR] ) == -2 ) )
13             if( tA >= 3 ) playerBallLocalization [pNR] = SoccerserverBallPosition;
14
15         if( ((SoccerserverBallPosition - playerLocalization[pNR] ) == 3 ) || ((SoccerserverBallPosition -
16 playerLocalization[pNR] ) == -3 ) )
17             if( tA >= 4 ) playerBallLocalization [pNR] = SoccerserverBallPosition;
18
19         if( ((SoccerserverBallPosition - playerLocalization[pNR] ) >= 4 ) || ((SoccerserverBallPosition -
20 playerLocalization[pNR] ) <= -4 ) )
21             if( tA >= 5 ) playerBallLocalization [pNR] = SoccerserverBallPosition;
22     }
23 }
```

Quadro 19. Código da função `setAction()` das regras que acionam o comportamento reativo *Search_Ball*.

Deste modo, considerando que a variável *tA* pode assumir valores inteiro de 0 a 5 ($tA == 0 || 1 || 2 || 3 || 4 || 5$), os jogadores que estão na mesma região do campo que a bola, possuem 5 chances em 6 ($\approx 83\%$) de conseguir efetivar a identificação da real posição da bola, caso o valor (pseudo)randomicamente atribuído à variável *tA* seja $tA \geq 1$. Os jogadores que estão a uma distância igual a 01 (uma) da região aonde a bola se encontra possuem 4 em 6 ($\approx 66\%$) chances de identificar a posição da bola. Caso os jogadores estejam a uma distância igual a 02 (dois) da região aonde a bola se encontra, estes possuem 3 em 6 (50%) chances de saber aonde a bola está. Já na situação em que os jogadores estão a uma

distância igual a 03 (três) da região onde a bola está situada, possuem 2 em 6 ($\approx 33\%$) chances de saber a localização da bola. Por fim, se os jogadores estiverem a uma distância igual ou superior a 04 (quatro), os mesmos possuem 1 em 6 ($\approx 17\%$) chances de saber a posição da bola.

As regras que acionam o comportamento reativo *Intercept_Ball* tiveram a implementação de acordo com o código apresentado no quadro 21.

```

1 void setAction(int tA) // Try to intercept ball
2 {
3     if( playerLocalization[pNR] == SoccerserverBallPosition ) // Player is really at the same place of the ball
4     {
5         if( SoccerserverPlayerHoldingTheBall == 0 ) // No player is holding the ball
6         {
7             if( tA >= 1 ) SoccerserverPlayerHoldingTheBall = pNR;
8         }
9         else
10        {
11            if( SoccerserverPlayerHoldingTheBall < 0 ) // Opponent team is holding the ball
12            {
13                if(pNR == 1) //Keeper
14                {
15                    if(playerHoldingTheBall[pNR] == 0)
16                    if(tA >= 3) SoccerserverPlayerHoldingTheBall = pNR;
17                    else
18                    if(tA >= 1) SoccerserverPlayerHoldingTheBall = pNR;
19                }
20                if((pNR >= 2)&&(pNR <= 6)) //Defensive players + midfielder 6
21                if(tA >= 2) SoccerserverPlayerHoldingTheBall = pNR;
22                if((pNR == 7)||pNR == 8)||pNR == 10)||pNR == 11)) //Mid-Attack players
23                if(tA >= 1) SoccerserverPlayerHoldingTheBall = pNR;
24                if(pNR == 9) //Attack player
25                if(tA == 0) SoccerserverPlayerHoldingTheBall = pNR;
26                playerHoldingTheBall[pNR] = SoccerserverPlayerHoldingTheBall;
27            }
28        }
29    }
30    else // Player is not at the same place of the ball
31    {
32        if(pNR != 1) // The player is not the keeper
33        {
34            // the player moves backward
35            if(playerBallLocalization [pNR] - SoccerserverBallPosition > 0) playerBallLocalization [pNR]--;
36            else playerBallLocalization [pNR]++; // the player moves forward
37        }
38    }

```

Quadro 20. Código da função *setAction()* das regras que acionam o comportamento reativo *Intercept_Ball*.

Esta função tem por objetivo permitir ao jogador tentar interceptar a bola ou se posicionar melhor para fazê-lo. Tal fato só acontece quando o jogador que está executando está na mesma posição da bola, pois não caberia ocorrer interceptação sem esta condição. Neste caso, se nenhum jogador das duas equipes não estiver de posse da bola (*SoccerserverPlayerHoldingTheBall* == 0), então as chances de um dos jogadores da equipe conseguir interceptar a pelota é de 5 chances em 6 ($\approx 83\%$) de êxito. Por outro lado, se a bola estiver de posse da equipe oponente (*SoccerserverPlayerHoldingTheBall* < 0), então a possibilidade de sucesso

vai depender da capacidade de marcação de cada jogador, conforme já explicado na regra *Mark_Hold_Ball*.

No entanto, se o jogador não estiver na mesma posição do campo que a bola, o mesmo efetuará um deslocamento em direção à posição da bola. Esta situação só ocorrerá se o jogador não for o goleiro, pois o mesmo não deve deixar a região de sua área de defesa.

As regras que acionam o comportamento *Strategic_Position* tem como objetivo posicionar o jogador numa posição melhor, ou seja, mais próxima da bola. Esta idéia é ilustrada, em termos de código, pelo que está apresentado no quadro 22.

```
1 void setAction(int tA) // Try to go to a strategic position
2 {
3     if( playerLocalization[pNR] != SoccerserverBallPosition ) // The player's position is different of the ball
4     {
5         if(pNR != 1) // The player is not the keeper
6         {
7             if(playerBallLocalization [pNR] - SoccerserverBallPosition > 0)
8                 playerBallLocalization [pNR]--; // the player moves backward
9             else
10                playerBallLocalization [pNR]++; // the player moves forward
11        }
12    }
13 }
```

Quadro 21. Código da função *setAction()* das regras que acionam o comportamento reativo *Strategic_Position*.

No caso em que as regra acionam o comportamento *Mark_Opponent*, o objetivo é identificar se o oponente ou nenhum jogador está de posse da bola e permitir ao jogador em questão se aproximar do adversário ou da posição da pelota, caso o jogador em questão não seja o goleiro (vide quadro 23).

```
1 void setAction(int tA) // Try to mark opponente
2 {
3     if( playerLocalization[pNR] == SoccerserverBallPosition )
4     {
5         if( tA >=-8 ) playerHoldingTheBall [pNR] = SoccerserverPlayerHoldingTheBall;
6     }
7     else
8     {
9         if(pNR != 1)
10        {
11            if(playerBallLocalization [pNR] - SoccerserverBallPosition > 0) playerBallLocalization [pNR]--;
12            else playerBallLocalization [pNR]++;
13        }
14    }
15 }
```

Quadro 22. Código da função *setAction()* das regras que acionam o comportamento reativo *Mark_Opponent*.

Se o jogador estiver na mesma localização que a bola o mesmo terá 8 chances em 12 para identificar o jogador que a detém, caso contrário deverá

identificar aonde a bola está e deslocar-se para a posição da mesma. Esta regra não se aplica ao goleiro, pois foi convencionado que o mesmo não deve sair da região de sua área.

As regras que acionam o comportamento *Pass_Ball_Uncond* tem por objetivo permitir a um jogador realizar um passe para outro, sem se preocupar com possíveis obstáculos ou impedimentos para isto.

```
1 void setAction(int tA) // Try to pass Ball unconditionally
2 {
3     if( playerLocalization[pNR] == SoccerServerBallPosition )
4     {
5         if( tA == playerLocalization[pReceiver] )
6         {
7             if( tA > playerLocalization[pNR] )
8             {
9                 SoccerServerBallPosition = playerLocalization[pReceiver];
10                SoccerServerPlayerHoldingTheBall = pReceiver;
11            }
12            else
13            {
14                SoccerServerBallPosition = tA;
15                SoccerServerPlayerHoldingTheBall = -2*tA;
16            }
17        }
18        else
19        {
20            SoccerServerBallPosition = tA;
21            SoccerServerPlayerHoldingTheBall = -2*tA;
22        }
23    }
24 }
```

Quadro 23. Código da função *setAction()* das regras que acionam o comportamento reativo *Pass_Ball_Uncond*.

Por outro lado, as regras que acionam o comportamento *Pass_Ball_Cond* tem por objetivo permitir a um jogador realizar um passe para outro, levando em consideração possíveis obstáculos ou impedimentos para isto.

Comparando as duas regras acima se percebe que, na primeira, o jogador que irá fazer o passe executará tal ação seja qual for o resultado. Já na segunda, se o valor da variável *tA* for igual a 0, então significa que o jogador não deve executar o passe e deve continuar com a posse de bola. Em ambas as situações, como em todos os casos de tentativas de execução de ações, o fato de se tentar executar um passe não significa que o mesmo será perfeitamente executado. Nos dois códigos pode ser observado que a jogada só surtirá efeito se o jogador que receberá a bola estiver na posição do campo (*playerLocalization[pReceiver]*) igual ao valor da variável aleatória *tA* (*tA == playerLocalization[pReceiver]*).

Como as duas regras em questão fazem parte do conjunto de regras associadas aos estados cognitivos *Advance* e *Side_Attack*, considerou-se que as

mesmas deviam promover ações ofensivas na direção do gol adversário. Assim, o passe só será realizado se o valor de tA for maior do que o valor da posição atual do jogador que está efetuando o passe ($tA > playerLocalization[pNR]$), ou seja, a jogada permite que a equipe avance em vez de recuar.

```
1  bool playerPassCondition = true;
2
3  void setPassCondition(int pPC)
4  {
5      if ( pPC == 1 ) playerPassCondition = true;
6      else playerPassCondition = false;
7  }
8
9  void setAction(int tA)
10 {
11     if( playerLocalization[pNR] == SoccerServerBallPosition )
12     {
13         if( playerPassCondition == true )
14         {
15             if( tA == playerLocalization[pReceiver] )
16             {
17                 if( tA > playerLocalization[pNR] )
18                 {
19                     SoccerServerBallPosition = playerLocalization[pReceiver];
20                     SoccerServerPlayerHoldingTheBall = pReceiver;
21                 }
22                 else
23                 {
24                     SoccerServerBallPosition = tA;
25                     SoccerServerPlayerHoldingTheBall = -2*tA;
26                 }
27             }
28             else
29             {
30                 SoccerServerBallPosition = tA;
31                 SoccerServerPlayerHoldingTheBall = -2*tA;
32             }
33         }
34         else
35         {
36             if( tA == playerLocalization[pNR] )
37             {
38                 SoccerServerBallPosition = tA;
39                 SoccerServerPlayerHoldingTheBall = pNR;
40             }
41             else
42             {
43                 SoccerServerBallPosition = tA;
44                 SoccerServerBallPosition = -2*tA;
45             }
46         }
47     }
48 }
```

Quadro 24. Código da função *setAction()* das regras que acionam o comportamento reativo *Pass_Ball_Cond*.

