UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS

José Gilmar Nunes de Carvalho Filho

MODELAGEM E SÍNTESE PARA COORDENAÇÃO DE SISTEMAS MULTI-ROBÔS BASEADA NUMA ESTRUTURA DE JOGO

Florianópolis

2012

José Gilmar Nunes de Carvalho Filho

MODELAGEM E SÍNTESE PARA COORDENAÇÃO DE SISTEMAS MULTI-ROBÔS BASEADA NUMA ESTRUTURA DE JOGO

Dissertação submetida ao programa de Pós-Graduação em Engenharia de Automação e Sistemas para a obtenção do Grau de Mestre em Engenharia de Automação e Sistemas.
Orientador: Jean-Marie Alexandre Farines, Dr.
Coorientador: José Eduardo Ribeiro Cury, Dr.

Florianópolis

2012

Catalogação na fonte pela Biblioteca Universitária
da
Universidade Federal de Santa Catarina

C331m Carvalho Filho, José Gilmar Nunes de
Modelagem e síntese para coordenação de sistemas multi-robôs baseada numa estrutura de jogo [dissertação] / José Gilmar Nunes de Carvalho Filho ; orientador, Jean-Marie Alexandre Farines. - Florianópolis, SC, 2012.
166 p.: il., tabs.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de sistemas. 2. Automação. 3. Robôs - Sistemas de controle. 4. Modelagem computacional. I. Farines, Jean Marie. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

CDU 621.3-231.2(021)

José Gilmar Nunes de Carvalho Filho

MODELAGEM E SÍNTESE PARA COORDENAÇÃO DE SISTEMAS MULTI-ROBÔS BASEADA NUMA ESTRUTURA DE JOGO

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Engenharia de Automação e Sistemas", e aprovada em sua forma final pelo programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis, 28 de fevereiro 2012.

José Eduardo Ribeiro Cury, Dr.
Coordenador

Banca Examinadora:

Jean-Marie Alexandre Farines, Dr
Presidente

José Eduardo Ribeiro Cury, Dr

Jomi Fred Hubner, Dr

Max Hering de Queiroz, Dr

Rosvelter João Coelho da Costa, Dr

AGRADECIMENTOS

Primeiramente agradeço a Deus, que sempre iluminou a minha caminhada. Sem Ele, nada somos, nada podemos.

Aos meus pais Dôra e Gilmar que compartilharam comigo os momentos mais sinuosos desta longa caminhada, apoiando-me independente dos obstáculos. Sou grato pela educação, amor, carinho e dedicação.

A "Nin", minha segunda mãe, por todos esses anos de cuidados, atenção, carinho e amor incondicional.

A minha amada Tamires, pela dedicação, amor e, acima de tudo, por ter compreendido e aceitado minhas ausências.

A Letícia, minha irmã, pela amizade incondicional e pelo apoio que nunca me foi negado.

A minha irmã Lívia, pelo carinho, amizade e confiança em meu potencial e às minhas sobrinhas Ana Flávia e Ana Sofia, pelo afeto.

Aos meus avós, Vovô Dezinho, que proporcionou meus primeiros contatos com o mundo da eletrônica e a Vovó Castália pelo carinho e amor. A "Vô Tonho" e "Vó Gilza" (in memorian), pelo modelo de simplicidade, honestidade, carinho e amor à família.

Agradeço também aos amigos do LTIC, em especial a Guinter, Leo, Luciano, Luizinho, "Presunto", Rattus, Thiago e Vanderley, pela amizade, aprimoramento das ideias e não poderia deixar de mencionar os momentos de diversão.

A Lucas, pela amizade e colaboração desde meus anos de pesquisa na UFS.

Aos meus Orientadores, Jean-Marie e Cury, sem a ajuda dos quais, esse trabalho não haveria de ser realizado.

A Renan pela ajuda na realização dos experimentos.

A Piterman e Yaniv pela atenção, explicações acerca do método desenvolvido por eles e por compartilharem seus códigos e exemplos.

Por fim agradeço a CAPES pelo apoio financeiro que tornou possível a realização deste trabalho.

Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrepende.

Leonardo da Vinci

RESUMO

Ao longo das últimas décadas, Sistemas Multi-Robôs (SMR) têm sido aplicados a diversas áreas com o intuito de aumentar a eficiência e a robustez na execução das tarefas a serem desempenhadas por eles. Problemas de planejamento de trajetória e de tarefas são alguns dos principais desafios desta área, destacando-se que a separação entre estes níveis de planejamento permite que o problema de controle seja abordado em diferentes níveis de abstração. Neste trabalho, é tratado o problema de planejamento de tarefas na coordenação de SMRs, utilizando métodos baseados em estruturas de jogos. Especificamente, é analisada a aplicabilidade desta teoria na geração de controladores que garantam segurança na execução das aplicações, evitando comportamentos de colisão entre robôs ou com outros objetos do espaço de trabalho. Além disso, também é analisado o aspecto da obtenção de estratégias que sejam tolerantes a falta. Foram propostos princípios de modelagem, considerando o método proposto em Piterman *et al.* (2006), a partir dos quais várias aplicações foram modeladas. Uma vez modeladas as aplicações, foram obtidas estratégias de coordenação para cada um dos casos e, a partir dos coordenadores gerados, foram realizados experimentos e analisadas a aplicabilidade de estruturas de jogos no problema de coordenação de SMRs.

Palavras-chave: Coordenação de SMRs, Planejamento de tarefas, Estrutura de jogo.

ABSTRACT

Over last decades, Multi-Robot Systems (MRS) have been applied in several areas in order to increase the efficiency and robustness in tasks execution. Path and tasks planning problems are among the main challenges in this area, pointing out that the separation between these levels of planning allows the control problem be approached at different levels of abstraction. This work is concerned with the problem of tasks planning in the coordination of MRS, using methods based on games' structures. Specifically, it is analyzed the applicability of this theory in the generation of controllers that ensure safety in the execution of applications, avoiding behaviors as collision with other robots or objects in the workspace. Furthermore, it is also analyzed obtaining aspects of fault tolerant strategies. Modeling principles have been proposed, considering the method proposed in Piterman et al.(2006), from which several applications have been modeled. Once modeled the applications, were obtained coordinating strategies for each case, and from the generated coordinators, experiments were performed and it was analyzed the applicability of game structures in the problem of coordinating MRS.

Keywords: MRSs coordination, Task planning, Game Structure.

LISTA DE FIGURAS

Figura 1 Taxonomia proposta em Iocchi *et al.* (2001) 32
Figura 2 Exemplo de um autômato 40
Figura 3 SED em malha fechada [19]. 44
Figura 4 Verificação de existência de estratégia vencedora 56
Figura 5 Representação parcial do processo de síntese do controlador . 59
Figura 6 Caso utilizado para exemplificar o método proposto em [56] . 59
Figura 7 Controlador obtido para o exemplo referente à figura 6. 62
Figura 8 Exemplo de espaço de trabalho e relação de vizinhança correspondente 69
Figura 9 Situações de colisão 71
Figura 10 Exemplos de espaço de trabalho 75
Figura 11 Exemplos de espaço de trabalho com portas, robô adversário e OPD 81
Figura 12 Tipos de alcançabilidade 100
Figura 13 Espaço de trabalho do caso de alcançabilidade, considerando portas não controladas 115
Figura 14 Representação do controlador obtido para o PAS com observabilidade global 117
Figura 15 Representação do controlador obtido para o PAS com observabilidade parcial 118
Figura 16 Espaço de trabalho considerando um robô adversário 121
Figura 17 Representação do controlador obtido para o PAC, observabilidade global 124
Figura 18 Representação do controlador obtido para o PAC, observabilidade parcial 125
Figura 19 Espaço de trabalho referente ao experimento considerando tolerância à falta 127
Figura 20 Representação do controlador obtido para o PAC, considerando falta 129
Figura 21 Espaço de trabalho do caso de busca e resgate 131
Figura 22 Representação do controlador obtido para o PBR com observabilidade global 136
Figura 23 Representação do controlador obtido para o PBR com observabilidade parcial 137

Figura 24 Robô e espaço de trabalho considerados nos experimentos apresentados em [54] 140
Figura 25 Espaço de trabalho e trajetória referentes ao experimento 141

LISTA DE TABELAS

Tabela 1 Resultados dos experimentos 142
Tabela 2 Dados sobre os controladores gerados 144
Tabela 3 Operadores lógicos e temporais 161
Tabela 4 Equivalência entre valores 162

LISTA DE ABREVIATURAS E SIGLAS

CTL	*Computation Tree Logic*
LT	Lógica Temporal
LTL	*Linear Temporal Logic*
LTLMoP	*Linear Temporal Logic Mission Planning*
OPD	Objeto com posição desconhecida
PAC	Problema de Alcançabilidade Cooperativa
PAS	Problema de Alcançabilidade Simples
RI	Região de interferência
RM	Região de Movimentação
SED	Sistemas a Eventos Discretos
SMR	Sistema Multi-Robôs
TCC	Teoria Clássica de Controle
TCS	Teoria de Controle Supervisório
TCTL	Timed Computation Tree Logic

LISTA DE SÍMBOLOS

Σ Alfabeto de eventos controláveis e não controláveis
Σ_c Alfabeto de eventos controláveis
Σ_u Alfabeto de eventos não controláveis
Q Espaço de estados
f Função de transição
q_0 Estado inicial
Q_m Conjunto de estados finais
$\mathcal{P}$ Conjunto de proposições
σ Sequência de estados
$\bigcirc$ Operador temporal *Next*
$\Box$ Operador temporal *sempre*
$\Diamond$ Operador temporal *no futuro*
$\mathcal{U}$ Operador temporal *até que*
$\mathcal{L}(G)$ Linguagem Gerada
$\mathcal{L}_m(G)$ Linguagem Marcada
V Conjunto com todas as variáveis de estado
$\mathcal{X}$ Conjunto com as variáveis de estado do ambiente
$\mathcal{Y}$ Conjunto com as variáveis de estado do sistema
Θ Asserção que caracteriza os estados iniciais
ρ_e Dinâmica do ambiente
ρ_e Dinâmica do sistema
φ Condição de vitória
φ^e Modelo do ambiente
φ_i^e Condição inicial do ambiente
φ_t^e Dinâmica do ambiente
φ_g^e Condição de justiça do ambiente
φ^s Modelo do sistema
φ_i^s Condição inicial do sistema
φ_t^s Dinâmica do sistema
φ_g^s Condição de justiça do sistema
$\varphi^{R_{Si}}$ Modelo de um robô do sistema, considerando observabilidade global

$\varphi^{Rp_{Si}}$ Modelo de um robô do sistema, considerando observabilidade parcial

$\varphi^{RF_{Si}}$ Modelo de um robô do sistema com possibilidade de falha, considerando observabilidade global

$\varphi^{RFp_{Si}}$ Modelo de um robô do sistema com possibilidade de falha, considerando observabilidade parcial

φ^{NCol} Modelo do mecanismo de não colisão entre agentes

φ^{P} Modelo de uma porta

φ^{R_A} Modelo de um robô adversário

φ^{Ob} Modelo de um objeto com posição desconhecida

V_{ind} Variável indicadora

φ_{alcS} Especificação de alcançabilidade simples

φ_{alcC} Especificação de alcançabilidade suficiente

φ_{alcA} Especificação de alcançabilidade em conjunto

φ_{cSuf} Especificação de cobertura de objetivos suficientes

φ_{cNes} Especificação de cobertura de objetivos necessários

SUMÁRIO

1 INTRODUÇÃO 27
1.1 OBJETIVOS 28
2 COORDENAÇÃO DE SISTEMAS MULTI-ROBÔS 31
2.1 O PROBLEMA DE COORDENAÇÃO DE SMRS 31
2.2 CARACTERIZAÇÃO DE SMRS 32
2.3 REATIVIDADE E DELIBERAÇÃO 33
2.3.1 Reatividade e deliberação num robô 33
2.3.2 Reatividade e deliberação num SMR 34
2.4 CLASSES DE APLICAÇÕES 35
2.4.1 Busca e coleta 35
2.4.2 Observação de múltiplos alvos 36
2.4.3 Movimentação de Objetos 36
2.4.4 Exploração e movimentação em bando 36
2.4.5 Futebol de robôs 37
2.5 CONCLUSÃO 38
3 MÉTODOS PARA COORDENAÇÃO DE SMRS BASEADOS EM SED 39
3.1 FUNDAMENTOS DE SISTEMAS A EVENTOS DISCRETOS 39
3.1.1 Formalismo para descrição de SEDs 40
3.1.2 Verificação Lógica 41
3.2 CONTROLE SUPERVISÓRIO 44
3.3 PLANEJAMENTO DE TAREFAS BASEADO EM *MODEL CHECKING* 46
3.4 TEORIA DE JOGOS 48
3.5 PITERMAN *ET. AL* (2006) [56] 51
3.5.1 μ-calculus sobre estruturas de jogos 52
3.5.2 Notação vetorial da condição de vitória 53
3.5.3 Algoritmos 55
3.5.4 Exemplo 59
3.6 DISCUSSÃO SOBRE AS ABORDAGENS 63
3.7 CONCLUSÃO 65
4 DESCRIÇÃO DO PROBLEMA 67
4.1 FORMULAÇÃO DO PROBLEMA 67
4.2 ELEMENTOS DE MODELAGEM 69
4.2.1 Espaço de trabalho 69
4.2.2 Sistema 70
4.2.3 Ambiente 71

4.2.4 Situações de falha ... 72
4.3 CLASSES DE APLICAÇÕES ... 74
4.4 CONCLUSÃO ... 77
5 PRINCÍPIOS DE MODELAGEM BASEADA EM ESTRUTURAS DE JOGOS ... 79
5.1 FERRAMENTAS DE MODELAGEM BASEADAS EM PITERMAN *ET AL.* (2006) ... 79
5.2 MODELOS CONSIDERANDO OBSERVABILIDADE GLOBAL 82
5.2.1 Sistema ... 82
5.2.2 Ambiente ... 85
5.2.2.1 Porta ... 85
5.2.2.2 Robô adversário ... 86
5.2.2.3 Objeto com posição desconhecida (OPD) ... 87
5.3 MODELOS CONSIDERANDO OBSERVABILIDADE PARCIAL 89
5.3.1 Sistema ... 89
5.3.2 Ambiente ... 91
5.3.2.1 Porta ... 91
5.3.2.2 Robôs adversários ... 92
5.3.2.3 Objeto com posição desconhecida (OPD) ... 93
5.4 SITUAÇÕES DE FALHA ... 96
5.4.1 Robô com possibilidade de falha ... 96
5.4.2 Simulador de falha ... 97
5.5 MODELAGEM DOS OBJETIVOS BÁSICOS DO SISTEMA ... 99
5.5.1 Alcançabilidade ... 100
5.5.2 Cobertura ... 102
5.5.3 Sequenciamento ... 105
5.6 METODOLOGIA PARA COMPOSIÇÃO DE ELEMENTOS DE MODELAGEM ... 107
5.7 CONCLUSÃO ... 111
6 RESOLUÇÃO DE DIFERENTES PROBLEMAS DE COORDENAÇÃO ... 113
6.1 PROBLEMA DE ALCANÇABILIDADE SIMPLES ... 113
6.1.1 Modelagem das especificações ... 113
6.1.2 Exemplo ... 114
6.2 PROBLEMA DE ALCANÇABILIDADE COOPERATIVA ... 120
6.2.1 Modelagem das especificações ... 120
6.2.2 Exemplo 1: Robô sem Falha ... 121
6.2.3 Exemplo 2: Robôs com possibilidade de falha ... 126
6.3 PROBLEMA DE BUSCA E RESGATE ... 129
6.3.1 Modelagem das especificações ... 129
6.3.2 Exemplo ... 131

6.4 EXPERIMENTOS REALIZADOS 139
6.4.1 Arquitetura utilizada 139
6.4.2 Descrição do experimento 140
6.5 DISCUSSÕES 143
6.6 CONCLUSÃO 146
7 CONCLUSÕES E PERSPECTIVAS 147
Perspectivas de trabalho 148
Referências Bibliográficas 151
APÊNDICE A – Modelagem em SMV 159

1 INTRODUÇÃO

Ao longo das últimas décadas, Sistemas Multi-Robôs (SMR) têm sido aplicados a diversas áreas e em diferentes configurações, principalmente, com o intuito de aumentar a eficiência e a robustez na execução das tarefas a serem desempenhadas por eles. Contudo, ao utilizar um sistema composto por vários agentes móveis autônomos, é necessário lidar com problemas relacionados com a interferência entre robôs, falhas de comunicação e a distribuição de tarefas, por exemplo. Além desses, ainda há o aumento na complexidade de outros aspectos que também são comuns a sistemas com apenas um agente, como a dinamicidade do ambiente.

Também é necessário desenvolver métodos que facilitem a obtenção de controladores ou estratégias que garantam o correto cumprimento das tarefas que lhe são impostas, sem que comportamentos indesejados, como a colisão entre agentes ou com obstáculos, ocorram. Nesse sentido, a separação entre planejamento de tarefas/atividades e de trajetória se mostra bastante promissora, pois permite tratar o problema de controle em diferentes níveis de abstração. Além disso, como a execução de tarefas/atividades está associada à ocorrência de eventos e ações, métodos baseados na teoria de Sistemas a Eventos Discretos (SED) são bastante adaptados para obtenção desta parte do planejamento. Outra vantagem importante desta teoria é que os modelos gerados, em geral, são passíveis de verificação formal, o que pode ser utilizado para garantir a presença ou ausência de propriedades e comportamentos específicos na dinâmica destes sistemas.

Neste trabalho, é abordada apenas a parte referente ao planejamento de tarefas em SMRs. Além disso, no restante desta dissertação, este nível do controlador é chamado de Coordenador, uma vez que ele será responsável por coordenar a ações dos robôs a fim de atender as especificações do sistema. Também são apresentadas algumas teorias e métodos baseados em SED para solução do problema de coordenação de SMRs. Na sequência, são propostos modelos para representar elementos e comportamentos básicos dos SMRs, os quais são utilizados para descrever os sistemas tratados. Além disso, também é proposta uma metodologia para modelagem de diferentes classes de aplicações através da composição destes modelos básicos.

Este trabalho está estruturado da seguinte forma. No restante deste capítulo são discutidos os objetivos do trabalho. No capítulo 2 são apresentados os principais conceitos a cerca do problema de coordenação de SMRs, uma taxonomia para classificação destes sistemas e algumas classes de aplicações significativas, definidas em Iocchi *et al.* (2001)[25]. No capítulo 3 são discutidas as abordagens para o problema de coordenação baseadas na teoria de

SED, focando no método proposto em Piterman *et al.* (2006) [56].

Em seguida, a descrição do problema tratado neste trabalho e da forma de modelagem baseada no método proposto em [56], são apresentadas nos capítulos 4 e 5, respectivamente. No capítulo 6, são mostrados e discutidos os resultados obtidos com a aplicação dos modelos e da metodologia propostos neste trabalho em várias classes de aplicações. Por fim, no capítulo 7, são apresentadas as conclusões e perspectivas do trabalho desenvolvido.

1.1 OBJETIVOS

De maneira geral, o objetivo deste trabalho é o estudo do problema de coordenação de sistemas multi-robôs, sob a ótica da teoria de estruturas de jogos, e o desenvolvimento e validação de uma metodologia para obtenção do coordenador de um SMR.

Vale destacar destacar que, os princípios de modelagem propostos neste trabalho tiveram como principal finalidade, a validação e análise da aplicabilidade de estruturas de jogos e do método proposto em [56] no problema de coordenação de SMRs. Contudo, os modelos e metodologia desenvolvidos também representam um primeiro esforço na criação de um *framework* para descrição de aplicações e obtenção de estratégias para coordenação de SMRs. Assim, as metas para atingir o objetivo principal deste trabalho foram as seguintes:

- Estudo a respeito da aplicabilidade de estruturas de jogos na solução do problema de coordenação de SMRs. Através de uma revisão da literatura da área, foi realizada uma primeira avaliação a cerca da aplicabilidade de estruturas de jogos na coordenação de SMRs, identificando métodos e ferramentas adequadas para o escopo deste trabalho.

- Desenvolvimento de modelos para representar os diversos elementos dos SMRs e comportamentos desejados. Uma vez definido o método a ser utilizado para descrever as aplicações, foram desenvolvidos modelos para cada um dos elementos dos SMRs e comportamentos básicos que podem estar presentes nelas.

- Desenvolvimento de uma metodologia para realizar a composição de modelos elementares. Após terem sido criados os modelos para cada um dos elementos básicos, foi proposta uma metodologia para descrição de aplicações através da composição dos modelos de cada um dos seus elementos.

- Análise da aplicabilidade de estruturas de jogos na obtenção de coordenadores e validação dos princípios de modelagem propostos. Considerando algumas classes de aplicações, exemplos simples, mas representativos, de cada classe foram descritos utilizando os princípios de modelagem propostos neste trabalho. Em seguida, os algoritmos propostos em [29, 56] foram aplicados aos modelos, sendo obtidas estratégias de execução para cada um dos casos. Além disso, alguns dos coordenadores obtidos foram implementados utilizando a arquitetura de navegação proposta na dissertação Pavei (2011) [54].

2 COORDENAÇÃO DE SISTEMAS MULTI-ROBÔS

Neste capítulo, é apresentado o problema de coordenação de SMRs e são discutidos alguns de seus aspectos, sendo destacada uma forma de caracterização destes sistemas, segundo as dimensões definidas por Iocchi *et al.* (2001)[25]. Ao final, são apresentadas cinco classes de aplicações frequentemente utilizadas como casos de estudo em diversos trabalhos. Vale destacar que este capítulo é fortemente baseado nos trabalhos descritos em Iocchi *et al.* (2001)[25].

2.1 O PROBLEMA DE COORDENAÇÃO DE SMRS

Ao longo das últimas décadas, diversas topologias de SMRs têm sido propostas com o intuito de agregar robustez e eficiência na execução de tarefas, ou ainda para permitir a realização destas quando sistemas com apenas um agente não são capazes de executá-las [25]. Surge, então, o problema de coordenação de SMRs, definido a seguir.

Definição 2.1. *O problema de coordenação de SMRs pode ser definido como a obtenção de uma estratégia (ou de um controlador que a implementa) que, considerando a dinâmica do ambiente, coordene as ações dos agentes do sistema a fim de que o objetivo global deste seja satisfeito.*

Na literatura, são encontradas diversas configurações de SMRs, que se diferenciam pelos tipos de agentes (homogêneos ou heterogêneos) e a forma como estes estão organizados (centralizada ou distribuída), dentre outros aspectos. Assim, é possível perceber a necessidade de um bom método para classificar os sistemas, facilitando a identificação e, inclusive, o planejamento destes.

2.2 CARACTERIZAÇÃO DE SMRS

Em Iocchi *et al.* (2001)[25], é proposta uma taxonomia para classificação de sistemas multi-robôs baseada nos tipos de mecanismos utilizados para coordenação dos agentes. Na figura 1 são apresentadas as quatro dimensões usadas para caracterizar os SMRs, segundo Iocchi *et al.* (2001).

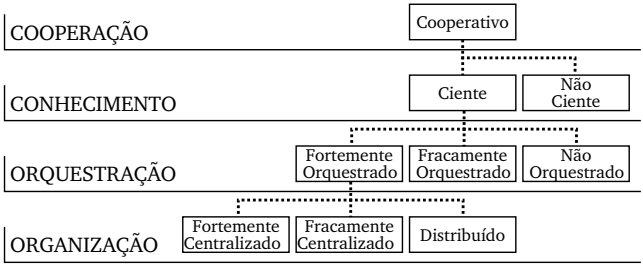


Figura 1: Taxonomia proposta em Iocchi *et al.* (2001)

A dimensão *Cooperação*, mostrada na figura 1, indica se os robôs operam juntos para realizar uma tarefa global que, ou não poderia ser executada por um único agente, ou cuja execução pode ser melhorada pelo uso de mais de um robô. Já a segunda dimensão (*Conhecimento*), está associada ao fato dos agentes do sistema terem conhecimento ou não da existência dos outros robôs do sistema.

Por sua vez, a existência de *Orquestração* indica que os robôs consideram as ações dos outros agentes, de modo que o sistema como um todo opera de forma coerente e coordenada. Além disso, quando a orquestração se baseia na utilização de um *protocolo de coordenação*[1] ela é dita *forte*, caso contrário a orquestração é *fraca*. Por último, a dimensão *Organização* indica a forma como o controle do SMR está organizado: Centralizado, quando um único agente (robô, PC, $\cdots$) comanda todos os outros ou distribuído, quando o sistema de controle é implementado em cada agente [25].

Na figura 1 é importante destacar que, se não existe uma ligação entre o valor de uma dimensão e os valores do nível inferior, é porque a classificação termina no último nível definido. Por exemplo, se um SMR é fracamente ou não-orquestrado, isso significa que não existe no sistema, segundo a taxonomia de Iocchi, o nível de organização. De forma análoga, se é dito que um SMR é distribuído, subentende-se que ele é cooperativo, ciente e fortemente orquestrado.

[1] Um protocolo de coordenação é um conjunto de regras que os robôs devem seguir a fim de interagir uns com os outros no espaço de trabalho

2.3 REATIVIDADE E DELIBERAÇÃO

Reatividade (*reactivity*) e deliberação (*deliberation*) são importantes características, tanto dos robôs quanto do sistema como um todo. De forma geral, estes aspectos estão associados ao tipo de arquitetura (ou mesmo estratégia) implementada a fim de realizar as tarefas do sistema. A seguir, estas características são abordadas considerando um robô, depois os conceitos são extendidos para SMR.

2.3.1 Reatividade e deliberação num robô

Quando é considerado apenas um robô, o tipo de comportamento assumido pelo agente na realização das tarefas associadas a ele tem sido abordado através do desenvolvimento de uma arquitetura de navegação que implemente o comportamento desejado. Neste sentido, existem três abordagens que são amplamente utilizadas: baseada em comportamento (*behavior-based*), sentir-modelar-planejar-agir (*sense-model-plan-act*) e a híbrida [25].

Abordagem reativa

Na primeira abordagem, baseada em comportamento, o controlador consiste num conjunto de comportamentos que são utilizados para ativar ou manter determinados objetivos, tais como realizar uma determinada tarefa e desviar de obstáculos, dentre outros. Além disso, esse tipo de abordagem (ou arquitetura) é caracterizado por um forte acoplamento entre percepção (sensoriamento) e atuação e por não utilizar modelos do ambiente em que o robô está operando.

De modo geral, as arquiteturas baseadas em comportamentos permitem que o agente responda rapidamente às mudanças no ambiente e não dependa de modelos do ambiente. Entretanto, características como a ausência de memória e dificuldade de lidar com tarefas globalmente complexas, tornam esta abordagem inapropriada para lidar com tarefas de alto nível, que exijam otimalidade em algum aspecto [46].

Abordagem deliberativa

Por sua vez, a abordagem sentir-modelar-planejar-agir é caracterizada pelo seu comportamento deliberativo. Este tipo de arquitetura se baseia na realização de quatro passos, são eles: sentir, modelar, planejar e agir. No primeiro passo, é realizada a coleta de dados sobre o ambiente a partir dos sensores do robô. Depois, estes dados são utilizados para construir um modelo simbólico do ambiente em que o robô está inserido.

Em seguida, o modelo do ambiente é utilizado para realizar o planejamento, que consistem numa sequência de ações que leva o robô a atingir seus objetivos. Por último, o plano gerado é convertido numa sequência de comandos de baixo nível, que serão enviados aos atuadores.

Assim, as arquiteturas baseadas nesta abordagem (*sense-model-plan-act*) são caracterizadas por possibilitar a obtenção de um comportamento ótimo, ou racional. Contudo, devido ao custo computacional inerente às etapas de modelagem e planejamento, este tipo de abordagem não é apropriado em casos onde os sistemas precisam responder rapidamente às mudanças no ambiente. Além disso, por utilizar modelos do ambiente para planejar a sequência de ações a ser executada, este tipo de abordagem precisa de informações confiáveis sobre o ambiente[46].

Abordagem híbrida

As abordagens híbridas combinam características das arquiteturas reativas e deliberativas. Desta forma, as arquiteturas híbridas permitem que as estratégias implementadas pelo robô considerem a execução ótima, segundo algum aspecto (como o tempo de execução), de uma tarefa e que o agente possa responder de forma rápida às mudanças ou imprevisibilidades do ambiente.

2.3.2 Reatividade e deliberação num SMR

Os aspectos de reatividade e deliberação num SMR foram descritos, em Iocchi *et al.*(2001)[25], com base na habilidade do time de robôs agir em resposta às mudanças no ambiente. Neste sentido, a deliberação e reatividade são definidas como segue.

SMR deliberativo Num SMR deliberativo, os robôs lidam com as mudanças ocorridas no ambiente como um time. Desta forma, as tarefas dos robôs são reorganizadas de modo a alcançar o objetivo global do sistema de forma eficiente [25].

SMR reativo Num SMR reativo, cada robô lida com as mudanças ocorridas no ambiente individualmente, reorganizando suas próprias tarefas de modo a realizar o objetivo que foi inicialmente designado para ele [25].

2.4 CLASSES DE APLICAÇÕES

Em Iocchi *et al.* (2001)[25], o principais tipos de tarefas em SMRs são agrupados em cinco classes de aplicações, são estas: Busca e coleta (*foraging*), observação de múltiplos alvos, movimentação de objetos (*box pushing*), exploração e movimentação em bando (*exploration and flocking*) e futebol de robôs. Nas subseções a seguir são tratadas cada uma destas classes.

2.4.1 Busca e coleta

Aplicações com tarefas de busca e coleta são frequentemente utilizadas como casos de estudo para SMRs, principalmente, por conta de suas similaridades com tarefas como operações de busca e resgate, limpeza de lixo tóxico e limpeza de minas, dentre outras [25].

Nestas tarefas é requerido que um agente colete objetos espalhados pelo espaço de trabalho. Em seguida, os objetos devem ser entregues em posições específicas. No contexto de SMRs, pode haver variações como diferenciação entre os agentes, de modo que certos objetos só podem ser coletados por um tipo de agente, ou que um tipo de robô é utilizado apenas para encontrar os objetos e outro tipo para coletar.

Existem diversos métodos que podem ser utilizados para resolver o problema de coordenação de SMRs com tarefas de *foraging*. Contudo, destaca-se a utilização de arquiteturas reativas, compostas por agentes robóticos com controladores baseados em comportamento. Por exemplo, em [6] é proposta a utilização de um SMR cooperativo, não ciente e reativo com agentes robóticos heterogêneos[2] e que possuem arquiteturas baseadas em comportamentos para realização de tarefas de coleta onde existem dois tipos de objetos, que devem ser entregues em locais diferente.

Já em [27], [42] e [51], tipos de tarefas similares aos considerados em [6], como retirada de lixo e limpeza de superfícies molhadas num escritório, foram tratados utilizando SMRs cooperativos, cientes, fortemente coordenados e distribuídos, com agentes heterogêneos e arquiteturas baseadas em comportamentos.