As regras *Advance_Drive_Ball_Forward* e *Side_Attack_Drive_Ball_Forward* acionam, como o próprio nome sugere, o comportamento *Drive_Ball_Forward*. Este comportamento é definido como a capacidade do jogador conduzir a bola à frente, ou seja, conduzir a bola para o campo de ataque. Esta capacidade foi implementada, conforme apresentado no quadro 26.


```

1 void setAction(int tA) // Try to drive ball forward
2 {
3     if( playerLocalization[pNR] == SoccerserverBallPosition ){
4         if( tA + SoccerserverBallPosition > 5 ) {
5
6             SoccerserverBallPosition = 5;
7             SoccerserverPlayerHoldingTheBall = -1;
8         }
9         else{
10            if( tA + SoccerserverBallPosition == 5 ){
11                SoccerserverBallPosition = 5;
12                if( tA <= 3) SoccerserverPlayerHoldingTheBall = -tA;
13                else SoccerserverPlayerHoldingTheBall = 0;
14            }
15            else{
16                if( tA <= 1){
17                    SoccerserverBallPosition = SoccerserverBallPosition + tA;
18                    SoccerserverPlayerHoldingTheBall = pNR;
19                    playerLocalization[pNR] = SoccerserverBallPosition + tA;
20                }
21                else {
22                    SoccerserverBallPosition = SoccerserverBallPosition + tA;
23                    SoccerserverPlayerHoldingTheBall = 0;
24                }
25            }
26        }
27    }
28 }
29

```

Quadro 25. Código da função *setAction()* das regras que acionam o comportamento reativo *Drive_Ball_Forward*.

A função do quadro 26 garante que a jogada só será realizada se o jogador tiver na posição onde a bola se encontra e o resultado da ação dependerá do valor da variável *tA* e da posição do jogador. Se *tA* somado ao valor da posição do jogador for maior que 5, a bola foi conduzida além do limite do campo e será concedido o tiro de meta para equipe adversária. No caso do somatório da posição do jogador com *tA* ser igual a 5, foi estipulado que o jogador deixou a bola escapar para a área de defesa do adversário e perdeu o controle da bola. No entanto se o jogador não adentra ou ultrapassa a área adversária e o valor de *tA* é igual ou menor que 1, significa que o jogador está conduzindo a bola e tem a sua posse, caso contrário, este adiantou a bola, mas não garantiu a sua posse.

2) Implementação das funções *setAction()* dos grupos de regras que possuem comportamentos não agrupáveis entre si.

Diferentemente das regras anteriores, as demais regras existentes implicam no acionamento de comportamentos muito específicos que não puderam ser agrupadas entre si ou incorporadas aos grupos anteriores, fato este que implicou na implementação individual das mesmas.

Assim, como exemplo, a regra *Advance_Pass_Ball_Forward* se caracteriza por acionar o comportamento de chutar a bola para frente, mais conhecido no futebol humano como “chutão pra frente”, conforme apresentado no quadro 27. Esta regra difere das demais regras que executam passe, devido ao fato de não possuir um jogador da própria equipe com o objetivo de entrega do controle da bola.

Neste comportamento, se o valor de *tA* adicionado à localização do jogador que está efetuando a jogada for maior que 5, significa dizer que o chute para frente saiu pela linha de fundo e isto implica em tiro de meta para a equipe adversária. Caso contrário, a posse de bola deverá ser do adversário ou de nenhuma das equipes. Apesar das regras *Advance_Drive_Ball_Fast* e *Side_Attack_Drive_Ball_Fast* terem similaridades nas suas assinaturas, similaridade esta que pode deduzir que ambas acionam os mesmos comportamentos, as mesmas não produzem as mesmas ações para o ambiente, como pode ser constatado ao se comparar as regras propriamente ditas e seus respectivos autômatos no quadro 28.

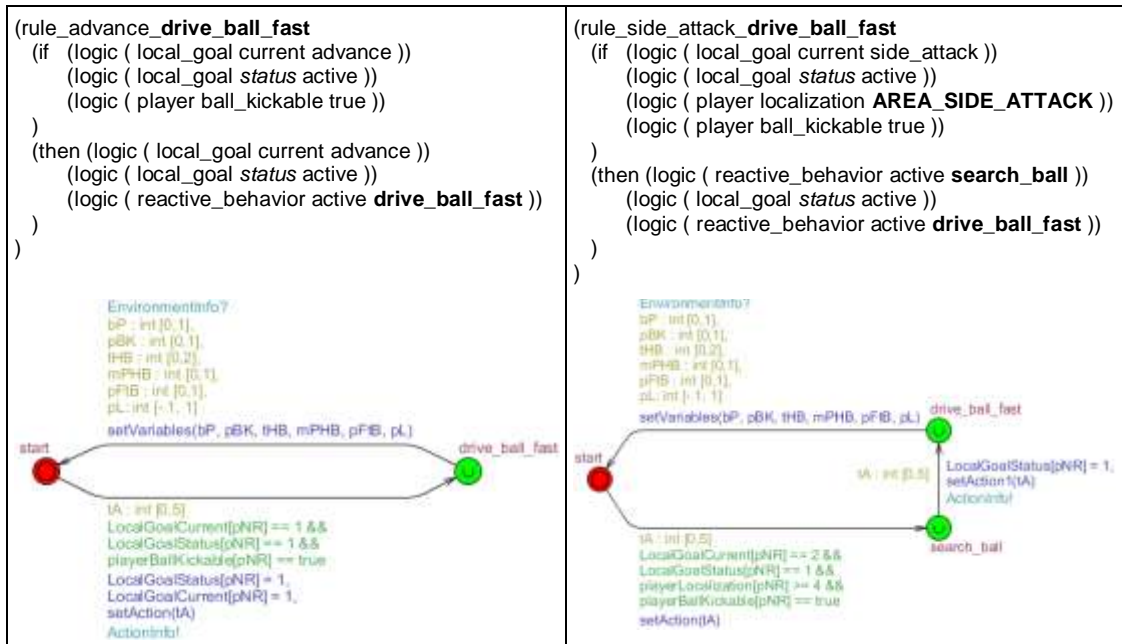
```

1 void setAction(int tA) { // Try to pass ball forward
2   if( playerLocalization[pNR] == SoccerserverBallPosition ){
3     if( tA + playerLocalization[pNR] > 5 ){
4       SoccerserverBallPosition = 5;
5       SoccerserverPlayerHoldingTheBall = -1;      }
6     else{
7       if( tA + playerLocalization[pNR] == 5 ){
8         SoccerserverBallPosition = 5;
9         if( tA <= 3) SoccerserverPlayerHoldingTheBall = -tA;
10        else SoccerserverPlayerHoldingTheBall = 0;  }
11      else{
12        SoccerserverBallPosition = tA + playerLocalization[pNR];
13        SoccerserverPlayerHoldingTheBall = 0;      }
14      }
15    }
16  }

```

Quadro 26. Código da função *setAction()* das regras que acionam o comportamento reativo *Pass_Ball_Forward*.

Observando o quadro 28, fica claro que as regras não são totalmente equivalentes e, por esse motivo, não podem ter igual tratamento. Assim, analisando a regra *Advance_Drive_Ball_Fast*, pode-se identificar uma única função que aciona um determinado comportamento para ser repassado ao ambiente, conforme apresentado no quadro 29.



Quadro 27. Comparativo das regras *Advance_Pass_Ball_Forward* e *Side_Attack_Pass_Ball_Forward* e seus respectivos autômatos.

```

1 void setAction(int tA) // Try to drive ball fast
2 {
3   if( playerLocalization[pNR] == SoccerserverBallPosition )
4   {
5     if( tA + SoccerserverBallPosition > 5 )
6     {
7       SoccerserverBallPosition = 5;
8       SoccerserverPlayerHoldingTheBall = -1;
9     }
10    else
11    {
12      if( tA + SoccerserverBallPosition == 5 )
13      {
14        SoccerserverBallPosition = 5;
15        if( tA <= 3 ) SoccerserverPlayerHoldingTheBall = -tA;
16        else SoccerserverPlayerHoldingTheBall = 0;
17      }
18    }
19    else
20    {
21      if((tA >= 1)&&(tA <= 2))
22      {
23        SoccerserverBallPosition = SoccerserverBallPosition + tA;
24        SoccerserverPlayerHoldingTheBall = pNR;
25        playerLocalization[pNR] = SoccerserverBallPosition + tA;
26      }
27    }
28    else
29    {
30      SoccerserverBallPosition = SoccerserverBallPosition + tA;
31      SoccerserverPlayerHoldingTheBall = 0;
32    }
33  }
34 }

```

Quadro 28. Código da função *setAction()* das regras que acionam o comportamento reativo *Pass_Ball_Fast* da regra *Advance_Pass_Ball_Fast*.

A regra *Advance_Drive_Ball_Fast* possui a função *setAction()* implementada de forma muito similar à função de mesmo nome das regras *Drive_Ball_Forward*. A principal diferença entre ambas é que na primeira a posse de

bola é garantida quando o deslocamento do jogador, promovido pelo valor obtido pela variável *tA*, for igual a 1 ou 2 (vide linha 20 do quadro 29), enquanto nas outras esta situação só ocorre para valores 0 ou 1 (vide linha 20 do quadro 26). Nos demais casos o deslocamento acelerado (*fast*) não obtém sucesso e as situações de perda do controle de bola são idênticas às das funções *Drive_Ball_Forward*.

A regra *Side_Attack_Strategic_Position_Free_Mark*, por sua vez, é uma variante das regras que acionam o comportamento *Strategic_Position*. A diferença entre as regras é que a que *Side_Attack_Strategic_Position_Free_Mark* aplica-se a um caso específico, quando o jogador que a está invocando encontra-se na região 3 do campo.

Por fim, tem-se as regras *Side_Attack_Kick_to_Goal* e *Ending_Kick_Ball* que acionam o comportamento *Kick_to_Goal*. A diferença entre estas é que, na presente configuração de regras, a primeira é utilizada pelos jogadores de ataque e a outra somente pelo goleiro. De um modo geral elas permitem que os jogadores possam tentar o chute a gol e, caso este seja bem sucedido, a variável *goalScored* seja mudada para o valor 1 e a partida possa ser reiniciada.

```
1 void setAction(int tA) // Try to search ball
2 {
3     if( playerLocalization[pNR] == SoccerserverBallPosition )
4     {
5         if( tA + playerLocalization[pNR] > 6 )
6         {
7             SoccerserverBallPosition = 5;
8             SoccerserverPlayerHoldingTheBall = -1;
9         }
10        else
11        {
12            if( tA + playerLocalization[pNR] == 6 ) // Goal made. Restart game
13            {
14                goalScored = 1;
15            }
16            else
17            {
18                SoccerserverBallPosition = tA + playerLocalization[pNR];
19                SoccerserverPlayerHoldingTheBall = 0;
20            }
21        }
22    }
23 }
```

Quadro 29. Código da função *setAction()* *Side_Attack_Kick_to_Goal*.