[2] Agentes homogenêos são iguais em *hardware* e *software* de controle, caso haja diferenças em algum destes aspectos eles são ditos heterogêneos

2.4.2 Observação de múltiplos alvos

As tarefas de observação de múltiplos alvos vêm sendo estudadas a pouco tempo, tendo sido introduzidas em Parker (1999)[52]. Contudo, devido a sua aplicabilidade em problemas de segurança, vigilância e reconhecimento, dentre outros, o interesse por este tipo de tarefa tem crescido bastante.

Nestas classes de aplicações, o objetivo geral é maximizar o tempo em que cada um dos alvos móveis está sendo observado por pelo menos um dos agentes do sistema. Neste contexto, é comum a ocorrência de problemas relacionados com comunicação, fusão de dados, cooperação e coordenação dos movimentos dos robôs.

Assim como ocorreu para as tarefas de *foraging*, os trabalhos considerando tarefas de observação de múltiplos alvos, tendo sido desenvolvidos em SMRs reativos, cientes e fortemente coordenados. Além disso, são utilizados agentes homogêneos e com arquitetura baseada em comportamentos [25, 52].

2.4.3 Movimentação de Objetos

Tarefas de movimentação de objetos estão associadas a problemas de estocagem, carga e descarga de *containers*, etc. Além disso, no contexto de SMRs, problemas relacionados com a alocação dos objetos, robustez do sistema e comunicação entre agentes, são inerentes às aplicações com este tipo de tarefas [36, 53].

Para este tipo de tarefas, várias são as propostas de solução. Em Kube e Bonabeau (2000)[36], por exemplo, é utilizado um SMR reativo e não-ciente. Além disso, o sistema proposto em [36] é um exemplo clássico da abordagem de enxame, tendo agentes homogêneos com controladores baseados em comportamentos. Já em [50] é proposto um SMR fortemente coordenado, fracamente centralizado e com agentes heterogêneos, implementados com uma arquitetura híbrida.

2.4.4 Exploração e movimentação em bando

Em tarefas de exploração e movimentação em bando, é requerido que os agentes do sistema se movimentem de maneira coordenada, seja para explorar um ambiente e construir um mapa dele ou mesmo para manter uma formação durante o movimento.

Deste modo, em aplicações de exploração, o objetivo é que os agentes sejam distribuídos o mais espaçadamente possível pelo ambiente, a fim de

que não haja (ou que se minimize) a sobreposição das informações obtidas por eles. Por sua vez, o objetivo das tarefas de *flocking* é que os agentes se movam juntos mantendo uma formação específica.

Para esta classe de aplicações, são propostos SMRs de várias topologias, sendo mais comuns as que consideram sistemas fortemente coordenados [25]. Contudo, também há trabalhos em que são propostos SMRs não-coordenados, como em [7].

2.4.5 Futebol de robôs

Quando são consideradas aplicações em futebol de robôs, são trabalhados diversos tipos de tarefas, associadas às funções de defesa, toque de bola, chute a gol, etc. Assim, esta é, possivelmente, a classe de aplicações mais complexa dentre as tratadas com multi-robôs, apresentando problemas relacionados com comunicação, ambiente altamente dinâmico, tolerância à falta, dentre outros.

Devido a essa complexidade, geralmente são utilizados SMRs fortemente centralizados e deliberativos. Contudo, abordagens reativas também podem ser encontradas, como a proposta em [8].

2.5 CONCLUSÃO

Neste capítulo, foram apresentados o problema de coordenação de SMRs e uma taxonomia usada para classificá-los. Proposta em Iocchi *et al.* (2001)[25], essa taxonomia se baseia no uso de quatro dimensões (cooperação, conhecimento, orquestração e organização) para caracterizar um SMR. Na sequência, são discutidos os aspectos de reatividade e deliberação em robôs e SMRs. Por último, são apresentadas as cinco classes de tarefas apresentadas em [25], que se destacam por serem comumente utilizadas como cenário de teste em diversos trabalhos.

3 MÉTODOS PARA COORDENAÇÃO DE SMRS BASEADOS EM SED

As ideias gerais das principais abordagens, baseadas em SED, utilizadas para resolver o problema de coordenação de SMRs são apresentadas neste capítulo. Além disso, o método proposto em Piterman *et a.* (2006)[56] é discutido em detalhes, sendo apresentada a forma de modelagem e os algoritmos utilizados para obtenção do controlador. Na seção a seguir, são apresentados os principais conceitos e formalismos relacionados com a área de sistemas a eventos discretos.

3.1 FUNDAMENTOS DE SISTEMAS A EVENTOS DISCRETOS

Na Teoria Clássica de Controle (TCC), equações diferenciais em função do tempo são utilizadas para modelar um sistema. Contudo, existem alguns casos em que o seu comportamento não evolui pela passagem contínua de tempo, mas pela ocorrência de eventos, são os sistemas a eventos discretos.

Definição 3.1. *Um Sistema a Eventos Discretos (SED) é um sistema de estados discretos e dirigido por eventos, ou seja, sua evolução no espaço de estados depende completamente da ocorrência de eventos discretos e assíncronos sobre o tempo [13].*

De modo geral, os SEDs são caracterizados por terem um espaço de estados discreto e por suas transições ocorrerem de forma abrupta. Além disso, é comum que elas estejam associadas aos eventos que ocorrem numa planta que se deseja descrever, sejam eles controláveis (sistema) ou não (ambiente) [13, 19]. Assim, o conjunto ou *alfabeto* de eventos de uma planta é dado por $\Sigma = \Sigma_c \cup \Sigma_u$, onde Σ_c é o *alfabeto* de eventos controláveis e Σ_u o dos não-controláveis.

Outro aspecto que merece destaque é a observabilidade dos eventos. Diz-se que um evento não observável é aquele sobre o qual o sistema não possui qualquer informação. Por sua vez, se o sistema sempre pode perceber a ocorrência de um determinado evento, ele é dito observável [13]. Neste trabalho, ainda é definido um terceiro tipo, um evento parcialmente observável.

Definição 3.2. *Um evento parcialmente observável é aquele que o sistema só é capaz de perceber em determinadas situações.*

Este tipo de evento pode representar, por exemplo, o mecanismo de detecção de uma porta em sistemas cuja informação sobre o ambiente é obtida

a partir dos sensores dos robôs que os compõem. Nestes casos, o sistema só consegue "ver" se ela está fechada quando um dos seus agentes está na proximidade dela.

3.1.1 Formalismo para descrição de SEDs

Existem vários formalismos que podem ser utilizados para representar SEDs, tais como redes de Petri, autômatos, expressões regulares, cadeias de Markov, etc. Contudo, devido ao fato dos autômatos serem uma das formas mais concisas e intuitivas de se representar um SED, este será descrito a seguir.

Autômatos

O conceito de autômato vem sendo estudado a muitas décadas e, durante esse tempo, diversas variações desse formalismo foram criadas, tais como o autômato de Büchi [65], estrutura de Kripke [35], autômatos temporizados [2, 3], etc. Contudo, para fins de definição do conceito, um autômato pode ser descrito pela tupla mostrada na equação 3.1.

$$\mathcal{A} = (Q, \Sigma, f, q_0, Q_m) \tag{3.1}$$

onde Q é o conjunto finito de estados do autômato, Σ é o *alfabeto* de eventos, $f : Q \times \Sigma \rightarrow Q$ é a função de transição, q_0 é o estado inicial do autômato e $Q_m \subseteq Q$ é o conjunto de estados marcados. Vale destacar que a função de transição define, para cada transição do autômato, os estados de partida e de destino e o evento que a dispara[13, 19].

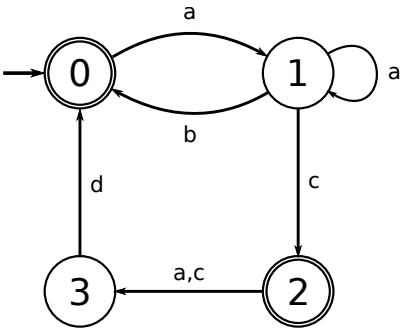


Figura 2: Exemplo de um autômato

Exemplo 3.1. *Na figura 2 é apresentado um autômato bastante simples, no qual estão representados todos os parâmetros mostrados na equação 3.1. Abaixo é apresentada a descrição de cada elemento da tupla $\mathcal{A}$.*

- $Q = \{0,1,2,3\}$.
- $\Sigma = \{a,b,c,d\}$.
- *A função de transição f é:* $f(0,a) = 1$, $f(1,a) = 1$, $f(1,b) = 0$, $f(1,c) = 2$, $f(2,a) = 3$, $f(2,c) = 3$ *e* $f(3,d) = 0$.
- $q_0 = 0$;
- $Q_m = \{0,2\}$

Vale destacar que o conjunto de estados marcados Q_m é composto por todos os estados que representam "estados finais" do sistema. Assim, estes podem ser utilizados para indicar que uma tarefa foi concluída ou que o sistema está numa configuração segura, dentre outros aspectos de interesse.

Como foi comentado nesta seção, existem diversas variações de autômatos, contudo não é objetivo deste trabalho discutir os detalhes de cada uma delas. Todavia, vale destacar a ideia geral de um autômato temporizado, empregado na modelagem de sistemas com restrições temporais.

Para esta variação, além dos conceitos já definidos, também são incluídas as ideias de relógios (*clocks*), guardas (*guards*) e invariantes para representar as características temporais da planta. Neste sentido, o *clocks* são utilizados para indicar a passagem de tempo, os *guards* para definir uma condição temporal para ativação de uma transição e os invariantes para definir uma restrição de permanência num estado [3, 1].

3.1.2 Verificação Lógica

Existem várias abordagens que podem ser utilizadas para validar o modelo de uma planta, tais como métodos baseados em simulação, verificação comportamental e verificação lógica[28]. Contudo, como os métodos baseados em verificação formal, apresentados neste trabalho, utilizam apenas verificação lógica, este tipo é o único discutido neste capítulo.

Na verificação, a validação do modelo do sistema é feita através da análise de determinadas propriedades. No caso da verificação lógica, as propriedades que se deseja verificar são descritas usando lógica temporal (LT) e é realizada uma análise sobre o modelo do sistema (*model checking*) para confirmar se elas são satisfeitas[16].

É importante destacar que, neste método de verificação, existe a possibilidade de explosão de estados, o que pode tornar a validação de algumas propriedades computacionalmente custosa. Dentre outros fatores, isso contribuiu para a criação de diversas linguagens baseadas em LT, no intuito de limitar sua expressibilidade e, assim, diminuir a complexidade da verificação de propriedades descritas usando cada uma destas linguagens.

Como contrapartida da redução da expressibilidade, pode ocorrer de não ser possível representar algumas propriedades. Dentre as linguagens mais utilizadas estão a *Propositional Linear Temporal Logic* (PLTL ou LTL) [58], a *Computation Tree Logic* (CTL) [17] e μ-calculus. A seguir são discutidas as linguagens LTL e μ-calculus, utilizadas em alguns métodos apresentados no restante deste capítulo.

Linguagem LTL

Seja A_G um autômato de estados finitos que descreve um sistema G e $\mathcal{P}$ um conjunto com todas as proposições realizadas sobre G. Quando se utiliza a linguagem LTL para descrever propriedades que devem ser verificadas em um sistema G, os estados de A_S devem fornecer uma interpretação a respeito das proposições em $\mathcal{P}$ [30].

Seja também $\sigma : q_0, q_1, \cdots$ a sequência de estados que representa uma execução de A_G e $p \in \mathcal{P}$ uma fórmula lógico-temporal. $(\sigma, j) \models p$ denota que p é satisfeito em σ no passo $j \geq 0$ [30, 39]. Uma fórmula p pode ser descrita em LTL usando operadores booleanos $(\neg, \vee)$ e os temporais *Next*($\bigcirc$), *Always*($\Box$), *Eventually*($\Diamond$), *Until*($\mathcal{U}$), *Unless*($\mathcal{W}$), *Previous*($\ominus$) e *Since*($\mathcal{S}$), definidos a seguir.

$$
\begin{array}{lcl}
(\sigma, j) \models p & \Leftrightarrow & s_j \models p \\
(\sigma, j) \models \neg p & \Leftrightarrow & (\sigma, j) \not\models p \\
(\sigma, j) \models p \vee q & \Leftrightarrow & (\sigma, j) \models p \text{ ou } (\sigma, j) \models q \\
(\sigma, j) \models \bigcirc p & \Leftrightarrow & (\sigma, j+1) \models p \\
(\sigma, j) \models \Box p & \Leftrightarrow & \text{para todo } k \geq j,\ (\sigma, k) \models p \\
(\sigma, j) \models \Diamond p & \Leftrightarrow & \text{para algum } k \geq j,\ (\sigma, k) \models p \\
(\sigma, j) \models p \mathcal{U} q & \Leftrightarrow & \text{para algum } k \geq j,\ (\sigma, k) \models q \text{ e, para todo } i \text{ tal que} \\
 & & j \leq i < k,\ (\sigma, i) \models p \\
(\sigma, j) \models p \mathcal{W} q & \Leftrightarrow & (\sigma, j) \models p \mathcal{U} q \text{ ou } (\sigma, j) \models \Box p \\
(\sigma, j) \models \ominus p & \Leftrightarrow & j > 0 \text{ e } (\sigma, j-1) \models p \\
(\sigma, j) \models p \mathcal{S} q & \Leftrightarrow & \text{para algum } k \leq j,\ (\sigma, k) \models q \text{ e, para todo } i \text{ tal que} \\
 & & j \geq i > k,\ (\sigma, i) \models p
\end{array}
$$

É importante ressaltar que os outros operadores booleanos ($\wedge, \Rightarrow e \Leftrightarrow$) podem ser construídos a partir de associações de $\neg$ e $\vee$. Da mesma forma, ou-

tros operadores temporais podem ser construídos a partir dos definidos acima, tais como o *infinitely often* ou *sempre no futuro*($\Box\Diamond$) e o *eventually permanently* ou *no futuro sempre*($\Diamond\Box$) [30, 39].

μ-calculus

A linguagem μ-calculus se baseia no conceito de ponto fixo[1] para representar as propriedades de um sistema. Neste sentido, os dois operadores principais desta linguagem são o menor (μ) e o maior (ν) ponto fixo. Abaixo é apresentada a sintaxe básica desta linguagem.

Seja um sistema de transições rotuladas $K = (Q, R, l)$, onde Q é o conjunto de estados, R as relações de transição entre eles, $l : Q \to 2^{\mathcal{P}}$ um mapeamento de cada estado ao conjunto de proposições atômicas satisfeitas nele e $\mathcal{P}$ o conjunto de todas as proposições atômicas. Dada uma interpretação $v : \mathcal{V} \to 2^{Q}$, onde $\mathcal{V}$ é uma variável, o conjunto de estados $[[\varphi]]^{v}_{K}$ é definido indutivamente como segue [41]:

$$
\begin{array}{lcl}
[[p]]^{v}_{K} & = & \{x \in Q \mid p \in l(x)\} \\
[[\neg p]]^{v}_{K} & = & \{x \in Q \mid p \notin l(x)\} \\
[[\varphi \vee \psi]]^{v}_{K} & = & [[\varphi]]^{v}_{K} \cup [[\psi]]^{v}_{K} \\
[[Z]]^{v}_{K} & = & v(Z) \\
[[\Box\varphi]]^{v}_{K} & = & \{x \in Q \mid \forall x' \in Q, (x,x') \in R \to x' \in [[\varphi]]^{v}_{K}\} \\
[[\mu Z.\varphi]]^{v}_{K} & = & \bigcap\{T \subseteq Q \mid [[\varphi]]^{v[Z\mapsto T]}_{K} \subseteq T\} \\
[[\nu Z.\varphi]]^{v}_{K} & = & \bigcup\{T \subseteq Q \mid T \subseteq [[\varphi]]^{v[Z\mapsto T]}_{K}\}
\end{array}
$$

Vale destacar que uma fórmula $[[\varphi]]^{v[Z\mapsto T]}_{K}$ indica que deve ser considerado que a variável Z recebe o conjunto T de estados. Por exemplo, para $\varphi = (p \vee \Box Z)$, atribui-se inicialmente o conjunto T à variável Z.

A linguagem μ-calculus possui grande expressividade, de modo que todos os operadores definidos usando CTL*, que é mais abrangente que LTL e CTL, podem ser codificados em μ-calculus. Por exemplo, em [56] é apresentada uma forma para transformar especificações em LTL em fórmulas em μ-calculus.

[1] No contexto de SED, o calculo do ponto fixo de uma função f pode ser entendido como a obtenção de um conjunto de estados Q, tal que, se f for aplicada em Q, o resultado é o próprio Q

3.2 CONTROLE SUPERVISÓRIO

Como é discutido em Cury (2001)[19], um sistema a ser controlado (*planta*) é composto, na maioria dos casos, por um conjunto de subsistemas (equipamentos, processos, ...). Contudo, cada subsistema tem um comportamento característico que deve ser restringido, de modo que o sistema global apresente o comportamento esperado.

Assim, a uma *planta G*, obtida a partir da composição de todos os subsistemas, é associada uma *linguagem gerada* ($\mathcal{L}(G)$) e uma *linguagem marcada* ($\mathcal{L}_m(G)$), que representam o conjunto de cadeias de eventos possíveis para a planta G e o conjunto de cadeias de eventos que levam a estados marcados, respectivamente.

Como a planta é formada pela composição dos comportamentos individuais de cada subsistemas, as linguagens $\mathcal{L}(G)$ e $\mathcal{L}_m(G)$ podem apresentar cadeias indesejáveis, por levar a estados em que ocorre *deadlock* ou *livelock* ou que representam situações inadmissíveis (colisão entre agentes móveis, etc.). Neste sentido, é definida uma *especificação E* contendo o conjunto de restrições associadas à planta.

Neste contexto, o problema de controle supervisório pode ser definido, informalmente, como a obtenção de um supervisor S que interaja com a planta G (figura 3), evitando a ocorrência de cadeias de eventos indesejadas.

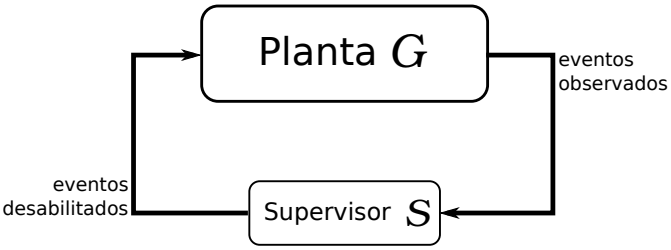


Figura 3: SED em malha fechada [19].

Para isto, o sistema S observa os eventos ocorridos na planta e desabilita todos os eventos controláveis que possam levar, a partir do estado atual, à ocorrência de cadeias indesejáveis. É importante ressaltar que este tipo de controle é dito permissível, uma vez que impede apenas a ocorrência dos eventos que levariam a planta a "maus estados".

Como a Teoria de Controle Supervisório (TCS) já foi amplamente discutida e apresentada em vários trabalhos, não serão mostrados detalhes sobre a obtenção da planta e do supervisor. Para mais informações sobre a TCS,

sugere-se a leitura das referências [19, 46, 55, 62, 61, 68], onde são definidos os conceitos da TCS e apresentadas aplicações na área de robótica móvel baseadas nessa teoria.

É muito importante destacar também que este trabalho foi desenvolvido em paralelo ao apresentado em Pavei (2011)[54, 55], onde o problema de planejamento de atividades e coordenação de SMRs foi estudado sob a ótica da teoria de controle supervisório, além da de autômato-jogo.

3.3 PLANEJAMENTO DE TAREFAS BASEADO EM *MODEL CHECKING*

Para resolver o problema de coordenação em SMRs, também pode ser feito uso de técnicas de *model checking*. Nesta abordagem, após a descrição do modelo da planta, as especificações do sistema são definidas em lógica temporal. Na sequência, o controlador é obtido a partir da verificação das especificações, utilizando um *observador* ou um *contra-exemplo*, sendo este último o mais comum.

A seguir são apresentados alguns aspectos sobre cada um destes passos. Contudo, assim como ocorreu na descrição da TCS, não serão mostrados detalhes sobre os métodos baseados em *model checking*. Para mais informações sugere-se a leitura dos trabalhos apresentados em [4, 12, 21, 22, 34, 60] e suas referências.

Vale destacar ainda que, nesse grupo de pesquisa, também foram desenvolvidos trabalhos sobre o planejamento de tarefas baseado em *model checking* [63, 64], onde foram tratados sistemas multi-robôs com especificações de alcançabilidade e operando em ambientes estáticos.

Descrição da planta

Dependendo do método, o modelo da planta pode ser descrito através de diferentes tipos de formalismos de SED, tais como autômatos lógicos (não-temporizados), autômatos temporizados, redes de petri, etc.

Vale destacar ainda que, apesar do formalismo mais utilizado para descrição das plantas ser o de autômatos temporizados [4, 21, 22, 60], existem vários trabalhos onde Redes de Petri [12, 31] ou mesmo asserções em LTL [29, 34, 56] são utilizados para descrever as plantas.

Descrição das especificações

A descrição das especificações pode ser feita utilizando diferentes linguagens de lógica temporal, sendo que cada uma tem suas vantagens e desvantagens, podendo ser mais ou menos adaptadas a descrever os vários tipos de especificações. Todavia, de modo geral, se o objetivo é uma verificação exaustiva, a linguagem CTL é mais adequada, pois evita uma explosão combinacional.

Por outro lado, quando o objetivo é especificar alguma propriedade ou mesmo quando a verificação não é exaustiva, LTL é mais adequada. Por exemplo, fórmulas de alcançabilidade podem ser facilmente descritas em CTL, mas não podem ser descritas diretamente em LTL. Já propriedades de vivacidade e segurança são facilmente expressas em LTL .

Vale destacar ainda que, apesar de serem mencionadas apenas as linguagens LTL e CTL, outras também podem ser utilizadas, como μ-calculus. Por exemplo, nos trabalhos apresentados em [29, 56], a especificação, apesar de descrita num fragmento de LTL, é convertida para μ-calculus antes da execução dos algoritmos.

Obtenção do Plano

Como já foi mencionado, os controladores podem ser obtidos como um observador ou um contra-exemplo. Na primeira, conhecida como geração de *observadores*, o controlador é uma computação em que a especificação é satisfeita, ou seja, uma sequência de eventos que levam ao cumprimento da especificação. Contudo, a maioria das ferramentas computacionais de *model checking* não suportam a geração de observadores [22]. Para estas ferramentas, o controlador pode ser obtido através da solução do problema dual: geração de contra-exemplos.

Na geração de contra-exemplos, é verificada a negação de uma propriedade (ou especificação) esperada. Assim, caso a propriedade seja válida, é gerado um contra-exemplo indicando por que a negação da propriedade não é satisfeita. Este contra-exemplo é o próprio controlador. Por exemplo, caso o objetivo seja alcançar uma determinada posição no espaço de trabalho, a especificação verificada é que não é possível alcançar tal posição (negação da propriedade). Se de fato for possível chegar à posição, é gerado um contra-exemplo indicado como o sistema pode deslocar seus agentes a fim de alcançá-la.

Vale destacar ainda que, para a quase totalidade das aplicações, existem várias possibilidades de controladores. Assim, é comum que se utilize critérios de escolha como o número de passos que os agentes devem realizar, a quantidade de estados do controlador ou o tempo de execução, dentre outros, para determinar o controlador.

3.4 TEORIA DE JOGOS

A teoria de jogos pode ser definida como a parte da matemática aplicada que estuda modelos de conflitos e cooperação entre decisores inteligentes e racionais[48]. Desse modo, essa teoria fornece um ferramental matemático para análise de casos onde existem diversos agentes independentes, cujas ações afetam o comportamento dos outros[24, 48].

Assim, a teoria de jogos vem sendo estudada e aplicada nas mais diferentes áreas da ciência. Por exemplo, para desenvolvimento e análise de modelos econômicos[33], análise de relações internacionais e políticas econômicas [47] e estudo de processos evolucionários (tais como evolução e competição entre espécies)[66]. Contudo, no contexto do problema de coordenação de SMRs, vale destacar a parte da teoria de jogos denominada Combinatória[18].

Na teoria combinatória, o estudo é focado na "jogada" ótima de um jogo com dois participantes, no qual cada jogador possui um turno para efetuar suas ações, de forma similar a um jogo de xadrez [24]. No contexto do problema de coordenação de SMRs, um jogador representa o sistema e o outro o ambiente, no qual são agrupados todos os elementos dos quais o sistema não tem controle.

Vale destacar ainda que, como o próprio nome sugere, na teoria combinatória a jogada ótima é obtida a partir da análise de todas as combinações de jogadas possíveis, não existindo o aspecto de aleatoriedade (como ocorre na teoria clássica de jogos).

Com relação ao problema de coordenação de SMRs, existem vários trabalhos baseados na teoria de jogos em que métodos para modelagem de sistemas e obtenção de um controlador são propostos. De forma geral, estas abordagens se baseiam em considerar os robôs como parte do sistema e os outros elementos dinâmicos presentes no espaço de trabalho como ambiente. É importante ressaltar também que, quando são considerados sistemas distribuídos, os controladores de cada robô podem ser obtidos considerando os outros agentes do sistema como parte do ambiente.

Uma vez obtido o modelo do sistema, é descrita a especificação do sistema e o controlador é gerado, quando possível, de forma que a especificação seja sempre satisfeita independentemente das ações do ambiente. Vale destacar ainda que, as principais diferenças entre os métodos estão no formalismo utilizado para descrever a planta, na capacidade de expressão da linguagem usada para definir as especificações e na possibilidade de exprimir cooperação entre ambiente e sistema.

No método proposto em Maler *et. al* (1995)[38], a planta é descrita por autômatos temporizados, sendo as transições rotuladas com os eventos

do alfabeto $\Sigma = \Sigma_S \cup \{e\}$, onde Σ_S é um alfabeto associado ao sistema e e é um evento utilizado para representar todas as ações associadas ao ambiente. Já a especificação pode ser descrita através de fórmulas LTL de alcançabilidade ($\Diamond F$), segurança ($\Box F$), "sempre no futuro" ($\Box\Diamond F$), "no futuro sempre" ($\Diamond\Box F$) e condição de Rabin, onde F é uma proposição. Para mais detalhes sugere-se a leitura do próprio trabalho, apresentado em [38].

Uma extensão do trabalho descrito em [38] é proposta em Asarin *et al.* (1998)[5]. Neste, é atribuído um alfabeto de eventos Σ_A ao ambiente ($\Sigma = \Sigma_S \cup \Sigma_A$), facilitando a modelagem das aplicações. Contudo, consideram-se apenas fórmulas de segurança em LTL para descrever a especificação do sistema.

Com base no formalismo de autômato-jogo temporizado, proposto e extendido nestes trabalhos ([38] e [5]), foi proposta e implementada uma ferramenta para modelagem, simulação e obtenção de controladores, o UPPAAL-TIGA (UPPAAL-*Timed Games*)[9, 11]. Vale destacar que esta ferramenta computacional implementa um algoritmo *on-the-fly* para síntese estratégias vencedoras[2][15, 37]. Para mais detalhes sobre o algoritmo *on-the-fly*, sugere-se a leitura dos trabalhos descritos em [14, 15] e no capítulo 5 de [54].

Outro aspecto importante sobre o UPPAAL-TIGA, é que as especificações do sistema são definidas em *Timed Computation Tree Logic* (TCTL). No manual da ferramenta [10], são definidos os três tipos de especificações que podem ser descritas: Alcançabilidade (*pure reachability*), Segurança (*pure safety*) e alcançabilidade com segurança (*reachability with avoidance*), sendo que não é possível aninhar fórmulas.

Com base no trabalho apresentado posteriormente em David *et al.* (2008) [20], foi introduzida na ferramenta a possibilidade de geração de estratégias cooperativas. Contudo, é muito importante observar que a forma como a cooperação ocorre não é descrita pelo projetista, sendo definida pelo próprio algoritmo durante a obtenção do controlador. Mais detalhes podem ser obtidos em [10] e [54].

Vale destacar que nos trabalhos apresentados em Pavei (2011) [54, 55], além da teoria de controle supervisório, também foi utilizada a ferramenta UPPAAL-TIGA para obtenção de estratégias para coordenação de SMRs. Em ambas as abordagens (TCS e autômato-jogo), foram considerados sistemas centralizados e distribuídos com especificações de alcançabilidade e inseridos em ambientes compostos por portas, das quais o sistema não tinha controle.

É importante ressaltar que foi necessário considerar cooperação para que fosse possível obter estratégias de coordenação. Contudo, a ferramenta não permite declarar a forma como a cooperação entre ambiente e sistema

[2]Uma estratégia vencedora é uma execução onde a especificação do sistema sempre é satisfeita

ocorre. Além disso, como só pode ocorrer um evento em cada transição, as estratégias geradas não consideram a movimentação simultânea de robôs do sistema, sendo que em cada passo apenas um agente pode se mover.

Outro método bastante promissor é o proposto em Piterman *et al.* (2006)[56]. Neste, o sistema e o ambiente são descritos separadamente, sendo possível definir em detalhes a dinâmica dos elementos que compõem o ambiente, o que não ocorre no primeiro método, proposto em [38] e extendido em [5]. Além disso, também é possível realizar hipóteses sobre o ambiente. Desta forma, é possível descrever como o ambiente pode cooperar com o sistema.

Em Kress-Gazit *et al.* (2009) [34], o método proposto em [56] foi utilizado na solução do problema de coordenação de SMRs, tendo sido obtidos bons resultados. A seguir, o método proposto em Piterman *et al.* (2006) [56], que foi o objeto de estudo deste trabalho, é apresentado em detalhes.

3.5 PITERMAN *ET. AL* (2006) [56]

No método proposto em Piterman *et al.* (2006)[56], há uma mudança significativa na forma de representar a planta, de modo que, ao invés de utilizar uma representação onde as transições estão associadas a ocorrência de algum evento, é definida uma "Estrutura de Jogo" (*Game Structure*), onde cada transição está associada à mudança no valor de uma (ou mais) variável de estado.

Uma estrutura de jogo é definida pela tupla $G : \langle V, \mathcal{X}, \mathcal{Y}, \Theta, \rho_e, \rho_s, \varphi \rangle$. A seguir são descritos cada um dos parâmetros da estrutura G.