Apêndice G – Resultados da verificação de cada jogador com o ambiente

<p>P1</p> <p>E<> P1_C.Advance Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_C.Mark Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_C.Side_Attack Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_MHB.hold_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_MSB.search_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_MMO.mark_opponent Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_MIB.intercept_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_AIB.intercept_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_ASP.strategic_position Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p>	<p>E<> P1_APBFW.pass_ball_forward Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_AF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_AA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_SASB.search_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_SAIB.intercept_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_SAMO.mark_opponent Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_SAPBU7.pass_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_SAA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p>P2</p> <p>E<> P2_MHB.hold_ball Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: OKB / OKB. Property is satisfied.</p> <p>E<> P2_MSB.search_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: OKB / OKB. Property is satisfied.</p> <p>E<> P2_MMO.mark_opponent Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: OKB / OKB. Property is satisfied.</p> <p>E<> P2_MIB.intercept_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: OKB / OKB. Property is satisfied.</p> <p>E<> P2_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: OKB / OKB. Property is satisfied.</p> <p>E<> P2_APBFW.pass_ball_forward Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: OKB / OKB. Property is satisfied.</p> <p>E<> P2_APBC8.pass_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: OKB / OKB. Property is satisfied.</p>	<p>P3</p> <p>E<> P3_C.Advance Verification/kernel/elapsed time used: 0.166s / 0.001s / 0.168s. Resident/virtual memory usage peaks: 42,644KB / 2,513,888KB. Property is satisfied.</p> <p>E<> P3_C.Mark Verification/kernel/elapsed time used: 0.167s / 0.002s / 0.17s. Resident/virtual memory usage peaks: 57,376KB / 2,529,128KB. Property is satisfied.</p> <p>E<> P3_MHB.hold_ball Verification/kernel/elapsed time used: 0.228s / 0.006s / 0.233s. Resident/virtual memory usage peaks: 72,892KB / 2,545,648KB. Property is satisfied.</p> <p>E<> P3_MSB.search_ball Verification/kernel/elapsed time used: 0.176s / 0.002s / 0.179s. Resident/virtual memory usage peaks: 88,088KB / 2,561,144KB. Property is satisfied.</p> <p>E<> P3_MMO.mark_opponent Verification/kernel/elapsed time used: 0.169s / 0.002s / 0.171s. Resident/virtual memory usage peaks: 103,416KB / 2,577,660KB. Property is satisfied.</p> <p>E<> P3_MIB.intercept_ball Verification/kernel/elapsed time used: 0.226s / 0.006s / 0.232s. Resident/virtual memory usage peaks: 118,676KB / 2,593,156KB. Property is satisfied.</p> <p>E<> P3_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.235s / 0.007s / 0.241s. Resident/virtual memory usage peaks: 133,972KB / 2,609,672KB. Property is satisfied.</p> <p>E<> P3_APBFW.pass_ball_forward Verification/kernel/elapsed time used: 0.188s / 0.003s / 0.191s. Resident/virtual memory usage peaks: 149,236KB / 2,625,172KB. Property is satisfied.</p> <p>E<> P3_ASB.search_ball Verification/kernel/elapsed time used: 0.166s / 0.002s / 0.167s. Resident/virtual memory usage peaks: 164,468KB / 2,641,688KB. Property is satisfied.</p> <p>E<> P3_AIB.intercept_ball Verification/kernel/elapsed time used: 0.2s / 0.003s / 0.203s. Resident/virtual memory usage peaks: 179,760KB / 2,658,204KB. Property is satisfied.</p> <p>E<> P3_ASP.strategic_position Verification/kernel/elapsed time used: 0.164s / 0.002s / 0.167s. Resident/virtual memory usage peaks: 195,052KB / 2,673,696KB. Property is satisfied.</p> <p>E<> P3_AF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.169s / 0.002s / 0.171s. Resident/virtual memory usage peaks: 210,276KB / 2,690,212KB. Property is satisfied.</p>
<p>P4</p> <p>E<> P4_C.Advance Verification/kernel/elapsed time used: 0.225s / 0.003s / 0.227s. Resident/virtual memory usage peaks: 240,764KB / 2,728,504KB. Property is satisfied.</p> <p>E<> P4_C.Mark Verification/kernel/elapsed time used: 0.224s / 0.003s / 0.227s. Resident/virtual memory usage peaks: 267,952KB / 2,763,336KB. Property is satisfied.</p> <p>E<> P4_C.Side_Attack Verification/kernel/elapsed time used: 0.229s / 0.003s / 0.232s. Resident/virtual memory usage peaks: 287,156KB / 2,788,436KB. Property is satisfied.</p> <p>E<> P4_MHB.hold_ball Verification/kernel/elapsed time used: 0.241s / 0.003s / 0.245s. Resident/virtual memory usage peaks: 354,812KB / 2,926,532KB. Property is satisfied.</p>	<p>E<> P4_MSB.search_ball Verification/kernel/elapsed time used: 0.243s / 0.003s / 0.246s. Resident/virtual memory usage peaks: 47,980KB / 2,526,956KB. Property is satisfied.</p> <p>E<> P4_MMO.mark_opponent Verification/kernel/elapsed time used: 0.258s / 0.003s / 0.261s. Resident/virtual memory usage peaks: 67,192KB / 2,552,060KB. Property is satisfied.</p> <p>E<> P4_MIB.intercept_ball Verification/kernel/elapsed time used: 0.248s / 0.004s / 0.251s. Resident/virtual memory usage peaks: 86,352KB / 2,577,160KB. Property is satisfied.</p> <p>E<> P4_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.226s / 0.002s / 0.229s. Resident/virtual memory usage peaks: 105,556KB / 2,602,264KB. Property is satisfied.</p>

<p>E<> P4_APBFW.pass_ball_forward Verification/kernel/elapsed time used: 0.227s / 0.002s / 0.23s. Resident/virtual memory usage peaks: 124,704KB / 2,627,368KB. Property is satisfied.</p> <p>E<> P4_ASB.search_ball Verification/kernel/elapsed time used: 0.227s / 0.003s / 0.23s. Resident/virtual memory usage peaks: 143,904KB / 2,652,468KB. Property is satisfied.</p> <p>E<> P4_AIB.intercept_ball Verification/kernel/elapsed time used: 0.229s / 0.003s / 0.231s. Resident/virtual memory usage peaks: 163,044KB / 2,677,568KB. Property is satisfied.</p> <p>E<> P4_ASP.strategic_position Verification/kernel/elapsed time used: 0.225s / 0.002s / 0.228s. Resident/virtual memory usage peaks: 182,252KB / 2,701,652KB. Property is satisfied.</p> <p>E<> P4_AF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.228s / 0.003s / 0.23s. Resident/virtual memory usage peaks: 201,428KB / 2,726,752KB. Property is satisfied.</p> <p>E<> P4_AA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.227s / 0.002s / 0.229s. Resident/virtual memory usage peaks: 220,600KB / 2,751,852KB. Property is satisfied.</p> <p>E<> P4_SASB.search_ball Verification/kernel/elapsed time used: 0.239s / 0.003s / 0.242s. Resident/virtual memory usage peaks: 239,780KB / 2,776,956KB. Property is satisfied.</p> <p>E<> P4_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.242s / 0.003s / 0.246s. Resident/virtual memory usage peaks: 258,956KB / 2,802,056KB. Property is satisfied.</p>	<p>E<> P4_SAMO.mark_opponent Verification/kernel/elapsed time used: 0.237s / 0.003s / 0.241s. Resident/virtual memory usage peaks: 278,128KB / 2,827,156KB. Property is satisfied.</p> <p>E<> P4_SAPBU7.pass_ball Verification/kernel/elapsed time used: 0.242s / 0.003s / 0.245s. Resident/virtual memory usage peaks: 297,300KB / 2,852,256KB. Property is satisfied.</p> <p>E<> P4_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.238s / 0.003s / 0.241s. Resident/virtual memory usage peaks: 316,476KB / 2,877,356KB. Property is satisfied.</p> <p>E<> P4_SAA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.226s / 0.003s / 0.228s. Resident/virtual memory usage peaks: 335,640KB / 2,901,432KB. Property is satisfied.</p>
<p>P5</p> <p>E<> P5_C.Advance Verification/kernel/elapsed time used: 0.178s / 0.002s / 0.181s. Resident/virtual memory usage peaks: 232,232KB / 2,714,516KB. Property is satisfied.</p> <p>E<> P5_C.Mark Verification/kernel/elapsed time used: 0.182s / 0.002s / 0.184s. Resident/virtual memory usage peaks: 248,016KB / 2,731,424KB. Property is satisfied.</p> <p>E<> P5_MHB.hold_ball Verification/kernel/elapsed time used: 0.194s / 0.003s / 0.197s. Resident/virtual memory usage peaks: 263,796KB / 2,748,332KB. Property is satisfied.</p> <p>E<> P5_MSB.search_ball Verification/kernel/elapsed time used: 0.226s / 0.004s / 0.23s. Resident/virtual memory usage peaks: 279,588KB / 2,766,264KB. Property is satisfied.</p> <p>E<> P5_MMO.mark_opponent Verification/kernel/elapsed time used: 0.192s / 0.003s / 0.194s. Resident/virtual memory usage peaks: 295,304KB / 2,783,172KB. Property is satisfied.</p> <p>E<> P5_MIB.intercept_ball Verification/kernel/elapsed time used: 0.198s / 0.004s / 0.203s. Resident/virtual memory usage peaks: 311,140KB / 2,800,096KB. Property is satisfied.</p> <p>E<> P5_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.954s / 0.055s / 1.009s. Resident/virtual memory usage peaks: 328,148KB / 2,818,028KB. Property is satisfied.</p> <p>E<> P5_APBU6.pass_ball Verification/kernel/elapsed time used: 0.182s / 0.003s / 0.184s. Resident/virtual memory usage peaks: 342,740KB / 2,833,912KB. Property is satisfied.</p>	<p>E<> P5_APB8C.pass_ball Verification/kernel/elapsed time used: 0.182s / 0.003s / 0.185s. Resident/virtual memory usage peaks: 358,456KB / 2,850,820KB. Property is satisfied.</p> <p>E<> P5_ASB.search_ball Verification/kernel/elapsed time used: 0.182s / 0.002s / 0.184s. Resident/virtual memory usage peaks: 374,284KB / 2,868,752KB. Property is satisfied.</p> <p>E<> P5_AIB.intercept_ball Verification/kernel/elapsed time used: 0.182s / 0.003s / 0.185s. Resident/virtual memory usage peaks: 390,068KB / 2,885,660KB. Property is satisfied.</p> <p>E<> P5_ASP.strategic_position Verification/kernel/elapsed time used: 0.181s / 0.002s / 0.183s. Resident/virtual memory usage peaks: 405,848KB / 2,902,568KB. Property is satisfied.</p> <p>E<> P5_AF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.182s / 0.002s / 0.184s. Resident/virtual memory usage peaks: 421,632KB / 2,919,476KB. Property is satisfied.</p>

<p>P6</p> <p>E<> P6_C.Side_Attack Verification/kernel/elapsed time used: 0.229s / 0.004s / 0.233s. Resident/virtual memory usage peaks: 920,860KB / 3,698,560KB. Property is satisfied.</p> <p>E<> P6_C.Mark Verification/kernel/elapsed time used: 0.217s / 0.003s / 0.221s. Resident/virtual memory usage peaks: 938,916KB / 3,722,636KB. Property is satisfied.</p> <p>E<> P6_MHB.hold_ball Verification/kernel/elapsed time used: 0.238s / 0.005s / 0.243s. Resident/virtual memory usage peaks: 956,984KB / 3,747,736KB. Property is satisfied.</p> <p>E<> P6_MSB.search_ball Verification/kernel/elapsed time used: 0.225s / 0.004s / 0.23s. Resident/virtual memory usage peaks: 975,048KB / 3,771,812KB. Property is satisfied.</p> <p>E<> P6_MIB.intercept_ball Verification/kernel/elapsed time used: 0.219s / 0.004s / 0.223s. Resident/virtual memory usage peaks: 993,108KB / 3,795,888KB. Property is satisfied.</p> <p>E<> P6_MSP.strategic_position Verification/kernel/elapsed time used: 0.219s / 0.005s / 0.223s. Resident/virtual memory usage peaks: 1,011,212KB / 3,829,180KB. Property is satisfied.</p> <p>E<> P6_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.886s / 0.033s / 0.919s. Resident/virtual memory usage peaks: 1,029,992KB / 3,854,280KB. Property is satisfied.</p>	<p>E<> P6_SASB.search_ball Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0.287s / 0.007s / 0.294s. Resident/virtual memory usage peaks: 27,220KB / 2,501,344KB. Property is satisfied.</p> <p>E<> P6_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.3s / 0.007s / 0.309s. Resident/virtual memory usage peaks: 45,920KB / 2,525,932KB. Property is satisfied.</p> <p>E<> P6_SAPBC8.pass_ball Verification/kernel/elapsed time used: 0.263s / 0.005s / 0.269s. Resident/virtual memory usage peaks: 63,956KB / 2,551,032KB. Property is satisfied.</p> <p>E<> P6_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.265s / 0.005s / 0.27s. Resident/virtual memory usage peaks: 82,020KB / 2,575,112KB. Property is satisfied.</p> <p>E<> P6_SAPBC11.pass_ball Verification/kernel/elapsed time used: 0.268s / 0.005s / 0.273s. Resident/virtual memory usage peaks: 100,084KB / 2,599,192KB. Property is satisfied.</p> <p>E<> P6_SAF.Set_Cognitive_Failed Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0.217s / 0.003s / 0.22s. Resident/virtual memory usage peaks: 26,996KB / 2,501,344KB. Property is satisfied.</p>
<p>P7</p> <p>E<> P7_C.Side_Attack Verification/kernel/elapsed time used: 0.208s / 0.004s / 0.212s. Resident/virtual memory usage peaks: 309,212KB / 2,851,232KB. Property is satisfied.</p> <p>E<> P7_C.Mark Verification/kernel/elapsed time used: 0.212s / 0.005s / 0.216s. Resident/virtual memory usage peaks: 319,988KB / 2,868,140KB. Property is satisfied.</p> <p>E<> P7_MHB.hold_ball Verification/kernel/elapsed time used: 0.214s / 0.005s / 0.22s. Resident/virtual memory usage peaks: 338,052KB / 2,892,216KB. Property is satisfied.</p> <p>E<> P7_MSB.search_ball Verification/kernel/elapsed time used: 0.215s / 0.005s / 0.22s. Resident/virtual memory usage peaks: 356,116KB / 2,917,316KB. Property is satisfied.</p> <p>E<> P7_MIB.intercept_ball Verification/kernel/elapsed time used: 0.214s / 0.005s / 0.219s. Resident/virtual memory usage peaks: 374,192KB / 2,941,408KB. Property is satisfied.</p> <p>E<> P7_MSP.strategic_position Verification/kernel/elapsed time used: 0.211s / 0.005s / 0.216s. Resident/virtual memory usage peaks: 392,320KB / 2,965,484KB. Property is satisfied.</p>	<p>E<> P7_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.87s / 0.033s / 0.903s. Resident/virtual memory usage peaks: 410,976KB / 2,990,584KB. Property is satisfied.</p> <p>E<> P7_SASB.search_ball Verification/kernel/elapsed time used: 0.286s / 0.009s / 0.295s. Resident/virtual memory usage peaks: 428,504KB / 3,014,660KB. Property is satisfied.</p> <p>E<> P7_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.287s / 0.009s / 0.295s. Resident/virtual memory usage peaks: 446,528KB / 3,038,736KB. Property is satisfied.</p> <p>E<> P7_SAPBC8.pass_ball Verification/kernel/elapsed time used: 0.257s / 0.008s / 0.264s. Resident/virtual memory usage peaks: 464,596KB / 3,063,836KB. Property is satisfied.</p> <p>E<> P7_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.257s / 0.007s / 0.264s. Resident/virtual memory usage peaks: 482,636KB / 3,087,920KB. Property is satisfied.</p> <p>E<> P7_SAPBC10.pass_ball Verification/kernel/elapsed time used: 0.258s / 0.007s / 0.265s. Resident/virtual memory usage peaks: 500,724KB / 3,111,996KB. Property is satisfied.</p> <p>E<> P7_SASPFM.free_mark Verification/kernel/elapsed time used: 49.005s / 0.077s / 49.083s. Resident/virtual memory usage peaks: 549,240KB / 3,166,792KB. Property is satisfied.</p> <p>E<> P7_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.211s / 0.004s / 0.214s. Resident/virtual memory usage peaks: 555,528KB / 3,186,772KB. Property is satisfied.</p>

<p>P8</p> <p>E<> P8_C.Side_Attack Verification/kernel/elapsed time used: 0.139s / 0.002s / 0.141s. Resident/virtual memory usage peaks: 32,296KB / 2,503,516KB. Property is satisfied.</p> <p>E<> P8_C.Mark Verification/kernel/elapsed time used: 0.138s / 0.002s / 0.14s. Resident/virtual memory usage peaks: 45,224KB / 2,519,008KB. Property is satisfied.</p> <p>E<> P8_MHB.hold_ball Verification/kernel/elapsed time used: 0.143s / 0.002s / 0.145s. Resident/virtual memory usage peaks: 58,160KB / 2,533,476KB. Property is satisfied.</p> <p>E<> P8_MSB.search_ball Verification/kernel/elapsed time used: 0.138s / 0.002s / 0.14s. Resident/virtual memory usage peaks: 71,096KB / 2,548,976KB. Property is satisfied.</p> <p>E<> P8_MIB.intercept_ball Verification/kernel/elapsed time used: 0.159s / 0.002s / 0.162s. Resident/virtual memory usage peaks: 84,020KB / 2,563,444KB. Property is satisfied.</p> <p>E<> P8_MSP.strategic_position Verification/kernel/elapsed time used: 0.135s / 0.002s / 0.137s. Resident/virtual memory usage peaks: 96,944KB / 2,577,912KB. Property is satisfied.</p> <p>E<> P8_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.408s / 0.013s / 0.422s. Resident/virtual memory usage peaks: 110,288KB / 2,593,404KB. Property is satisfied.</p> <p>E<> P8_SASB.search_ball Verification/kernel/elapsed time used: 0.198s / 0.006s / 0.204s. Resident/virtual memory usage peaks: 123,116KB / 2,607,884KB. Property is satisfied.</p>	<p>E<> P8_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.207s / 0.006s / 0.213s. Resident/virtual memory usage peaks: 136,044KB / 2,623,376KB. Property is satisfied.</p> <p>E<> P8_SAPBC9.pass_ball Disconnected.</p> <p>E<> P8_SAPBC10.pass_ball Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Disconnected.</p> <p>E<> P8_SADBFW.drive_ball_forward Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Disconnected.</p> <p>E<> P8_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.261s / 0.004s / 0.266s. Resident/virtual memory usage peaks: 35,368KB / 2,511,588KB. Property is satisfied.</p> <p>E<> P8_SAPBC10.pass_ball Verification/kernel/elapsed time used: 0.269s / 0.005s / 0.274s. Resident/virtual memory usage peaks: 61,112KB / 2,545,396KB. Property is satisfied.</p> <p>E<> P8_SAPBC11.pass_ball Verification/kernel/elapsed time used: 0.264s / 0.004s / 0.269s. Resident/virtual memory usage peaks: 79,244KB / 2,569,476KB. Property is satisfied.</p> <p>E<> P8_SADBFW.drive_ball_forward Verification/kernel/elapsed time used: 0.287s / 0.007s / 0.294s. Resident/virtual memory usage peaks: 97,476KB / 2,594,580KB. Property is satisfied.</p> <p>E<> P8_SASPFM.free_mark Verification/kernel/elapsed time used: 0.289s / 0.007s / 0.296s. Resident/virtual memory usage peaks: 115,604KB / 2,618,404KB. Property is satisfied.</p> <p>E<> P8_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.232s / 0.003s / 0.235s. Resident/virtual memory usage peaks: 133,688KB / 2,643,404KB. Property is satisfied.</p>