- $V = \{u_1, \ldots, u_n\}$ é um conjunto finito de variáveis de estado. Cada estado do autômato é uma representação dessas variáveis, de forma que para cada $u \in V$ é associado um valor $s[u]$, onde $s \in \Sigma$ é um estado e Σ é o espaço de estados;
- $\mathcal{X} \subseteq V$ é o conjunto de variáveis de entrada, associadas ao ambiente. D_X corresponde ao conjunto de todas as combinações de valores possíveis para as variáveis em $\mathcal{X}$;
- $\mathcal{Y} \subseteq V/\mathcal{X}$ é o conjunto de variáveis de saída, associadas ao sistema. D_Y corresponde ao conjunto de todas as combinações de valores possíveis para as variáveis em $\mathcal{Y}$;
- Θ é uma asserção que caracteriza os estados iniciais. Um estado é dito inicial se ele satisfaz Θ;
- $\rho_e(\mathcal{X}, \mathcal{Y}, \mathcal{X}')$ é a relação de transição do ambiente. É uma asserção que relaciona um estado aos próximos valores de entrada $\xi' \in D_X$ possíveis. Assim, $\xi' \in D_X$ é uma possível entrada no estado s, caso $(s, \xi') \vDash \rho_e(\mathcal{X}, \mathcal{Y}, \mathcal{X}')$.
- $\rho_s(\mathcal{X}, \mathcal{Y}, \mathcal{X}', \mathcal{Y}')$ é a relação de transições do sistema. É uma asserção que relaciona um estado s aos próximos valores de saída $\eta' \in D_Y$ possíveis, dada uma entrada $\xi' \in D_X$. Assim, $\eta' \in D_Y$ é uma possível saída no estado s, caso $(s, \xi', \eta') \vDash \rho_s(\mathcal{X}, \mathcal{Y}, \mathcal{X}', \mathcal{Y}')$.
- φ é a condição de vitória, dada por uma fórmula LTL.

No método apresentado em [56], são consideradas especificações em LTL na forma $\varphi^e \rightarrow \varphi^s$, onde φ^e e φ^s são dadas por $\varphi^\alpha = \varphi_i^\alpha \wedge \varphi_t^\alpha \wedge \varphi_g^\alpha$, com $\alpha \in \{e, s\}$. Essa restrição da linguagem LTL é feita com o intuito de diminuir o tempo de verificação de existência de solução, que é 2EXPTIME-completo [57], considerando toda a linguagem.

Em relação aos termos que compõem φ^e e φ^s, φ_i^e e φ_i^s correspondem as condições iniciais do ambiente e do sistema, respectivamente. Assim, temos que $\Theta = \varphi_i^e \wedge \varphi_i^s$. Por sua vez, φ_t^e e φ_t^s correspondem às dinâmicas do ambiente e do sistema, de modo que $\varphi_t^e \equiv \rho_e$ e $\varphi_t^s \equiv \rho_s$. Por último, temos que $\varphi = \varphi_g^e \rightarrow \varphi_g^s$, deste modo, a condição de vitória é dada pela fórmula 3.2.

$$\varphi = \bigwedge_{i=1}^{m} \square\Diamond p_i \rightarrow \bigwedge_{j=1}^{n} \square\Diamond q_j \tag{3.2}$$

onde m e n são os números de proposições a serem satisfeitas pelo ambiente e pelo sistema, respectivamente, p_i representa a i-ésima proposição a ser satisfeita pelo ambiente "sempre no futuro". Analogamente, q_j é a j-ésima proposição a ser atendida pelo sistema.

É muito importante destacar que, ao considerar especificações no formato mostrado na equação 3.2, a verificação de existência de solução e síntese do controlador são realizadas em tempo $O(|Q|^3)$, onde $|Q|$ corresponde à quantidadede elementos no espaço de estados da estrutura de jogo[29].

Dada uma estrutura de jogo $G : \langle V, \mathcal{X}, \mathcal{Y}, \rho_e, \rho_s, \varphi \rangle$, é utilizada uma abordagem que transforma a condição de vitória, em LTL, para um formato em μ-calculus [29, 56]. Para poder definir a condição de vitória (equação 3.2) em μ-calculus, são definidos alguns operadores de μ-calculus sobre estrutura de jogos com base nos resultados apresentados em Kozen *et. al* (1983) [32].

3.5.1 μ-calculus sobre estruturas de jogos

Seja uma estrutura de jogo $G : \langle V, \mathcal{X}, \mathcal{Y}, \rho_e, \rho_s, \varphi \rangle$ e uma proposição atômica v. Seja também $V_{ar} = \{X, Y, \cdots\}$ um conjunto de variáveis relacionais. As fórmulas em μ-calculus são construídas como segue.

$$\phi \ ::= \ v \,|\, \neg v \,|\, X \,|\, \phi_1 \vee \phi_2 \,|\, \phi_1 \wedge \phi_2 \,|\, \odot \phi \,|\, \boxdot \phi \,|\, \mu X \phi \,|\, \nu X \phi$$

Uma fórmula ϕ é interpretada como um conjunto de G-estados[3] nos quais ϕ é válida. Este conjunto de estados é definido como $[[\phi]]_G^e$, onde G é uma estrutura de jogo e $e : V_{ar} \rightarrow 2^{\Sigma}$ é um *ambiente*. Vale ressaltar que o ambiente atribui a uma variável X um subconjunto de Q e que se denota por $e[X \leftarrow S]$ o ambiente tal que $e[X \leftarrow S](X) = S$ e $e[X \leftarrow S](Y) = e(Y)$. O conjunto $[[\phi]]_G^e$ é definido, por indução, como segue [29, 49, 56].

[3] Estados pertencentes a estrutura de jogo G

$$[[v]]^e_G = \{s \in \Sigma \,|\, s[v] = 1\}$$

$$[[\neg v]]^e_G = \{s \in \Sigma \,|\, s[v] = 0\}$$

$$[[X]]^e_G = e(X)$$

$$[[\phi \vee \psi]]^e_G = [[\phi]]^e_G \cup [[\psi]]^e_G$$

$$[[\phi \wedge \psi]]^e_G = [[\phi]]^e_G \cap [[\psi]]^e_G$$

$$[[\odot\phi]]^e_G = \left\{ s \in \Sigma \;\middle|\; \begin{array}{l} \forall x', (s,x') \models \rho_e \rightarrow \exists y' \text{ tal que } (s,x',y') \models \rho_s \\ \text{e } (x',y') \in [[\phi]]^e_G \end{array} \right\}$$

Um estado s é incluído em $[[\odot\phi]]^e_G$ se o sistema pode forçar uma jogada que alcance um estado em $[[\phi]]^e_G$. Ou seja, independentemente de como o ambiente se "move" a partir de s, o sistema pode escolher uma ação que leve a planta a um dos estados em $[[\phi]]^e_G$ [56].

$$[[\boxdot\phi]]^e_G = \left\{ s \in \Sigma \;\middle|\; \begin{array}{l} \exists x' \text{ tal que } (s,x') \models \rho_e \\ \text{e } \forall y', (s,x',y') \models \rho_s \rightarrow (x',y') \in [[\phi]]^e_G \end{array} \right\}$$

Um estado s é incluído em $[[\boxdot\phi]]^e_G$ se o ambiente pode forçar uma jogada que alcance um estado em $[[\phi]]^e_G$. Como o ambiente "joga" primeiro, ele escolher uma entrada $x' \in X$ tal que, para qualquer escolha do sistema, o sucessor (x',y') está em $[[\phi]]^e_G$.

$[[\mu X\phi]]^e_G = \cup_i S_i$, onde $S_0 = \emptyset$ e $S_{i+1} = [[\phi]]^{e[X \leftarrow S_i]}_G$

Um estado s pertence a $[[\mu X\phi]]^e_G$ se ele está contido num conjunto S_i, tal que $S_i = [[\phi]]^{e[X \leftarrow S_{i-1}]}_G$. Vale destacar que os conjuntos S_i são calculados considerando $X = S_{i-1}$ e até que $S_{i+1} = S_i$.

$[[\nu X\phi]]^e_G = \cap_i S_i$, onde $S_0 = \Sigma$ e $S_{i+1} = [[\phi]]^{e[X \leftarrow S_i]}_G$

Um estado s pertence a $[[\nu X\phi]]^e_G$ se ele está contido em todos os conjuntos S_i, tal que $S_i = [[\phi]]^{e[X \leftarrow S_{i-1}]}_G$.

3.5.2 Notação vetorial da condição de vitória

Dado que X, Y e Z são variáveis e com base nos operadores definidos anteriormente, a condição de vitória (equação 3.2) pode ser reescrita como mostrado na equação 3.3.

$$\varphi = \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y \left(\bigvee_{i=1}^{m} \nu X (q_1 \wedge \odot Z_2 \vee \odot Y \vee \neg p_i \wedge \odot X) \right) \\ \mu Y \left(\bigvee_{i=1}^{m} \nu X (q_2 \wedge \odot Z_3 \vee \odot Y \vee \neg p_i \wedge \odot X) \right) \\ \vdots \\ \vdots \\ \mu Y \left(\bigvee_{i=1}^{m} \nu X (q_n \wedge \odot Z_1 \vee \odot Y \vee \neg p_i \wedge \odot X) \right) \end{bmatrix} \tag{3.3}$$

Na equação 3.3, cada linha é responsável por verificar se uma das n proposições do sistema, que estão associadas a realização dos seus objetivos, é satisfeita. Neste sentido, a equação 3.4 é responsável por verificar se é possível atender a j-ésima proposição do sistema (q_j).

$$\varphi_j = \nu Z_j \left(\mu Y \left(\bigvee_{i=1}^{m} \nu X (q_j \wedge \odot Z_{j\oplus 1} \vee \odot Y \vee \neg p_i \wedge \odot X) \right) \right) \tag{3.4}$$

onde $j \oplus 1 = (j \, mod \, n) + 1$ e representa o resto da divisão de j por n, mais 1.

Na equação 3.4, o maior *ponto fixo* $\nu X (q_j \wedge \odot Z_{j\oplus 1} \vee \odot Y \vee \neg p_i \wedge \odot X)$ caracteriza o conjunto de estados a partir dos quais o sistema pode forçar uma jogada para, ou ficar indefinidamente em estados que satisfazem $\neg p_i$ (o que viola o lado esquerdo da implicação mostrada na equação 3.2) ou, em um número finito de passos, alcançar um estado no conjunto $q_j \wedge \odot Z_{j\oplus 1} \vee \odot Y$. De forma geral, neste conjunto estão os estados onde a proposição q_j está sendo satisfeita ou a partir dos quais é possível alcançar um estado em que q_j é satisfeita ou em que uma das m proposições do ambiente (p_i) não é satisfeita.

Já o menor *ponto fixo* μY garante que as jogadas do sistema são finitas e terminam em um estado de $q_j \wedge \odot Z_{j\oplus 1}$. Por último, o *ponto fixo* νZ_j garante que, depois de visitar um estado que atende a especificação do sistema q_j, é possível passar a "perseguir" a especificação $q_{j\oplus 1}$, e assim sucessivamente.

Vale ressaltar que, as ferramentas computacionais que implementam os algoritmos propostos em [29, 56] codificam a condição de justiça, definida em LTL, no formato mostrado na equação 3.3.

3.5.3 Algoritmos

No método proposto em Piterman *et al.* (2006) [56], a obtenção do controlador é dividida em três partes: Verificar a existência de solução, realizar a síntese do controlador e sua minimização. A seguir são apresentados e discutidos os algoritmos responsáveis por realizar cada uma destas etapas.

Existência de solução

No código 3.5.1, é apresentada uma versão simplificada do algoritmo que implementa a equação 3.3 e, assim, permite verificar se existe uma estratégia vencedora.

```
1  Func ganhaJogo(m,n);
2    MaiorPontoFixo(z)
3      Para (j=1 até n)
4        Seja r := 1;
5        MenorPontoFixo(y)
6          Seja start := q(j) & cox(z) | cox(y);
7          Seja y := 0;
8          Para (i=0 até m)
9            MaiorPontoFixo(x <= z)
10             Seja x := start | !p(i) & cox(x)
11           Fim -- MaiorPontoFixo(x <= z)
12           Seja x[j][r][i] := x;
13           Seja y := y | x;
14         Fim -- Para (i=0 até m)
15         Seja y[j][r] := y;
16         Seja r := r + 1;
17       Fim -- MenorPontoFixo(y)
18       Seja z := y;
19       Seja maxr[j] := r - 1;
20     Fim -- Para (j=1 até n)
21   Fim -- MaiorPontoFixo(z)
22   Retorna z;
23 Fim -- Func ganhaJogo(m,n);
```

Código 3.5.1: Implementação da equação 3.3 [56]

No código 3.5.1, as variáveis $p(i)$ e $q(j)$ representam as especificações p_i e q_j, o operador $cox(V_{ar})$ representa $\odot V_{ar}$ e a conjunção, disjunção e negação são indicadas por &, | e !, respectivamente. Vale destacar ainda que, o ponto fixo $MaiorPontoFixo(x <= z)$ define que o valor inicial de x é z, ao invés do conjunto com todos os estados (Q). Já os conjuntos $x[j][r][i]$ e $y[j][r]$ são utilizados posteriormente para definir uma estratégia vencedora.

Seja φ_j (equação 3.4) a parte da condição de vitória que indica que, se todas as proposições assumidas sobre o ambiente ($p_1, \cdots, p_m$) são satisfeitas sempre no futuro, então a j-ésima proposição do sistema (q_j) também o será. Assim, φ_j é dado por:

$$\varphi_j = \left(\bigwedge_{i=1}^{n} \Box\Diamond p_i \right) \rightarrow \Box\Diamond q_j$$

Seja também Y_j um conjunto em que estão, inicialmente, apenas os estados em que a especificação φ_j está sendo cumprida. De uma forma geral, para cada especificação φ_j, este algoritmo é responsável por ir agregando dois tipos de estados a Y_j. O primeiro tipo corresponde aos estados a partir dos quais é possível alcançar os que já estão dentro Y_j.

Já o segundo tipo, corresponde aos estados nos quais uma das hipóteses não está sendo satisfeita e, além disso, o sistema pode forçar uma jogada no próximo passo, a partir destes estados, de modo a manter a hipótese falsa ou alcançar um estado já agregado a Y_j.

Caso os estados considerados iniciais sejam incorporados a este conjunto Y_j em algum passo da execução do algoritmo, isso significa que é possível criar um controlador que garanta a realização da especificação φ_j. Na figura 4 é mostrada uma representação bastante simplificada deste processo.

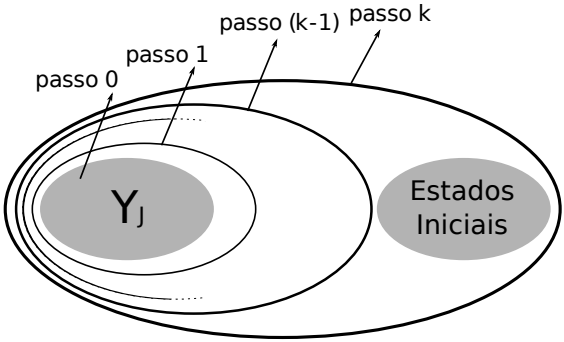


Figura 4: Verificação de existência de estratégia vencedora

Se, num passo do algoritmo, nenhum estado for agregado ao conjunto e todos os estados iniciais não tiverem sido incluídos, isso indica que não é possível gerar uma solução que garante o cumprimento das especificações. Quando isso ocorre, o algoritmo para de executar. Vale ressaltar que, basta que uma das especificações φ_j não seja cumprida para que não seja gerado um controlador.

Síntese de controlador

Para o método proposto em Piterman *et al.* (2006)[56], o controlador é obtido como um sistema de transição, similar a uma estrutura de jogo sem a condição de vitória, definido por $\mathcal{D} : \langle V_\mathcal{D}, \mathcal{X}, \mathcal{Y}_\mathcal{D}, \theta_\mathcal{D}, \rho \rangle$, onde $V_\mathcal{D} = V \cup \{jx\}$, com jx sendo uma variável utilizada para armazenar internamente qual proposição ou objetivo q_j do sistema está sendo tratada e que varia sobre $[1..n]$, e $\mathcal{Y}_\mathcal{D} = \mathcal{Y} \cup \{jx\}$, $\theta_\mathcal{D} = \theta \wedge (jx = 1)$.

Por último, a função de transição ρ é dada pela disjunção $\rho_1 \vee \rho_2 \vee \rho_3$, onde ρ_1 define as transições ativas em estados onde uma propriedade q_j é verdade e que levam a um estado onde a estratégia passa a tratar a especificação $\varphi_{j\oplus 1}$. Assim, as transições em ρ_1 são responsáveis por alterar o valor de jx.

Dado que em cada estado do controlador é definido, através do índice jx, o objetivo do sistema que está sendo tratado, as transições definidas em ρ_2 levam a estados mais próximos dos estados em que a proposição q_j é satisfeita. Já em ρ_3 são definidas as transições partindo de um estado em $s \in x[j][r][i]$ tal que $s \models \neg p_i$ e levam a um estado em $x[j][r][i]$. Assim, caso existam transições que possam manter a planta num estado em que uma das proposições (ou hipóteses) realizadas sobre o ambiente não é cumprida, indefinidamente, também é possível gerar uma estratégia vencedora.

No código 3.5.2 é apresentado o algoritmo responsável por obter a função de transição ρ do controlador. Neste código, as estruturas de repetição definidas entre as linhas [3..6], [8..15] e [17..26] são responsáveis por agregar a ρ as transições referentes a ρ_1, ρ_2 e ρ_3, respectivamente. Vale destacar ainda que a variável $trans12$ indica a conjunção entre ρ_e e ρ_s.

De forma bastante simplificada, o algoritmo apresentado no código 3.5.2 pode ser exemplificado pelo gráfico mostrado na figura 5

Na figura 5 é ilustrada a parte de um controlador referente ao cumprimento da especificação φ_j. Nesta é apresentado um sistema de transição indicando que, a partir de um conjunto de estados onde é feita a mudança de objetivo ($jx - 1 \mapsto jx$), o controlador evolui por diversos estados até chegar a um em que, ou a propriedade q_j é verdade, ou uma das hipóteses do ambiente $(p_1, \cdots, p_m)$ está sendo descumprida e o sistema pode tomar uma ação que mantenha esse hipótese inválida indefinidamente.

Vale destacar que, se nenhuma hipótese for descumprida, ao chegar num estado onde a proposição q_j é satisfeita, na próxima transição é alcançado um estado em que o objetivo tratado é novamente alterado ($jx \rightarrow jx+1$), a fim de atender a especificação do sistema $q_{j\oplus 1}$. Outro aspecto importante é que o controlador começa em um dos estados iniciais ($s \models \Theta$) e considerando o objetivo de atender q_1.

```
1  Para estrategia_simbol;
2    Seja trans := 0;
3    Para (j=1 até n)
4      Seja jp1 := (j mod n) + 1;
5      Seja trans := trans | (jx=j) & z & q(j) & trans12 &
6            next(z) & (next(jx)=jp1);
7    Fim -- Para (j=1 até n);
8
9    Para (j=1 até n)
10     Seja inferior := y[j][1];
11     Para (r=2 até maxr[j])
12       Seja trans := trans | (jx=j) & y[j][r] & !inferior &
13                  trans12 & next(inferior) & (next(jx)=j);
14       Seja inferior := inferior | y[j][r];
15     Fim -- Para (r=2 até maxr[j])
16   Fim -- Para (j=1 até n);
17
18   Para (j=1 até n)
19     Seja inferior := 0;
20     Para (r=2 até maxr[j])
21       Para (i=1 até m)
22         Seja trans := trans | (jx=j) & x[j][r][i] & !inferior &
23              !p(i) & trans12 & next(x[j][r][i]) & (next(jx)=j);
24         Seja inferior := inferior | x[j][r][i]
25       Fim -- Para (i=1 até m)
26     Fim -- Para (r=2 até maxr[j])
27   Fim -- Para (j=1 até n);
28
29 Fim --Para estrategia_simbol;
```

Código 3.5.2: Algoritmo proposto [56] em para obtenção da estratégia vencedora.

Minimização da estratégia

Para minimização do controlador gerado, é utilizada uma estratégia bastante simples, consistindo em eliminar estados redundantes. Os estados considerados redundantes são aqueles que diferem apenas pelo valor da variável interna jx. Assim, quando encontrados dois estados s e s' tais que $\rho(s,s')$, $s[\mathcal{X} \cup \mathcal{Y}] = s'[\mathcal{X} \cup \mathcal{Y}]$ e $s'[jx] = s[jx] \oplus 1$, o estado s e as transições partindo dele são removidos e as transições levando a ele são direcionadas para s'.

No código 3.5.3 é apresentada uma versão simplificada do algoritmo de redução, utilizado para minimizar a estratégia gerada.

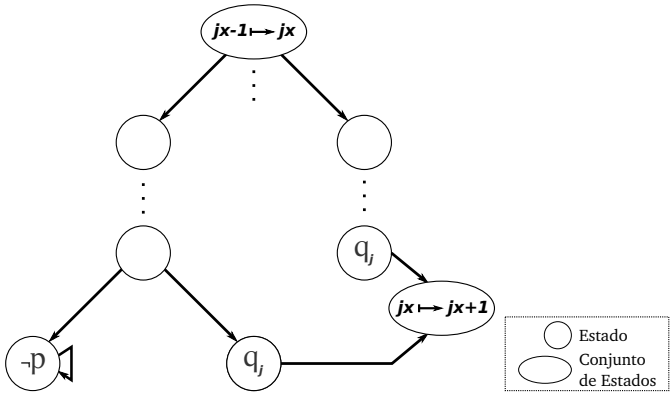


Figura 5: Representação parcial do processo de síntese do controlador

3.5.4 Exemplo

A fim de exemplificar o processo de modelagem e obtenção do controlador, utilizando o método apresentado em [56], é usado o caso mostrado na figura 6.

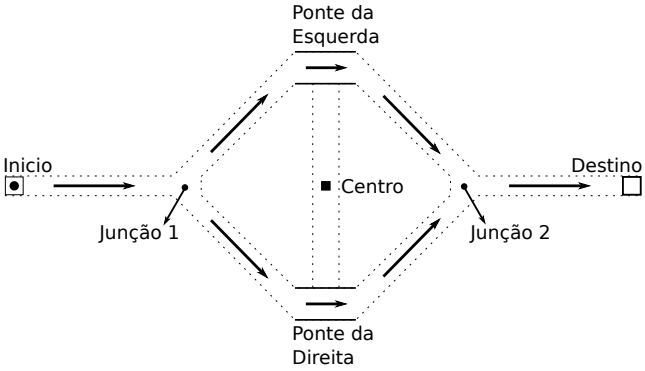


Figura 6: Caso utilizado para exemplificar o método proposto em [56]

Neste exemplo, um robô deve partir de *Inicio* e alcançar a posição *Destino*. Para isto, ele deve atravessar uma das pontes (da esquerda ou da direita). Ainda neste exemplo, existe um adversário que inicia na posição *Centro* e pode fechar uma das pontes, a fim de impedir a passagem do robô do sistema.

```
1  Para (j=1 até n)
2    Seja nextj := (j mod n)+1;
3    reduz(j,nextj);
4  Fim -- Para (j=1 até n)
5
6  Para (j=1 até n-1)
7    reduz(n,j);
8  Fim -- Para (j=1 até n-1)
9
10 Func reduz(j,k)
11   Seja inativo := trans & obseq & jx=j & next(jx)=k;
12   Seja estados := inativo paraAlgum next(V);
13   Seja add_trans := ((trans & next(estados) & next(jx)=j)
14                      paraAlgum jx) & next(jx)=k;
15   Seja rem_trans := next(estados) & next(jx)=j1 | estados & jx=j1;
16   Seja add_inicial := ((inicial & estados & jx=j1) paraAlgum jx)
17                       & jx=k;
18   Seja rem_inicial := estados & jx=j;
19   Seja trans := (trans & !rem_trans) | add_trans;
20   Seja inicial := (inicial & !rem_inicial) | add_inicial;
21   Retorna;
22 Fim -- Func reduz(j,k)
```

Código 3.5.3: Algoritmo proposto em [56] para minimização da estratégia vencedora.

Para modelar este exemplo foram utilizadas duas variáveis de estado: *robo* e *adv*, sendo a primeira associada ao sistema e a segunda ao ambiente. Além disso, foi adotado o seguinte espaço de valores para cada uma das variáveis.

$$\begin{array}{lcl} robo & : & \{I, J_1, P_E, P_D, J_2, D\} \\ adv & : & \{C, P_E, P_D\} \end{array}$$

onde os valores I, J_1, P_E, P_D, J_2 e D de *robo* representam que o agente está nas posições *Inicio*, *Junção 1*, *Ponte da Esquerda*, *Ponte da Direita*, *Junção 2* e *Destino*, respectivamente. Por sua vez, os valores C, P_E e P_D de *adv* indicam que o adversário está no centro, bloqueando a ponte da esquerda ou bloqueando a ponte da direita, respectivamente.

Vale destacar que, ao escolher uma das pontes para bloquear, o adversário não pode mais mudar sua opção. Assim, a condição inicial e dinâmica do sistema e do ambiente são definidas como mostrado nas equações 3.5 e 3.6.

$$\varphi_i^{robo} : (robo = I)$$
$$\varphi_t^{robo} : \begin{cases} (robo = I) \rightarrow \bigcirc [(robo = I) \vee (robo = J_1)] \\ \bigwedge (robo = J_1) \rightarrow \bigcirc [(robo = I) \vee (robo = J_1) \vee (robo = P_E) \vee (robo = P_D)] \\ \bigwedge (robo = P_E) \rightarrow \bigcirc [(robo = J_1) \vee (robo = P_E) \vee (robo = J_2)] \\ \bigwedge (robo = P_D) \rightarrow \bigcirc [(robo = J_1) \vee (robo = P_D) \vee (robo = J_2)] \\ \bigwedge (robo = J_2) \rightarrow \bigcirc [(robo = P_E) \vee (robo = P_D) \vee (robo = J_2) \vee (robo = D)] \\ \bigwedge (robo = D) \rightarrow \bigcirc (robo = D) \\ \bigwedge [(robo = P_E) \wedge \bigcirc (adv = P_E)] \rightarrow \bigcirc (robo \neq J_2) \\ \bigwedge [(robo = P_D) \wedge \bigcirc (adv = P_D)] \rightarrow \bigcirc (robo \neq J_2) \\ \bigwedge [(robo = J_2) \wedge \bigcirc (adv = P_E)] \rightarrow \bigcirc (robo \neq P_E) \\ \bigwedge [(robo = J_2) \wedge \bigcirc (adv = P_D)] \rightarrow \bigcirc (robo \neq P_D) \end{cases} \tag{3.5}$$

Nas 6 primeiras asserções da equação 3.5, é definido como o robô pode se deslocar pelo espaço de trabalho mostrado na figura 6, sem considerar o fechamento das pontes. Na segunda parte de dinâmica (definida nas 4 últimas asserções), são declaradas as restrições de movimentação associadas ao fechamento das pontes, sendo indicado que, se uma ponte estiver fechada, não é possível atravessá-la para alcançar a junção 2 (J_2). Por sua vez, caso o robô já esteja em J_2, não é possível voltar para uma ponte que esteja fechada.

$$\varphi_i^{adv} : (adv = C)$$
$$\varphi_t^{adv} : \begin{cases} (adv = C) \rightarrow \bigcirc [(adv = C) \vee (adv = P_E) \vee (adv = P_D)] \\ \bigwedge (adv \neq C) \rightarrow [(\bigcirc adv) = adv] \end{cases} \tag{3.6}$$

Na equação 3.6, é definido que o adversário inicia no centro e pode escolher entre fechar uma das pontes ou permanecer nesta posição. Uma vez que uma das pontes foi fechada ($adv = P_E$ ou $adv = P_D$), não é possível alterar o *status* do adversário, ou seja, não é possível liberar uma ponte ou bloquear a outra.

Por último, temos que não são definidas hipóteses sobre o ambiente, assim, $\varphi_g^e = \Box \Diamond true$. Por sua vez, o objetivo do sistema foi definido como $\varphi_g^s = \Box \Diamond (robo = D)$, indicando que *sempre no futuro* a posição *Destino* deverá ser alcançada. Como é definido na dinâmica do robô que, ao chegar nesta posição, ele permanecerá parado, a fórmula se torna uma especificação de alcançabilidade simples. Na figura 7, é apresentada uma versão gráfica do controlador obtido.

No controlador mostrado na figura 7, o par de valores dentro de cada estado indica os *status* do robô e do adversário, respectivamente. Assim, no estado inicial, por exemplo, é indicado que o robô está na posição *Inicio* ($robo = I$) e o adversário no centro ($adv = C$). Deste modo, o comportamento do controlador mostrado na figura 7 pode ser interpretado da seguinte forma.

O robô segue pelas posições *Inicio*, *junção 1*, *Ponte da Esquerda*, *Junção 2* e, por fim, *Destino*. Caso a ponte da esquerda seja fechada pelo adversário ($adv = P_E$), o robô utiliza o caminho passando pela ponte da direita para

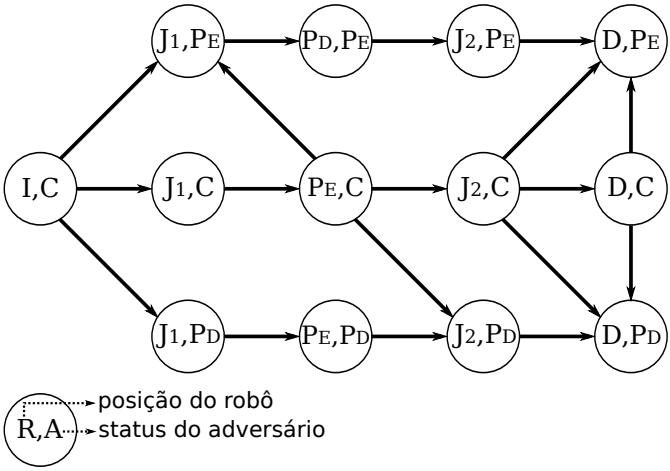


Figura 7: Controlador obtido para o exemplo referente à figura 6.

alcançar a posição final (*Destino*). Vale destacar o caso onde o robô está na ponte da esquerda e o adversário ainda está no centro (P_E,C). Nesta situação, se o adversário fechar a ponte da esquerda, o robô recua para a primeira junção e, então, segue pelo caminho da direita.

Sobre o controlador gerado, é importante ressaltar que as ferramentas utilizadas durante o trabalho não são capazes de gerar uma representação gráfica dele. Assim, a versão mostrada na figura 7 foi criada manualmente, contudo, é uma representação exata do autômato textual obtido.

3.6 DISCUSSÃO SOBRE AS ABORDAGENS

Analisando as abordagens baseadas na TCS, verificação formal e teoria de jogos (autômato-jogo e estrutura de jogo), é possível perceber que a característica permissiva dos controladores obtidos utilizando a teoria de controle supervisório torna necessário o desenvolvimento de um mecanismo para escolha da próxima ação, como o mostrado em Molina (2010)[46].

Por sua vez, quando se utiliza técnicas de verificação lógica para obter o controlador, este é gerado usando um observador ou como um contra-exemplo e correspondem a uma das execuções onde a especificação é satisfeita. Por conta disso, é impossível tratar, diretamente, um ambiente dinâmico.

A dinamicidade do ambiente pode ser abordada, por exemplo, considerando que se houver alguma mudança no espaço de trabalho, o plano de atividades é recalculado em tempo de execução e tomando a posição atual dos robôs como inicial. Contudo, dependendo da quantidade de agentes do sistema, do tamanho do espaço de trabalho e do nível de dinamicidade do ambiente considerado, dentre outros aspectos, o custo computacional do cálculo do controlador em tempo de execução pode ser muito elevado.