<p>P9</p> <p>E<> P9_C.Side_Attack Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0.239s / 0.004s / 0.242s. Resident/virtual memory usage peaks: 28,092KB / 2,501,344KB. Property is satisfied.</p> <p>E<> P9_C.Mark Verification/kernel/elapsed time used: 0.232s / 0.003s / 0.236s. Resident/virtual memory usage peaks: 47,932KB / 2,526,956KB. Property is satisfied.</p> <p>E<> P9_MHB.hold_ball Verification/kernel/elapsed time used: 0.26s / 0.005s / 0.265s. Resident/virtual memory usage peaks: 67,280KB / 2,552,060KB. Property is satisfied.</p> <p>E<> P9_MSB.search_ball Verification/kernel/elapsed time used: 0.264s / 0.005s / 0.269s. Resident/virtual memory usage peaks: 86,480KB / 2,577,160KB. Property is satisfied.</p> <p>E<> P9_MIB.intercept_ball Verification/kernel/elapsed time used: 0.259s / 0.005s / 0.264s. Resident/virtual memory usage peaks: 105,620KB / 2,602,264KB. Property is satisfied.</p> <p>E<> P9_MSP.strategic_position Verification/kernel/elapsed time used: 0.227s / 0.004s / 0.231s. Resident/virtual memory usage peaks: 124,864KB / 2,627,368KB. Property is satisfied.</p> <p>E<> P9_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.676s / 0.03s / 0.706s. Resident/virtual memory usage peaks: 145,632KB / 2,652,468KB. Property is satisfied.</p>	<p>E<> P9_SASB.search_ball Verification/kernel/elapsed time used: 0.244s / 0.004s / 0.249s. Resident/virtual memory usage peaks: 164,188KB / 2,677,568KB. Property is satisfied.</p> <p>E<> P9_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.238s / 0.004s / 0.243s. Resident/virtual memory usage peaks: 183,424KB / 2,701,652KB. Property is satisfied.</p> <p>E<> P9_SAPBC10.pass_ball Verification/kernel/elapsed time used: 0.233s / 0.004s / 0.237s. Resident/virtual memory usage peaks: 202,628KB / 2,726,752KB. Property is satisfied.</p> <p>E<> P9_SAPBC11.pass_ball Verification/kernel/elapsed time used: 0.244s / 0.004s / 0.248s. Resident/virtual memory usage peaks: 221,828KB / 2,751,852KB. Property is satisfied.</p> <p>E<> P9_SADBFW.drive_ball_forward Verification/kernel/elapsed time used: 0.262s / 0.005s / 0.266s. Resident/virtual memory usage peaks: 240,968KB / 2,776,956KB. Property is satisfied.</p> <p>E<> P9_SAKTG.kick_to_goal Verification/kernel/elapsed time used: 0.429s / 0.014s / 0.443s. Resident/virtual memory usage peaks: 260,452KB / 2,802,056KB. Property is satisfied.</p> <p>E<> P9_SASPFM.free_mark Verification/kernel/elapsed time used: 0.233s / 0.003s / 0.236s. Resident/virtual memory usage peaks: 279,404KB / 2,827,156KB. Property is satisfied.</p> <p>E<> P9_SASP.strategic_position Verification/kernel/elapsed time used: 0.234s / 0.003s / 0.237s. Resident/virtual memory usage peaks: 298,536KB / 2,852,256KB. Property is satisfied.</p> <p>E<> P9_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.233s / 0.004s / 0.236s. Resident/virtual memory usage peaks: 317,772KB / 2,877,356KB. Property is satisfied.</p>

<p>P10</p> <p>E<> P10_C.Side_Attack Verification/kernel/elapsed time used: 0.233s / 0.003s / 0.237s. Resident/virtual memory usage peaks: 350,808KB / 2,920,896KB. Property is satisfied.</p> <p>E<> P10_C.Mark Verification/kernel/elapsed time used: 0.219s / 0.003s / 0.222s. Resident/virtual memory usage peaks: 368,808KB / 2,944,988KB. Property is satisfied.</p> <p>E<> P10_MHB.hold_ball Verification/kernel/elapsed time used: 0.228s / 0.004s / 0.232s. Resident/virtual memory usage peaks: 386,800KB / 2,969,064KB. Property is satisfied.</p> <p>E<> P10_MSB.search_ball Verification/kernel/elapsed time used: 0.237s / 0.006s / 0.244s. Resident/virtual memory usage peaks: 404,856KB / 2,994,164KB. Property is satisfied.</p> <p>E<> P10_MIB.intercept_ball Verification/kernel/elapsed time used: 0.227s / 0.005s / 0.232s. Resident/virtual memory usage peaks: 422,808KB / 3,018,240KB. Property is satisfied.</p> <p>E<> P10_MSP.strategic_position Verification/kernel/elapsed time used: 0.225s / 0.004s / 0.229s. Resident/virtual memory usage peaks: 440,800KB / 3,042,316KB. Property is satisfied.</p> <p>E<> P10_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.438s / 0.017s / 0.456s. Resident/virtual memory usage peaks: 459,028KB / 3,067,424KB. Property is satisfied.</p> <p>E<> P10_SASB.search_ball Verification/kernel/elapsed time used: 0.224s / 0.004s / 0.229s. Resident/virtual memory usage peaks: 476,780KB / 3,091,500KB. Property is satisfied.</p>	<p>E<> P10_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.22s / 0.004s / 0.224s. Resident/virtual memory usage peaks: 494,772KB / 3,115,576KB. Property is satisfied.</p> <p>E<> P10_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.221s / 0.004s / 0.225s. Resident/virtual memory usage peaks: 512,800KB / 3,147,844KB. Property is satisfied.</p> <p>E<> P10_SADBFW.drive_ball_forward Verification/kernel/elapsed time used: 0.223s / 0.004s / 0.227s. Resident/virtual memory usage peaks: 530,820KB / 3,172,944KB. Property is satisfied.</p> <p>E<> P10_SAKTG.kick_to_goal Verification/kernel/elapsed time used: 0.319s / 0.01s / 0.329s. Resident/virtual memory usage peaks: 548,884KB / 3,197,020KB. Property is satisfied.</p> <p>E<> P10_SASP.strategic_position Verification/kernel/elapsed time used: 0.214s / 0.004s / 0.218s. Resident/virtual memory usage peaks: 566,800KB / 3,221,096KB. Property is satisfied.</p> <p>E<> P10_SADBFS.drive_ball_fast Verification/kernel/elapsed time used: 0.449s / 0.019s / 0.468s. Resident/virtual memory usage peaks: 585,008KB / 3,246,196KB. Property is satisfied.</p> <p>E<> P10_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.213s / 0.004s / 0.216s. Resident/virtual memory usage peaks: 602,740KB / 3,270,272KB. Property is satisfied.</p>
<p>P11</p> <p>E<> P11_C.Side_Attack Verification/kernel/elapsed time used: 0.211s / 0.004s / 0.214s. Resident/virtual memory usage peaks: 635,752KB / 3,313,812KB. Property is satisfied.</p> <p>E<> P11_C.Mark Verification/kernel/elapsed time used: 0.251s / 0.005s / 0.256s. Resident/virtual memory usage peaks: 653,800KB / 3,337,888KB. Property is satisfied.</p> <p>E<> P11_MHB.hold_ball Verification/kernel/elapsed time used: 0.238s / 0.005s / 0.243s. Resident/virtual memory usage peaks: 671,752KB / 3,361,964KB. Property is satisfied.</p> <p>E<> P11_MSB.search_ball Verification/kernel/elapsed time used: 0.238s / 0.004s / 0.243s. Resident/virtual memory usage peaks: 689,780KB / 3,387,064KB. Property is satisfied.</p> <p>E<> P11_MIB.intercept_ball Verification/kernel/elapsed time used: 0.228s / 0.004s / 0.232s. Resident/virtual memory usage peaks: 707,764KB / 3,411,172KB. Property is satisfied.</p> <p>E<> P11_MSP.strategic_position Verification/kernel/elapsed time used: 0.22s / 0.003s / 0.223s. Resident/virtual memory usage peaks: 725,752KB / 3,435,248KB. Property is satisfied.</p> <p>E<> P11_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.414s / 0.016s / 0.43s. Resident/virtual memory usage peaks: 743,976KB / 3,460,348KB. Property is satisfied.</p> <p>E<> P11_SASB.search_ball Verification/kernel/elapsed time used: 0.228s / 0.004s / 0.232s. Resident/virtual memory usage peaks: 761,796KB / 3,484,424KB. Property is satisfied.</p>	<p>E<> P11_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.217s / 0.004s / 0.22s. Resident/virtual memory usage peaks: 779,752KB / 3,508,500KB. Property is satisfied.</p> <p>E<> P11_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.229s / 0.005s / 0.233s. Resident/virtual memory usage peaks: 797,736KB / 3,532,576KB. Property is satisfied.</p> <p>E<> P11_SADBFW.drive_ball_forward Verification/kernel/elapsed time used: 0.227s / 0.004s / 0.231s. Resident/virtual memory usage peaks: 815,728KB / 3,557,676KB. Property is satisfied.</p> <p>E<> P11_SAKTG.kick_to_goal Verification/kernel/elapsed time used: 0.334s / 0.01s / 0.345s. Resident/virtual memory usage peaks: 833,892KB / 3,581,752KB. Property is satisfied.</p> <p>E<> P11_SASP.strategic_position Verification/kernel/elapsed time used: 0.214s / 0.004s / 0.217s. Resident/virtual memory usage peaks: 851,736KB / 3,605,828KB. Property is satisfied.</p> <p>E<> P11_SADBFS.drive_ball_fast Verification/kernel/elapsed time used: 0.431s / 0.017s / 0.449s. Resident/virtual memory usage peaks: 870,008KB / 3,630,928KB. Property is satisfied.</p> <p>E<> P11_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.214s / 0.003s / 0.218s. Resident/virtual memory usage peaks: 887,716KB / 3,655,004KB. Property is satisfied.</p>

Apêndice H – Resultados da verificação da equipe com o ambiente

<p>P1</p> <p>E<> P1_C.Advance Verification/kernel/elapsed time used: 2.089s / 0.023s / 2.112s. Resident/virtual memory usage peaks: 324,260KB / 2,859,836KB. Property is satisfied.</p> <p>E<> P1_C.Mark Verification/kernel/elapsed time used: 2.089s / 0.023s / 2.112s. Resident/virtual memory usage peaks: 484,252KB / 3,055,192KB. Property is satisfied.</p> <p>E<> P1_MHB.hold_ball Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_MSB.search_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_MMO.mark_opponent Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_MIB.intercept_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_AIB.intercept_ball Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_ASP.strategic_position Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_APBFW.pass_ball_forward Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P1_AF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p>	<p>P2</p> <p>E<> P2_MHB.hold_ball Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P2_MSB.search_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P2_MMO.mark_opponent Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P2_MIB.intercept_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P2_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P2_APBFW.pass_ball_forward Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p> <p>E<> P2_APBCB.pass_ball Verification/kernel/elapsed time used: 0s / 0s / 0s. Resident/virtual memory usage peaks: 0KB / 0KB. Property is satisfied.</p>