Por último, ao utilizar a teoria de jogos combinatória, o ambiente é tratado como adversário e o controlador, quando obtido, é capaz de satisfazer a especificação do sistema independentemente das ações do ambiente. Porém, há casos em que é impossível que os objetivos do sistema sejam realizados sem que exista uma cooperação entre sistema e ambiente.

Nos trabalhos apresentados em [20] é tratado o aspecto da geração de controladores que consideram cooperação com o ambiente, utilizando a teoria de autômato-jogo. Todavia, a forma como a cooperação entre ambiente e sistema ocorre é determinada pelo algoritmo proposto em [20] durante o cálculo do controlador. Assim, o comportamento que o ambiente deve assumir de modo a possibilitar que as especificações do sistema sejam satisfeitas não é descrito pelo projetista e podem não fazer sentido fisicamente.

Por sua vez, nos trabalhos apresentados em [56], é proposto um método baseado em estrutura de jogos em que é possível, através da realização de hipóteses (proposições na condição de justiça do ambiente) sobre o comportamento do ambiente, definir uma cooperação com o sistema. Contudo, este método não é capaz de modelar sistemas com restrições temporais, o que pode restringir, mas não inviabilizar, sua utilização em problemas de coordenação de SMRs.

Considerando os tipos de abordagens apresentados neste capítulo (TCS, *model checking*, autômato-jogo e estrutura de jogo), vale destacar os tipos de controladores obtidos usando cada uma delas. Nos métodos baseados em

TCS, o controlador contém todos os comportamentos que satisfazem as especificações da planta. Já nos métodos baseados em verificação formal, o controlador obtido corresponde a uma sequência de eventos que leva ao cumprimento da especificação do sistema, sem considerar possíveis alterações no espaço de trabalho.

Para os métodos baseados na teoria de jogos (autômato-jogo e estrutura de jogos), o controlador também é obtido como uma sequência de eventos que leva a realização das especificações do sistema. Contudo, diferente dos métodos de planejamento baseados em *model checking*, as abordagens de teoria de jogos geram estratégias com sequências de eventos alternativas, que são criadas ao considerar as possíveis mudanças no ambiente.

3.7 CONCLUSÃO

Neste capítulo, foram discutidas as principais teorias utilizadas para coordenação de SMRs. Contudo, como existe uma vasta bibliografia sobre cada uma das teorias e diversos métodos que as implementam, optou-se por destacar apenas os aspectos que as caracterizam.

Neste trabalho, o método proposto em Piterman *et al.* (2006)[56] foi estudado e utilizado para modelar e obter controladores para diversas classes de aplicações. Assim, foi possível analisar a aplicabilidade deste método na geração de estratégias de coordenação para SMRs. Vale destacar que optou-se por utilizar este método devido à existência de características como a possibilidade de definir a forma de cooperação com o ambiente e de considerar a mudança do *status* de diversos elementos da planta simultaneamente.

4 DESCRIÇÃO DO PROBLEMA

Neste capítulo é realizada a formulação do problema tratado, sendo destacados o tipo de SMRs e ambientes considerados, bem como as principais características dos robôs do sistema e dos elementos utilizados para representar a dinâmica do ambiente. Além disso, são apresentadas as classes de aplicações utilizadas para analisar a aplicabilidade, no contexto de coordenação de SMRs, de métodos baseados em estruturas de jogos.

4.1 FORMULAÇÃO DO PROBLEMA

No capítulo 2, o problema de coordenação de SMRs foi apresentado e discutido, sendo mostradas também algumas das classes de tarefas mais comuns a estes sistemas. Neste trabalho, a coordenação de SMRs foi estudada sob a ótica da teoria de estruturas de jogos, focando na utilização do método proposto em Piterman *et al.* (2006)[56] para obtenção de estratégias de coordenação que garantam o cumprimento das especificações do sistema, independentemente das ações do ambiente.

Considerando a taxonomia proposta em Iocchi *et al.* (2001) [25] e discutida no capítulo 2, os SMRs abordados neste trabalho podem ser classificados como cooperativos, cientes, de orquestração forte e centralizados. Além disso, o ambiente no qual eles estão inseridos é dinâmico e foram considerados tanto os casos em que o sistema possui observabilidade global das ações do ambiente quanto aqueles em que a observabilidade é parcial.

Neste contexto, problemas como a colisão entre agentes do sistema ou destes com algum dos elementos do ambiente e falhas em robôs precisam ser tratados, de modo a evitar a ocorrência de "maus" comportamentos nas estratégias geradas e aumentando a robustez dos sistemas, tornando-os tolerante a faltas.

Outro aspecto importante tratado neste trabalho, e que foi a grande motivação da escolha do método proposto em [56] para modelagem das aplicações, foi a cooperação por parte do ambiente. Há diversas situações onde o ambiente poderia impedir a realização de um dos objetivos do sistema, por exemplo, bloqueando o único caminho (pelo fechamento de uma porta, de uma ponte, etc.) através do qual os robôs do sistema podem alcançar uma região específica do espaço de trabalho.

Como, na maioria das aplicações, o ambiente não se caracteriza como um inimigo do sistema, pode ser interessante assumir que existe certo grau de cooperação entre eles, permitindo, assim, que uma estratégia para a realização

dos objetivos do sistema possa ser gerada.

Neste sentido, foram definidos e modelados alguns elementos básicos, a partir dos quais o comportamento das diversas partes de uma aplicação pode ser representado. Uma vez descritos estes componentes elementares, foi proposta uma metodologia para obtenção da estrutura de jogo, que representa a aplicação, através da composição deles. A seguir, são definidos cada um dos elementos e suas características principais são discutidas. Na sequência, são apresentadas três classes de aplicações, que foram utilizadas para analisar e validar a utilização de métodos baseados em estruturas de jogos na coordenação de SMRs.

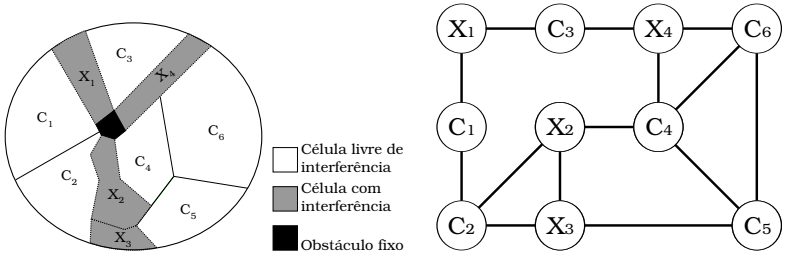
4.2 ELEMENTOS DE MODELAGEM

Para compor os modelos das aplicações, são considerados diversos elementos de modelagem, associados tanto ao ambiente quanto ao sistema. Além destes componentes, a seguir, é discutido o tipo e as características dos espaços de trabalho nos quais os robôs e os elementos do ambiente estão inseridos.

4.2.1 Espaço de trabalho

De maneira geral, quando algum dos métodos apresentados no capítulo 3 é utilizado para solução do problema de coordenação de SMRs, o espaço de trabalho deve ser dividido em áreas, representando, numa visão mais abstrata, as posições discretas (células) que podem ser ocupadas pelos agentes do ambiente e do sistema. Além disso, o espaço de trabalho é modelado indicando, para cada célula, quais posições podem ser alcançadas.

Assim, características como a forma, tamanho e posição real não são descritas, sendo definida apenas a relação de vizinhança entre as células. Então, desde que haja um controlador que garanta a realização de trajetórias, pelos robôs, entre posições vizinhas, qualquer topologia de espaço de trabalho pode ser adotada. Na figura 8(a) é apresentado um exemplo simples de espaço de trabalho e na figura 8(b) é mostrado um grafo não direcionado, representando a relação de vizinhança entre as células de espaço de trabalho.



(a) Espaço de Trabalho (b) Relação de Vizinhança

Figura 8: Exemplo de espaço de trabalho e relação de vizinhança correspondente

No espaço de trabalho e no grafo mostrados na figura 8, um nó C_i corresponde a uma célula onde não há interferência do ambiente, enquanto que um X_i está associado a uma posição que pode ser ocupada por um dos

elementos ambiente, o que pode interferir na movimentação de um robô do sistema

Além disso, foi adotado que, se qualquer uma das extremidades de um robô estiver sobre uma célula, é considerado que o agente está ocupando-a. Assim, durante o deslocamento, é assumido que o robô está ocupando a célula atual e a de destino, até que o movimento cesse. Dependendo da configuração do espaço de trabalho e das dimensões dos robôs, pode ser necessário considerar ainda que outras células estejam sendo ocupadas durante a movimentação de um agente.

4.2.2 Sistema

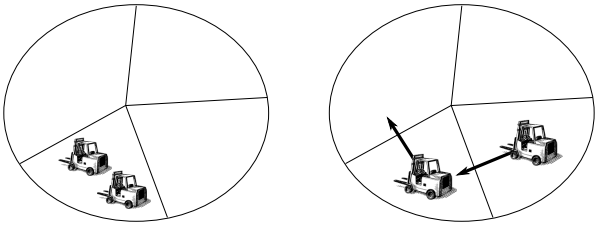
Para SMRs centralizados, um coordenador central é responsável por controlar todos os robôs, garantindo que as tarefas impostas ao sistema serão cumpridas sem que ocorram "maus comportamentos" (colisão entre robôs ou com objetos associados ao ambiente). Neste sentido, o sistema é composto pelos robôs e pelos mecanismos de segurança que evitam as situações de colisão. Além desses, os sensores dos robôs e as especificações associadas à realização de cada tarefa também fazem parte do sistema.

Neste trabalho, foram tratados dois aspectos importantes a respeito do tipo de sistema a ser desenvolvido. O primeiro está associado à forma como ele percebe as ações do ambiente, sendo estudados tanto os casos de observabilidade global quanto parcial. O segundo aspecto, por sua vez, está associado à possibilidade de falhas num (ou mais) robô. Neste sentido, foram considerados também os casos em que um agente pode sofrer uma falha e, então, permanecer imóvel durante todo o resto da execução.

Quando se considera sistemas com observabilidade parcial, nos quais as ações do ambiente são percebidas através dos sensores presentes nos robôs do sistema, deve-se levar em conta o alcance destes sensores. Este parâmetro é utilizado para definir quais células do espaço de trabalho podem ser observadas pelo sensor de um robô, dada a posição atual deste.

Outro ponto importante quando se tem um sistema composto por múltiplos robôs, é a necessidade de garantir a não colisão entre os agentes. Assim, é preciso definir um mecanismo de segurança que evite a ocorrência de uma das situações consideradas como colisão, mostradas na figura 9.

Os casos de colisão, mostrados na figura 9, correspondem às situações onde dois robôs do sistema estão ocupando a mesma célula (figura 9(a)) ou um deles se dirige à célula que estava sendo ocupada por outro agente até o instante anterior (figura 9(b)). Como foi discutido na seção 4.2.1, é assumido que, durante a etapa de movimentação, um robô ocupa a célula atual e a de



(a) (b)

Figura 9: Situações de colisão

destino. Assim, um agente do sistema R_{Si} não deve se deslocar para uma posição da qual outro robô R_{Sj} esteja saindo.

Vale destacar que, aspectos dos robôs como suas dimensões e os parâmetros específicos de seus sensores, rodas, motores e outros atuadores não afetam diretamente a modelagem, sendo utilizados apenas no desenvolvimento de controladores de trajetória, utilizados para controlar a movimentação entre células vizinhas.

4.2.3 Ambiente

Para representar a dinamicidade do ambiente onde os robôs do sistema estão inseridos, foram considerados quatro tipos de componentes: *Portas*, *robôs adversários* e *objeto com posição desconhecida* (OPD).

Porta

De forma geral, uma *porta* pode ser utilizada para representar um objeto que pode estar ocupando uma célula específica do espaço de trabalho em determinados momentos, tal como uma ponte, que pode estar fechada (levantada) ou aberta (não levantada).

Neste trabalho, a abertura e fechamento da porta são representados por eventos não controláveis pelo sistema. Além disso, é importante destacar também a necessidade de um mecanismo para evitar colisão com os robôs do sistema, impedindo que a porta feche caso haja um robô na posição à qual ela está associada.

Robôs adversários

Um *robô adversário* pode ser utilizado na modelagem de uma aplicação em que parte do espaço de trabalho é compartilhada com outros agentes

móveis, dos quais não se tem controle (robôs de um outro sistema, pessoas, etc.). Nestes casos, para o SMR que está sendo desenvolvido, este tipo de agente é considerado como um objeto dinâmico fazendo parte do ambiente.

É assumido neste trabalho que, de forma similar aos robôs do sistema, um *adversário* deve ser dotado de algum mecanismo anti-colisão, de modo que ele não tente ocupar uma célula onde já se encontra um robô do sistema.

Objeto com posição desconhecida (OPD)

Um OPD corresponde a um objeto presente no espaço de trabalho e do qual não se tem certeza da posição, sendo definido um conjunto de células nas quais ele pode estar. Assim, a posição real do objeto só pode ser determinada após o início da execução.

Como são considerados casos com observabilidade global e parcial, o objeto pode ter dois tipos de comportamentos. Na abordagem global, a célula ocupada pelo objeto é determinada no primeiro instante da execução, uma vez que o sistema pode "ver" todo o espaço de trabalho. Já para o caso parcial, o objeto só é encontrado quando o sensor de um robô do sistema o detecta numa das posições em que ele pode estar.

Além destes aspectos, é importante destacar que um OPD pode representar tanto um objeto a ser coletado quanto um obstáculo a ser evitado. De forma que, no primeiro caso ele pode ser utilizado, por exemplo, na modelagem de aplicações de busca e resgate. No segundo caso, um OPD pode estar representando incertezas sobre a posição de obstáculos fixos em certas regiões do espaço de trabalho.

4.2.4 Situações de falha

Muitos são os tipos de falhas que podem ocorrer num robô (falhas nos sensores, no sistema de navegação, nos controles dos atuadores, no sistema de comunicação, etc.) e, ainda mais variados, são os comportamentos que podem ser assumidos por um robô em falha, como cessar imediatamente o movimento, não detectar elementos do ambiente a sua frente e se mover de forma aleatória, dentre outros.

Neste trabalho, foram considerados robôs com possibilidade falha, no intuito de avaliar os métodos baseados em estrutura de jogos quanto ao aspecto da tolerância à falta. Assim, optou-se por considerar um tipo de falha bastante simples, de modo que o comportamento de um robô com problema é voltar para a célula anterior e permanecer nela até o fim da execução.

Vale destacar que a ideia de assumir que o robô volta para a célula anterior quando ocorre uma falha, está associada ao fato desta ter sido a última

posição do agente informada ao sistema antes da falha. Em trabalhos futuros, outros tipos de problemas podem ser descritos e modelados.

Por último, também é assumido que existe um número máximo de falhas, permitindo ao projetista definir um limite máximo de robôs que podem apresentar problemas. Vale destacar ainda que, aspectos como a detecção da falha não são tratados neste nível de controle, sendo assumido que o sistema recebe a informação de que ocorreu uma falha num robô R_{Si} de um dos seus subsistemas.

Vale destacar ainda a diferença entre a ocorrência de uma falta e de uma falha. Neste trabalho, uma falta no sistema significa que ele terá menos robôs disponíveis para realizar seu objetivo global. Por outro lado, a ocorrência de várias faltas (falha nos robôs) pode acarretar uma falha no sistema, fazendo com sua especificação não possa mais ser atendida devido ao sistema não ter mais controle sobre um número suficiente de robôs para realizar a tarefa. Vale destacar também que neste trabalho só é tratado o caso de tolerância à falta.