<p>P3</p> <p>E<> P3_C.Advance Verification/kernel/elapsed time used: 0.166s / 0.001s / 0.168s. Resident/virtual memory usage peaks: 42,644KB / 2,513,888KB. Property is satisfied.</p> <p>E<> P3_C.Mark Verification/kernel/elapsed time used: 0.167s / 0.002s / 0.17s. Resident/virtual memory usage peaks: 57,376KB / 2,529,128KB. Property is satisfied.</p> <p>E<> P3_MHB.hold_ball Verification/kernel/elapsed time used: 0.228s / 0.006s / 0.233s. Resident/virtual memory usage peaks: 72,892KB / 2,545,648KB. Property is satisfied.</p> <p>E<> P3_MSB.search_ball Verification/kernel/elapsed time used: 0.176s / 0.002s / 0.179s. Resident/virtual memory usage peaks: 88,088KB / 2,561,144KB. Property is satisfied.</p> <p>E<> P3_MMO.mark_opponent Verification/kernel/elapsed time used: 0.169s / 0.002s / 0.171s. Resident/virtual memory usage peaks: 103,416KB / 2,577,660KB. Property is satisfied.</p> <p>E<> P3_MIB.intercept_ball Verification/kernel/elapsed time used: 0.226s / 0.006s / 0.232s. Resident/virtual memory usage peaks: 118,676KB / 2,593,156KB. Property is satisfied.</p>	<p>P4</p> <p>E<> P4_C.Advance Verification/kernel/elapsed time used: 0.225s / 0.003s / 0.227s. Resident/virtual memory usage peaks: 240,764KB / 2,728,504KB. Property is satisfied.</p> <p>E<> P4_C.Mark Verification/kernel/elapsed time used: 0.224s / 0.003s / 0.227s. Resident/virtual memory usage peaks: 267,952KB / 2,763,336KB. Property is satisfied.</p> <p>E<> P4_C.Side_Attack Verification/kernel/elapsed time used: 0.229s / 0.003s / 0.232s. Resident/virtual memory usage peaks: 287,156KB / 2,788,436KB. Property is satisfied.</p> <p>E<> P4_MHB.hold_ball Verification/kernel/elapsed time used: 0.241s / 0.003s / 0.245s. Resident/virtual memory usage peaks: 354,812KB / 2,926,532KB. Property is satisfied.</p> <p>E<> P4_MSB.search_ball Verification/kernel/elapsed time used: 0.243s / 0.003s / 0.246s. Resident/virtual memory usage peaks: 47,980KB / 2,526,956KB. Property is satisfied.</p> <p>E<> P4_MMO.mark_opponent Verification/kernel/elapsed time used: 0.258s / 0.003s / 0.261s. Resident/virtual memory usage peaks: 67,192KB / 2,552,060KB. Property is satisfied.</p> <p>E<> P4_MIB.intercept_ball Verification/kernel/elapsed time used: 0.248s / 0.004s / 0.251s. Resident/virtual memory usage peaks: 86,352KB / 2,577,160KB. Property is satisfied.</p> <p>E<> P4_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.226s / 0.002s / 0.229s. Resident/virtual memory usage peaks: 105,556KB / 2,602,264KB. Property is satisfied.</p>

<p>E<> P3_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.235s / 0.007s / 0.241s. Resident/virtual memory usage peaks: 133,972KB / 2,609,672KB. Property is satisfied.</p> <p>E<> P3_APBFW.pass_ball_forward Verification/kernel/elapsed time used: 0.188s / 0.003s / 0.191s. Resident/virtual memory usage peaks: 149,236KB / 2,625,172KB. Property is satisfied.</p> <p>E<> P3_ASB.search_ball Verification/kernel/elapsed time used: 0.166s / 0.002s / 0.167s. Resident/virtual memory usage peaks: 164,468KB / 2,641,688KB. Property is satisfied.</p> <p>E<> P3_AIB.intercept_ball Verification/kernel/elapsed time used: 0.2s / 0.003s / 0.203s. Resident/virtual memory usage peaks: 179,760KB / 2,658,204KB. Property is satisfied.</p> <p>E<> P3_ASP.strategic_position Verification/kernel/elapsed time used: 0.164s / 0.002s / 0.167s. Resident/virtual memory usage peaks: 195,052KB / 2,673,696KB. Property is satisfied.</p> <p>E<> P3_AF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.169s / 0.002s / 0.171s. Resident/virtual memory usage peaks: 210,276KB / 2,690,212KB. Property is satisfied.</p>	<p>E<> P4_APBFW.pass_ball_forward Verification/kernel/elapsed time used: 0.227s / 0.002s / 0.23s. Resident/virtual memory usage peaks: 124,704KB / 2,627,368KB. Property is satisfied.</p> <p>E<> P4_ASB.search_ball Verification/kernel/elapsed time used: 0.227s / 0.003s / 0.23s. Resident/virtual memory usage peaks: 143,904KB / 2,652,468KB. Property is satisfied.</p> <p>E<> P4_AIB.intercept_ball Verification/kernel/elapsed time used: 0.229s / 0.003s / 0.231s. Resident/virtual memory usage peaks: 163,044KB / 2,677,568KB. Property is satisfied.</p> <p>E<> P4_ASP.strategic_position Verification/kernel/elapsed time used: 0.225s / 0.002s / 0.228s. Resident/virtual memory usage peaks: 182,252KB / 2,701,652KB. Property is satisfied.</p> <p>E<> P4_AF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.228s / 0.003s / 0.23s. Resident/virtual memory usage peaks: 201,428KB / 2,726,752KB. Property is satisfied.</p> <p>E<> P4_AA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.227s / 0.002s / 0.229s. Resident/virtual memory usage peaks: 220,600KB / 2,751,852KB. Property is satisfied.</p> <p>E<> P4_SASB.search_ball Verification/kernel/elapsed time used: 0.239s / 0.003s / 0.242s. Resident/virtual memory usage peaks: 239,780KB / 2,776,956KB. Property is satisfied.</p> <p>E<> P4_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.242s / 0.003s / 0.246s. Resident/virtual memory usage peaks: 258,956KB / 2,802,056KB. Property is satisfied.</p> <p>E<> P4_SAMO.mark_opponent Verification/kernel/elapsed time used: 0.237s / 0.003s / 0.241s. Resident/virtual memory usage peaks: 278,128KB / 2,827,156KB. Property is satisfied.</p> <p>E<> P4_SAPBU7.pass_ball Verification/kernel/elapsed time used: 0.242s / 0.003s / 0.245s. Resident/virtual memory usage peaks: 297,300KB / 2,852,256KB. Property is satisfied.</p> <p>E<> P4_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.238s / 0.003s / 0.241s. Resident/virtual memory usage peaks: 316,476KB / 2,877,356KB. Property is satisfied.</p> <p>E<> P4_SAA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.226s / 0.003s / 0.228s. Resident/virtual memory usage peaks: 335,640KB / 2,901,432KB. Property is satisfied.</p>
<p>P5</p> <p>E<> P5_C.Advance Verification/kernel/elapsed time used: 0.178s / 0.002s / 0.181s. Resident/virtual memory usage peaks: 232,232KB / 2,714,516KB. Property is satisfied.</p> <p>E<> P5_C.Mark Verification/kernel/elapsed time used: 0.182s / 0.002s / 0.184s. Resident/virtual memory usage peaks: 248,016KB / 2,731,424KB. Property is satisfied.</p> <p>E<> P5_MHB.hold_ball Verification/kernel/elapsed time used: 0.194s / 0.003s / 0.197s. Resident/virtual memory usage peaks: 263,796KB / 2,748,332KB. Property is satisfied.</p> <p>E<> P5_MSB.search_ball Verification/kernel/elapsed time used: 0.226s / 0.004s / 0.23s. Resident/virtual memory usage peaks: 279,588KB / 2,766,264KB. Property is satisfied.</p> <p>E<> P5_MMO.mark_opponent Verification/kernel/elapsed time used: 0.192s / 0.003s / 0.194s. Resident/virtual memory usage peaks: 295,304KB / 2,783,172KB. Property is satisfied.</p> <p>E<> P5_MIB.intercept_ball Verification/kernel/elapsed time used: 0.198s / 0.004s / 0.203s. Resident/virtual memory usage peaks: 311,140KB / 2,800,096KB. Property is satisfied.</p> <p>E<> P5_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.954s / 0.055s / 1.009s. Resident/virtual memory usage peaks: 328,148KB / 2,818,028KB. Property is satisfied.</p> <p>E<> P5_APBU6.pass_ball Verification/kernel/elapsed time used: 0.182s / 0.003s / 0.184s. Resident/virtual memory usage peaks: 342,740KB / 2,833,912KB. Property is satisfied.</p>	<p>P6</p> <p>E<> P6_C.Side_Attack Verification/kernel/elapsed time used: 0.229s / 0.004s / 0.233s. Resident/virtual memory usage peaks: 920,860KB / 3,698,560KB. Property is satisfied.</p> <p>E<> P6_C.Mark Verification/kernel/elapsed time used: 0.217s / 0.003s / 0.221s. Resident/virtual memory usage peaks: 938,916KB / 3,722,636KB. Property is satisfied.</p> <p>E<> P6_MHB.hold_ball Verification/kernel/elapsed time used: 0.238s / 0.005s / 0.243s. Resident/virtual memory usage peaks: 956,984KB / 3,747,736KB. Property is satisfied.</p> <p>E<> P6_MSB.search_ball Verification/kernel/elapsed time used: 0.225s / 0.004s / 0.23s. Resident/virtual memory usage peaks: 975,048KB / 3,771,812KB. Property is satisfied.</p> <p>E<> P6_MIB.intercept_ball Verification/kernel/elapsed time used: 0.219s / 0.004s / 0.223s. Resident/virtual memory usage peaks: 993,108KB / 3,795,888KB. Property is satisfied.</p> <p>E<> P6_MSP.strategic_position Verification/kernel/elapsed time used: 0.219s / 0.005s / 0.223s. Resident/virtual memory usage peaks: 1,011,212KB / 3,829,180KB. Property is satisfied.</p> <p>E<> P6_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.886s / 0.033s / 0.919s. Resident/virtual memory usage peaks: 1,029,992KB / 3,854,280KB. Property is satisfied.</p>

<p>E<> P5_APB8.pass_ball Verification/kernel/elapsed time used: 0.182s / 0.003s / 0.185s. Resident/virtual memory usage peaks: 358,456KB / 2,850,820KB. Property is satisfied.</p> <p>E<> P5_AS8.search_ball Verification/kernel/elapsed time used: 0.182s / 0.002s / 0.184s. Resident/virtual memory usage peaks: 374,284KB / 2,868,752KB. Property is satisfied.</p> <p>E<> P5_AIB.intercept_ball Verification/kernel/elapsed time used: 0.182s / 0.003s / 0.185s. Resident/virtual memory usage peaks: 390,068KB / 2,885,660KB. Property is satisfied.</p> <p>E<> P5_ASP.strategic_position Verification/kernel/elapsed time used: 0.181s / 0.002s / 0.183s. Resident/virtual memory usage peaks: 405,848KB / 2,902,568KB. Property is satisfied.</p> <p>E<> P5_AF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.182s / 0.002s / 0.184s. Resident/virtual memory usage peaks: 421,632KB / 2,919,476KB. Property is satisfied.</p>	<p>E<> P6_SAS8.search_ball Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0.287s / 0.007s / 0.294s. Resident/virtual memory usage peaks: 27,220KB / 2,501,344KB. Property is satisfied.</p> <p>E<> P6_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.3s / 0.007s / 0.309s. Resident/virtual memory usage peaks: 45,920KB / 2,525,932KB. Property is satisfied.</p> <p>E<> P6_SAP88.pass_ball Verification/kernel/elapsed time used: 0.263s / 0.005s / 0.269s. Resident/virtual memory usage peaks: 63,956KB / 2,551,032KB. Property is satisfied.</p> <p>E<> P6_SAP89.pass_ball Verification/kernel/elapsed time used: 0.265s / 0.005s / 0.27s. Resident/virtual memory usage peaks: 82,020KB / 2,575,112KB. Property is satisfied.</p> <p>E<> P6_SAP8C11.pass_ball Verification/kernel/elapsed time used: 0.268s / 0.005s / 0.273s. Resident/virtual memory usage peaks: 100,084KB / 2,599,192KB. Property is satisfied.</p> <p>E<> P6_SAF.Set_Cognitive_Failed Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0.217s / 0.003s / 0.221s. Resident/virtual memory usage peaks: 26,996KB / 2,501,344KB. Property is satisfied.</p>
<p>P7</p> <p>E<> P7_C.Side_Attack Verification/kernel/elapsed time used: 0.208s / 0.004s / 0.212s. Resident/virtual memory usage peaks: 309,212KB / 2,851,232KB. Property is satisfied.</p> <p>E<> P7_C.Mark Verification/kernel/elapsed time used: 0.212s / 0.005s / 0.216s. Resident/virtual memory usage peaks: 319,988KB / 2,868,140KB. Property is satisfied.</p> <p>E<> P7_MHB.hold_ball Verification/kernel/elapsed time used: 0.214s / 0.005s / 0.22s. Resident/virtual memory usage peaks: 338,052KB / 2,892,216KB. Property is satisfied.</p> <p>E<> P7_MSB.search_ball Verification/kernel/elapsed time used: 0.215s / 0.005s / 0.22s. Resident/virtual memory usage peaks: 356,116KB / 2,917,316KB. Property is satisfied.</p> <p>E<> P7_MIB.intercept_ball Verification/kernel/elapsed time used: 0.214s / 0.005s / 0.219s. Resident/virtual memory usage peaks: 374,192KB / 2,941,408KB. Property is satisfied.</p> <p>E<> P7_MSP.strategic_position Verification/kernel/elapsed time used: 0.211s / 0.005s / 0.216s. Resident/virtual memory usage peaks: 392,320KB / 2,965,484KB. Property is satisfied.</p>	<p>P8</p> <p>E<> P8_C.Side_Attack Verification/kernel/elapsed time used: 0.139s / 0.002s / 0.141s. Resident/virtual memory usage peaks: 32,296KB / 2,503,516KB. Property is satisfied.</p> <p>E<> P8_C.Mark Verification/kernel/elapsed time used: 0.138s / 0.002s / 0.14s. Resident/virtual memory usage peaks: 45,224KB / 2,519,008KB. Property is satisfied.</p> <p>E<> P8_MHB.hold_ball Verification/kernel/elapsed time used: 0.143s / 0.002s / 0.145s. Resident/virtual memory usage peaks: 58,160KB / 2,533,476KB. Property is satisfied.</p> <p>E<> P8_MSB.search_ball Verification/kernel/elapsed time used: 0.138s / 0.002s / 0.14s. Resident/virtual memory usage peaks: 71,096KB / 2,548,976KB. Property is satisfied.</p> <p>E<> P8_MIB.intercept_ball Verification/kernel/elapsed time used: 0.159s / 0.002s / 0.162s. Resident/virtual memory usage peaks: 84,020KB / 2,563,444KB. Property is satisfied.</p> <p>E<> P8_MSP.strategic_position Verification/kernel/elapsed time used: 0.135s / 0.002s / 0.137s. Resident/virtual memory usage peaks: 96,944KB / 2,577,912KB. Property is satisfied.</p> <p>E<> P8_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.408s / 0.013s / 0.422s. Resident/virtual memory usage peaks: 110,288KB / 2,593,404KB. Property is satisfied.</p> <p>E<> P8_SAS8.search_ball Verification/kernel/elapsed time used: 0.198s / 0.006s / 0.204s. Resident/virtual memory usage peaks: 123,116KB / 2,607,884KB. Property is satisfied.</p>

<p>E<> P7_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.87s / 0.033s / 0.903s. Resident/virtual memory usage peaks: 410,976KB / 2,990,584KB. Property is satisfied.</p> <p>E<> P7_SASB.search_ball Verification/kernel/elapsed time used: 0.286s / 0.009s / 0.295s. Resident/virtual memory usage peaks: 428,504KB / 3,014,660KB. Property is satisfied.</p> <p>E<> P7_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.287s / 0.009s / 0.295s. Resident/virtual memory usage peaks: 446,528KB / 3,038,736KB. Property is satisfied.</p> <p>E<> P7_SAPBC8.pass_ball Verification/kernel/elapsed time used: 0.257s / 0.008s / 0.264s. Resident/virtual memory usage peaks: 464,596KB / 3,063,836KB. Property is satisfied.</p> <p>E<> P7_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.257s / 0.007s / 0.264s. Resident/virtual memory usage peaks: 482,636KB / 3,087,920KB. Property is satisfied.</p> <p>E<> P7_SAPBC10.pass_ball Verification/kernel/elapsed time used: 0.258s / 0.007s / 0.265s. Resident/virtual memory usage peaks: 500,724KB / 3,111,996KB. Property is satisfied.</p> <p>E<> P7_SASPFM.free_mark Verification/kernel/elapsed time used: 49.005s / 0.077s / 49.083s. Resident/virtual memory usage peaks: 549,240KB / 3,166,792KB. Property is satisfied.</p> <p>E<> P7_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.211s / 0.004s / 0.214s. Resident/virtual memory usage peaks: 555,528KB / 3,186,772KB. Property is satisfied.</p>	<p>E<> P8_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.207s / 0.006s / 0.213s. Resident/virtual memory usage peaks: 136,044KB / 2,623,376KB. Property is satisfied.</p> <p>E<> P8_SAPBC9.pass_ball Disconnected.</p> <p>E<> P8_SAPBC10.pass_ball Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Disconnected.</p> <p>E<> P8_SADBFW.drive_ball_forward Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Disconnected.</p> <p>E<> P8_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.261s / 0.004s / 0.266s. Resident/virtual memory usage peaks: 35,368KB / 2,511,588KB. Property is satisfied.</p> <p>E<> P8_SAPBC10.pass_ball Verification/kernel/elapsed time used: 0.269s / 0.005s / 0.274s. Resident/virtual memory usage peaks: 61,112KB / 2,545,396KB. Property is satisfied.</p> <p>E<> P8_SAPBC11.pass_ball Verification/kernel/elapsed time used: 0.264s / 0.004s / 0.269s. Resident/virtual memory usage peaks: 79,244KB / 2,569,476KB. Property is satisfied.</p> <p>E<> P8_SADBFW.drive_ball_forward Verification/kernel/elapsed time used: 0.287s / 0.007s / 0.294s. Resident/virtual memory usage peaks: 97,476KB / 2,594,580KB. Property is satisfied.</p> <p>E<> P8_SASPFM.free_mark Verification/kernel/elapsed time used: 0.289s / 0.007s / 0.295s. Resident/virtual memory usage peaks: 115,604KB / 2,618,656KB. Property is satisfied.</p> <p>E<> P8_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.232s / 0.003s / 0.234s. Resident/virtual memory usage peaks: 133,688KB / 2,643,760KB. Property is satisfied.</p>
<p>P9</p> <p>E<> P9_C.Side_Attack Established direct connection to local server. (Academic) UPPAAL version 4.1.4 (rev. 4835), June 2011 -- server. Verification/kernel/elapsed time used: 0.239s / 0.004s / 0.242s. Resident/virtual memory usage peaks: 28,092KB / 2,501,344KB. Property is satisfied.</p> <p>E<> P9_C.Mark Verification/kernel/elapsed time used: 0.232s / 0.003s / 0.236s. Resident/virtual memory usage peaks: 47,932KB / 2,526,956KB. Property is satisfied.</p> <p>E<> P9_MHB.hold_ball Verification/kernel/elapsed time used: 0.26s / 0.005s / 0.265s. Resident/virtual memory usage peaks: 67,280KB / 2,552,060KB. Property is satisfied.</p> <p>E<> P9_MSB.search_ball Verification/kernel/elapsed time used: 0.264s / 0.005s / 0.269s. Resident/virtual memory usage peaks: 86,480KB / 2,577,160KB. Property is satisfied.</p> <p>E<> P9_MIB.intercept_ball Verification/kernel/elapsed time used: 0.259s / 0.005s / 0.264s. Resident/virtual memory usage peaks: 105,620KB / 2,602,264KB. Property is satisfied.</p> <p>E<> P9_MSP.strategic_position Verification/kernel/elapsed time used: 0.227s / 0.004s / 0.231s. Resident/virtual memory usage peaks: 124,864KB / 2,627,368KB. Property is satisfied.</p> <p>E<> P9_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.676s / 0.03s / 0.706s. Resident/virtual memory usage peaks: 145,632KB / 2,652,468KB. Property is satisfied.</p>	<p>P10</p> <p>E<> P10_C.Side_Attack Verification/kernel/elapsed time used: 0.233s / 0.003s / 0.237s. Resident/virtual memory usage peaks: 350,808KB / 2,920,896KB. Property is satisfied.</p> <p>E<> P10_C.Mark Verification/kernel/elapsed time used: 0.219s / 0.003s / 0.222s. Resident/virtual memory usage peaks: 368,808KB / 2,944,988KB. Property is satisfied.</p> <p>E<> P10_MHB.hold_ball Verification/kernel/elapsed time used: 0.228s / 0.004s / 0.232s. Resident/virtual memory usage peaks: 386,800KB / 2,969,064KB. Property is satisfied.</p> <p>E<> P10_MSB.search_ball Verification/kernel/elapsed time used: 0.237s / 0.006s / 0.244s. Resident/virtual memory usage peaks: 404,856KB / 2,994,164KB. Property is satisfied.</p> <p>E<> P10_MIB.intercept_ball Verification/kernel/elapsed time used: 0.227s / 0.005s / 0.232s. Resident/virtual memory usage peaks: 422,808KB / 3,018,240KB. Property is satisfied.</p> <p>E<> P10_MSP.strategic_position Verification/kernel/elapsed time used: 0.225s / 0.004s / 0.229s. Resident/virtual memory usage peaks: 440,800KB / 3,042,316KB. Property is satisfied.</p> <p>E<> P10_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.438s / 0.017s / 0.456s. Resident/virtual memory usage peaks: 459,028KB / 3,067,424KB. Property is satisfied.</p> <p>E<> P10_SASB.search_ball Verification/kernel/elapsed time used: 0.224s / 0.004s / 0.229s. Resident/virtual memory usage peaks: 476,780KB / 3,091,500KB. Property is satisfied.</p>

<p>E<> P9_SASB.search_ball Verification/kernel/elapsed time used: 0.244s / 0.004s / 0.249s. Resident/virtual memory usage peaks: 164,188KB / 2,677,568KB. Property is satisfied.</p> <p>E<> P9_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.238s / 0.004s / 0.243s. Resident/virtual memory usage peaks: 183,424KB / 2,701,652KB. Property is satisfied.</p> <p>E<> P9_SAPBC10.pass_ball Verification/kernel/elapsed time used: 0.233s / 0.004s / 0.237s. Resident/virtual memory usage peaks: 202,628KB / 2,726,752KB. Property is satisfied.</p> <p>E<> P9_SAPBC11.pass_ball Verification/kernel/elapsed time used: 0.244s / 0.004s / 0.248s. Resident/virtual memory usage peaks: 221,828KB / 2,751,852KB. Property is satisfied.</p> <p>E<> P9_SADBFW.drive_ball_forward Verification/kernel/elapsed time used: 0.262s / 0.005s / 0.266s. Resident/virtual memory usage peaks: 240,968KB / 2,776,956KB. Property is satisfied.</p> <p>E<> P9_SAKTG.kick_to_goal Verification/kernel/elapsed time used: 0.429s / 0.014s / 0.443s. Resident/virtual memory usage peaks: 260,452KB / 2,802,056KB. Property is satisfied.</p> <p>E<> P9_SASPFM.free_mark Verification/kernel/elapsed time used: 0.233s / 0.003s / 0.236s. Resident/virtual memory usage peaks: 279,404KB / 2,827,156KB. Property is satisfied.</p> <p>E<> P9_SASP.strategic_position Verification/kernel/elapsed time used: 0.234s / 0.003s / 0.237s. Resident/virtual memory usage peaks: 298,536KB / 2,852,256KB. Property is satisfied.</p> <p>E<> P9_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.233s / 0.004s / 0.236s. Resident/virtual memory usage peaks: 317,772KB / 2,877,356KB. Property is satisfied.</p>	<p>E<> P10_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.22s / 0.004s / 0.224s. Resident/virtual memory usage peaks: 494,772KB / 3,115,576KB. Property is satisfied.</p> <p>E<> P10_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.221s / 0.004s / 0.225s. Resident/virtual memory usage peaks: 512,800KB / 3,147,844KB. Property is satisfied.</p> <p>E<> P10_SADBFW.drive_ball_forward Verification/kernel/elapsed time used: 0.223s / 0.004s / 0.227s. Resident/virtual memory usage peaks: 530,820KB / 3,172,944KB. Property is satisfied.</p> <p>E<> P10_SAKTG.kick_to_goal Verification/kernel/elapsed time used: 0.319s / 0.01s / 0.329s. Resident/virtual memory usage peaks: 548,884KB / 3,197,020KB. Property is satisfied.</p> <p>E<> P10_SASP.strategic_position Verification/kernel/elapsed time used: 0.214s / 0.004s / 0.218s. Resident/virtual memory usage peaks: 566,800KB / 3,221,096KB. Property is satisfied.</p> <p>E<> P10_SADBFS.drive_ball_fast Verification/kernel/elapsed time used: 0.449s / 0.019s / 0.468s. Resident/virtual memory usage peaks: 585,008KB / 3,246,196KB. Property is satisfied.</p> <p>E<> P10_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.213s / 0.004s / 0.216s. Resident/virtual memory usage peaks: 602,740KB / 3,270,272KB. Property is satisfied.</p>
<p>P11</p> <p>E<> P11_C.Side_Attack Verification/kernel/elapsed time used: 0.211s / 0.004s / 0.214s. Resident/virtual memory usage peaks: 635,752KB / 3,313,812KB. Property is satisfied.</p> <p>E<> P11_C.Mark Verification/kernel/elapsed time used: 0.251s / 0.005s / 0.256s. Resident/virtual memory usage peaks: 653,800KB / 3,337,888KB. Property is satisfied.</p> <p>E<> P11_MHB.hold_ball Verification/kernel/elapsed time used: 0.238s / 0.005s / 0.243s. Resident/virtual memory usage peaks: 671,752KB / 3,361,964KB. Property is satisfied.</p> <p>E<> P11_MSB.search_ball Verification/kernel/elapsed time used: 0.238s / 0.004s / 0.243s. Resident/virtual memory usage peaks: 689,780KB / 3,387,064KB. Property is satisfied.</p> <p>E<> P11_MIB.intercept_ball Verification/kernel/elapsed time used: 0.228s / 0.004s / 0.232s. Resident/virtual memory usage peaks: 707,764KB / 3,411,172KB. Property is satisfied.</p> <p>E<> P11_MSP.strategic_position Verification/kernel/elapsed time used: 0.22s / 0.003s / 0.223s. Resident/virtual memory usage peaks: 725,752KB / 3,435,248KB. Property is satisfied.</p> <p>E<> P11_MA.Set_Cognitive_Achieved Verification/kernel/elapsed time used: 0.414s / 0.016s / 0.43s. Resident/virtual memory usage peaks: 743,976KB / 3,460,348KB. Property is satisfied.</p> <p>E<> P11_SASB.search_ball Verification/kernel/elapsed time used: 0.228s / 0.004s / 0.232s. Resident/virtual memory usage peaks: 761,796KB / 3,484,424KB. Property is satisfied.</p>	<p>E<> P11_SAIB.intercept_ball Verification/kernel/elapsed time used: 0.217s / 0.004s / 0.22s. Resident/virtual memory usage peaks: 779,752KB / 3,508,500KB. Property is satisfied.</p> <p>E<> P11_SAPBC9.pass_ball Verification/kernel/elapsed time used: 0.229s / 0.005s / 0.233s. Resident/virtual memory usage peaks: 797,736KB / 3,532,576KB. Property is satisfied.</p> <p>E<> P11_SADBFW.drive_ball_forward Verification/kernel/elapsed time used: 0.227s / 0.004s / 0.231s. Resident/virtual memory usage peaks: 815,728KB / 3,557,676KB. Property is satisfied.</p> <p>E<> P11_SAKTG.kick_to_goal Verification/kernel/elapsed time used: 0.334s / 0.01s / 0.345s. Resident/virtual memory usage peaks: 833,892KB / 3,581,752KB. Property is satisfied.</p> <p>E<> P11_SASP.strategic_position Verification/kernel/elapsed time used: 0.214s / 0.004s / 0.217s. Resident/virtual memory usage peaks: 851,736KB / 3,605,828KB. Property is satisfied.</p> <p>E<> P11_SADBFS.drive_ball_fast Verification/kernel/elapsed time used: 0.431s / 0.017s / 0.449s. Resident/virtual memory usage peaks: 870,008KB / 3,630,928KB. Property is satisfied.</p> <p>E<> P11_SAF.Set_Cognitive_Failed Verification/kernel/elapsed time used: 0.214s / 0.003s / 0.218s. Resident/virtual memory usage peaks: 887,716KB / 3,655,004KB. Property is satisfied.</p>



TERMO DE AUTORIZAÇÃO PARA PUBLICAÇÃO DIGITAL NA BIBLIOTECA DIGITAL DA UFBA

1 Identificação do tipo de documento

Tese [] Dissertação [X] Monografia [] Trabalho de Conclusão de Curso []

2 Identificação do autor e do documento

Nome completo: Rui Carlos Botelho Almeida da Silva

CPF: 678.685.685 - 34

Telefone: 71 9665-5513 e-mail: ruicbs@gmail.com

Programa/Curso de Pós-Graduação/Graduação/Especialização: Programa de Pós-graduação em Mecatrônica - PPGM

Título do documento: VERIFICAÇÃO FORMAL DE PLANOS DE AGENTES AUTÔNOMOS E SISTEMAS MULTIAGENTES: UM ESTUDO DE CASO APLICADO AO FUTEBOL DE ROBÔS. Data da defesa: 09 / 03 / 12

3 Autorização para publicação na Biblioteca Digital da UFBA

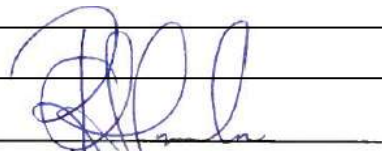
Autorizo com base no disposto na Lei Federal nº 9.610, de 19 de fevereiro de 1998 e na Lei nº 10.973, de 2 de dezembro de 2004, a Universidade Federal da Bahia (UFBA) disponibilizar gratuitamente sem ressarcimento dos direitos autorais, o documento supracitado, de minha autoria, na Biblioteca Digital da UFBA para fins de leitura e/ou impressão pela Internet a título de divulgação da produção científica gerada pela Universidade.

Texto completo [X] Texto parcial []

Em caso de autorização parcial, especifique a (s) parte(s) do texto que deverão ser disponibilizadas:

Salvador - Bahia, 09 / 03 / 12

Local Data



Rui Carlos Botelho Almeida da Silva

4 Restrições de acesso ao documento

Documento confidencial? [X] Não

[] Sim Justifique: _____

Informe a data a partir da qual poderá ser disponibilizado na Biblioteca Digital da UFBA:

01 / 01 / 2013 [] Sem previsão

Assinatura do Orientador: _____ (Opcional)

O documento está sujeito ao registro de patente? Não [X]

Sim []

O documento pode vir a ser publicado como livro? Sim [X]

Não []

Universidade Federal da Bahia

Sistema de Biblioteca da UFBA

Grupo Técnico da Biblioteca da UFBA



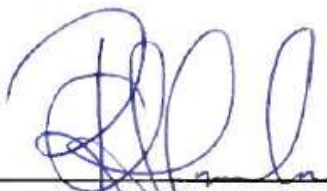
**CADASTRO DE INFORMAÇÕES PARA PUBLICAÇÃO DIGITAL
NA BIBLIOTECA DIGITAL DA UFBA**

1. Identificação do tipo de documento	
Tese () Dissertação (X) Monografia () Trabalho de Conclusão de Curso ()	
2. Colegiado do Curso de Pós-Graduação:	
Título: VERIFICAÇÃO FORMAL DE PLANOS DE AGENTES AUTÔNOMOS E SISTEMAS MULTIAGENTES: UM ESTUDO DE CASO APLICADO AO FUTEBOL DE ROBÔS.	
Autor(a): Rui Carlos Botelho Almeida da Silva	
CPF: 678.685.685-34	E-mail: ruicbs@gmail.com
Orientador(a)::	
Nome: Profa. Dra. Aline Maria Santos Andrade	
CPF: 159.207.635-15	E-mail: aline@ufba.br
Co-orientadores::	
Nome: Prof. Dr. Augusto César Pinto Loureiro da Costa	
CPF: 211637905-91	E-mail: augusto.loureiro@ufba.br
Membros da Banca	
Nome: Profa. Dra. Aline Maria Santos Andrade	
CPF: 159.207.635-15	E-mail: aline@ufba.br
Nome: Prof. Dr. David Boris Paul Déharbe	
CPF: 008.090.854-37	E-mail: deharbe@gmail.com
Nome: Prof. Dr. Flávio Morais de Assis Silva	
CPF: 763.873.136-00	E-mail: fassis@ufba.br
Nome:	
CPF:	E-mail:
Data de Homologação Pós-Graduação:	
Financiadores: FAPESB (APR 0187/2008)	
Data:	
Assinatura:	

Salvador, 09/03/2012

DECLARAÇÃO

Declaro para os devidos fins que o texto final apresentado para a conclusão do meu curso de Mestrado em Mecatrônica da Universidade Federal da Bahia é de minha autoria. Declaro também que quaisquer informações utilizadas neste texto, mas que sejam provenientes de outros trabalhos tem fonte claramente expressa e, quando for o caso, foram devidamente autorizadas pelo(s) respectivo(s) autor(es).



Nome: Rui Carlos Botelho Almeida da Silva
CPF: 678.685.685-34