4.3 CLASSES DE APLICAÇÕES

Para validar o uso de métodos baseados em estruturas de jogos, bem como dos princípios de modelagem desenvolvidos neste trabalho, foi proposta a utilização de três classes de aplicações. Neste sentido, os tipos de tarefas que compõem cada um destas classes foram descritos de forma geral. Além disso, exemplos simples, mas representativos, de cada classe foram modelados através dos princípios propostos.

É importante ressaltar ainda que, estas classes de aplicações foram escolhidas por serem comumente utilizadas como cenários de testes em diversos trabalhos na área de SMR [25] e por serem tratáveis utilizando métodos baseados em SED, mais especificamente usando estruturas de jogos. Além disso, estas classes podem ser utilizadas para caracterizar os principais objetivos da utilização de SMRs: melhoria da eficiência na realização de uma tarefa, garantia de adaptabilidade e tolerância à falta e realização de especificações impossíveis de serem cumpridas com apenas um robô.

Aplicações com tarefas de alcançabilidade simples

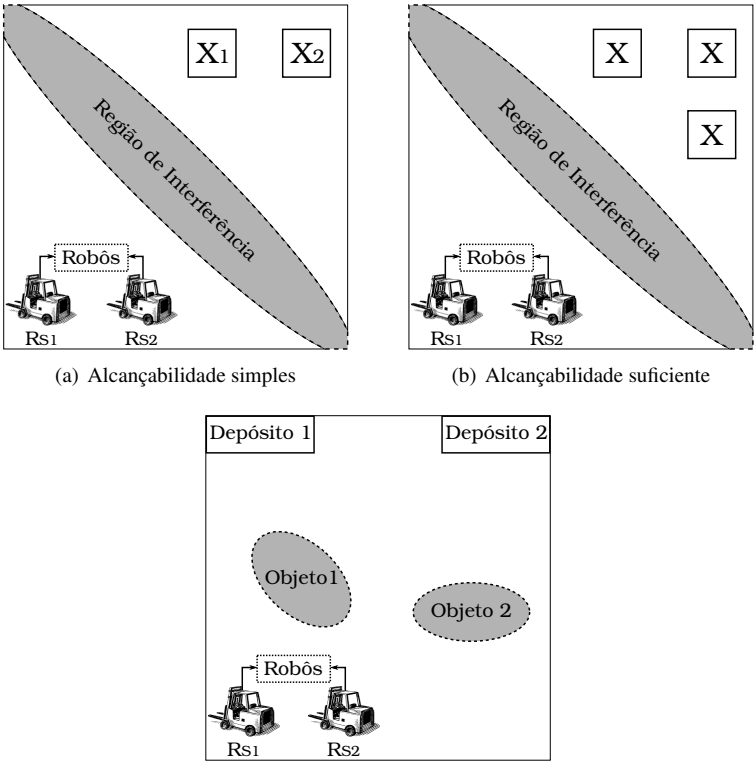
Alcançabilidade simples (ou apenas alcançabilidade) de uma posição no espaço de trabalho é o tipo mais comum de especificação em aplicações de robótica móvel, sendo utilizado para definir a posição de destino de cada agente do sistema. Neste trabalho, aplicações com objetivos de alcançabilidade foram utilizadas para verificar a aplicabilidade da teoria de estrutura de jogo e dos princípios de modelagem propostos na solução de problemas envolvendo a observabilidade global e parcial das ações do ambiente e a necessidade de uma "cooperação" mínima com os elementos do ambiente.

De modo geral, foram tratados casos em que, sem adotar hipóteses sobre o funcionamento dos elementos que compõem o ambiente, seria impossível gerar um controlador que garantisse a realização dos objetivos do sistema. Na figura 10(a) é apresentado um caso bastante simples e geral desta classe, onde dois robôs devem atravessar uma Região de Interferência (RI), na qual existem elementos do ambiente, a fim de alcançar suas posições de destino (X_1 para R_{S1} e X_2 para R_{S2}).

Como pode ser observado na figura 10(a), para chegar a suas posições de destino, os robôs devem passar pela região de interferência. Deste modo, o ambiente pode interferir na movimentação dos agentes do sistema, impedindo que eles atravessem a RI. Assim, hipóteses sobre o funcionamento do ambiente podem ser necessárias para garantir a passagem dos robôs.

Aplicações com tarefas de alcançabilidade suficiente

De forma similar aos casos com alcançabilidade simples, as aplicações



(a) Alcançabilidade simples

(b) Alcançabilidade suficiente

(c) Busca e coleta

Figura 10: Exemplos de espaço de trabalho

com objetivos de alcançabilidade suficiente são caracterizadas, neste trabalho, por serem compostas por múltiplos robôs que devem atravessar uma região de interferência a fim de alcançar suas posições de destino.

A principal diferença entre estas classes de aplicações (com especificações de alcançabilidade simples e suficiente) está no fato de que, ao considerar alcançabilidade simples, é feita uma diferenciação entre os robôs, pois apenas um agente específico pode realizar o objetivo de atingir uma determinada célula. Por sua vez, ao considerar uma especificação suficiente, basta que um dos robôs alcance a posição de destino para que o objetivo seja cumprido. Na figura 10(b) é apresentado um exemplo simples de uma aplicação com tarefas de alcançabilidade suficientes. Neste caso, para realizar o objetivo associado a uma posição indicada por X, basta que um dos robôs (R_{S1} e R_{S2}) a alcance.

Outro aspecto importante sobre esta classe de aplicações é que, por não haver diferenciação entre os agentes, caso ocorra uma falha num (ou mais) deles, outro robô pode ser alocado para realizar o objetivo previamente designado para o agente em que ocorreu a falha. Assim, esta classe também é utilizada para avaliar o método proposto em [56] e os princípios de modelagem propostos quanto à solução de problemas de tolerância à falta.

Aplicações de busca e resgate

Em aplicações de busca e resgate, o objetivo do sistema é recolher objetos espalhados pelo espaço de trabalho e levá-los a um local pré-definido. De forma geral, é utilizado o conceito de objetos com posição desconhecida (OPD). Assim, uma vez que começa a execução, o sistema identifica a posição dos objetos, os coleta e leva até a posição pré-definida. Na figura 10(c) é apresentado um exemplo bastante simples desta classe de aplicações. Como mostrado nesta figura, é informada uma região onde o objeto se encontra, mas não sua posição real.

4.4 CONCLUSÃO

Neste capítulo foi definida, utilizando a taxonomia proposta em Iocchi *et al.* (2001), a classe de SMRs tratada neste trabalho: cooperativos, cientes, de orquestração forte e centralizados. Também são apresentadas as classes de aplicações que foram utilizadas para analisar a aplicabilidade de métodos baseados em estruturas de jogos, especificamente o proposto em [56], e a viabilidade dos princípios de modelagem propostos na solução de problemas de coordenação de SMRs.

5 PRINCÍPIOS DE MODELAGEM BASEADA EM ESTRUTURAS DE JOGOS

Neste capítulo são discutidas algumas das ferramentas computacionais que implementam o método proposto em Piterman *et. al*(2006)[56], escolhido para modelar as classes de problemas abordadas e gerar estratégias de coordenação. Também são descritos cada um dos elementos de modelagem propostos neste trabalho considerando sistemas com observabilidade global e parcial.

Esses elementos de modelagem, introduzidos no capítulo 4, são utilizados para compor a estrutura de jogo que representa uma aplicação. No final do capítulo é proposta uma metodologia para descrição de aplicações através da composição dos seus diversos componentes elementares.

5.1 FERRAMENTAS DE MODELAGEM BASEADAS EM PITERMAN *ET AL.* (2006)

Neste trabalho, foi estudada a aplicabilidade de estruturas de jogos na coordenação de SMRs em ambientes dinâmicos. Neste sentido, foi proposta a utilização de várias classes de tarefas para avaliar o desempenho de controladores obtidos através do método proposto em Piterman *et al.* (2006)[56], no qual as aplicações são descritas usando estruturas de jogos.

Foram encontradas algumas ferramentas computacionais que o implementam, tais como a Anzu[26], LTLMoP[23], TuLiP[67] e a AspectLTL[40]. Contudo, apenas na Anzu, o método proposto em [56] é completamente implementado, sendo que em nenhuma das outras ferramentas é possível realizar hipóteses sobre o ambiente.

Por sua vez, a Anzu é uma ferramenta computacional voltada para a síntese de circuitos lógicos, gerando um controlador na linguagem de descrição de *hardware* Verilog. Assim, como nenhuma das ferramentas encontradas é adequada para modelar as classes de problemas estudadas, decidiu-se utilizar uma implementação do método, em java e baseada no *plug-in* JTLV[59][1], obtida dos próprios autores do artigo.

No método proposto em [56], a estrutura de jogo que modela uma aplicação é descrita através da definição da condição inicial, dinâmica e condição de justiça do sistema e do ambiente. Neste trabalho, optou-se por tratar a modelagem das aplicações de forma modular, sendo definidos modelos bá-

[1] JTLV é uma ferramenta que visa facilitar e fornecer um *framework* unificado para o desenvolvimento de algoritmos de verificação formal

sicos para representar os elementos que podem fazer parte do sistema (como robôs e sensores) e do ambiente (portas e robôs adversários, dentre outros). Na seção 5.6, a metodologia utilizada para realizar a composição de modelos é descrita em detalhes.

Vale ressaltar que, no modelo do ambiente, são descritos os comportamentos dos objetos dinâmicos que podem atrapalhar a movimentação dos robôs, interferindo na realização dos objetivos do sistema. Além disso, os modelos dos elementos do sistema e do ambiente são descritos através de asserções em lógica temporal, de modo similar ao mostrado na equação 5.1.

$$\begin{aligned} &\varphi_i^{el} : (V_i = v_{inicial}) \\ &\varphi_t^{el} : \left\{ \begin{array}{l} \cdots \\ \wedge(\text{condição}) \rightarrow \bigcirc(V_i = \text{novo valor}) \\ \cdots \end{array} \right. \\ &\varphi_g^{el} : \Box\Diamond(\text{proposição}) \end{aligned} \tag{5.1}$$

Na equação 5.1 é indicada a inicialização (φ_i^{el}) de uma das variáveis de estado V_i associadas ao elemento *el*. Além disso, é definida a dinâmica deste elemento (φ_t^{el}), indicando em uma das asserções que se uma determinada *condição* for satisfeita, a variável V_i receberá um *novo valor*. Por último, é declarada a condição de justiça do elemento (φ_g^{el}), onde é definido que uma determinada *proposição* deverá ser satisfeita sempre no futuro.

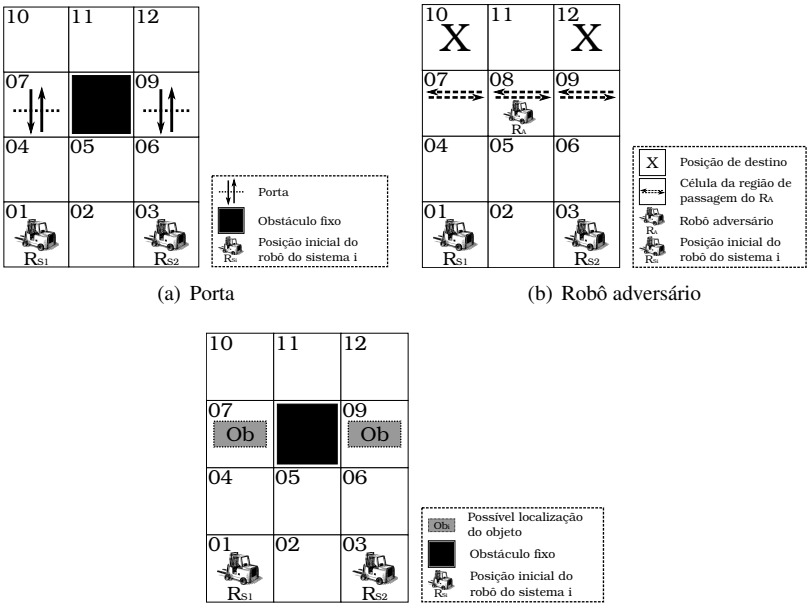
Para auxiliar na explicação dos modelos, poderão ser utilizados também exemplos baseados num dos espaços de trabalho mostrados nas figuras 11(a) (composto por portas), 11(b) (composto por robô adversário) e 11(c) (composto por objeto com posição desconhecida), onde os robôs do sistema dividem o espaço de trabalho com elementos dinâmicos associados ao ambiente (portas, robôs adversários e objetos com posição desconhecida).

É importante destacar que, apesar de terem sido utilizados espaços de trabalho com células quadradas e de mesmo tamanho para exemplificar a descrição dos componentes apresentados no capítulo 4, os modelos propostos neste trabalho podem ser aplicados a qualquer topologia de espaço de trabalho, desde que existam controladores de trajetória que garantam a movimentação dos agentes entre as células vizinhas.

A seguir são indicadas as condições em que uma célula do espaço de trabalho é considerada *ocupada*, levando em conta cada um dos elementos de modelagem (robôs do sistema, portas, adversários e objetos com posição desconhecida).

- Robô do sistema: $(R_{Si} = C_j \Rightarrow C_j = Ocupada)$. Caso um robô R_{Si} esteja numa célula C_j, ela é considerada ocupada.

- Porta: $(P_k = F \Rightarrow C_j = Ocupada)$. Se uma porta P_k associada à posição C_j estiver fechada ($P_k = F$), C_j é considerada ocupada pela porta.



(a) Porta

(b) Robô adversário

(c) Objeto com posição desconhecida (OPD)

Figura 11: Exemplos de espaço de trabalho com portas, robô adversário e OPD

- Adversário: $(R_A = C_j \Rightarrow C_j = Ocupada)$. De forma similar ao robô do sistema, se um robô adversário (R_A) estiver numa posição C_j ($R_A = C_j$), esta é considerada ocupada.
- OPD: $(Ob = C_j \Rightarrow C_j = Ocupada)$. Para um OPD ($Ob$) que representa um obstáculo fixo do qual não se tem certeza da posição, caso ele seja encontrado na posição C_j ($Ob = C_j$), esta célula é considerada ocupada.

Vale destacar ainda que, na implementação fornecida pelos autores do método apresentado em [56], as condições iniciais, dinâmica, condições de justiça são descritas no formato SMV[43, 45]. Além disso, caso exista um controlador que garanta a realização das especificações do sistema (definida em sua condição de justiça), este é representado no formato de um autômato textual.

No apêndice A são descritos os principais conceitos relacionados à modelagem em SMV, sendo apresentados os tipos de variáveis e de construções aceitos. Nas seções a seguir, os modelos propostos para os elementos de modelagem são apresentados usando fórmulas em lógica temporal.

5.2 MODELOS CONSIDERANDO OBSERVABILIDADE GLOBAL

Considerando o caso em que o sistema observa globalmente as ações dos objetos dinâmicos (portas, robôs adversários e objetos com posição desconhecida) associados ao ambiente, os elementos de modelagem são descritos como mostrado nas subseções a seguir.

Vale destacar que a seguir, para o sistema, são descritos apenas os aspectos associados a movimentação dos seus robôs (os modelos dos robôs e do mecanismo que impede a ocorrência de colisão entre eles). Apenas na seção 5.5 são apresentadas as descrições dos objetivos associados à realização das tarefas, que também fazem parte do sistema.

5.2.1 Sistema

Para o caso considerando observabilidade global, o modelo do sistema é composto pelos modelos dos seus robôs, do mecanismo de não colisão e pela descrição dos objetivos associados à realização das tarefas impostas ao sistema, sendo esta última parte tratada separadamente na seção 5.5.

Robô sem falha

No modelo de um robô, é definida a célula onde ele inicia a operação, a forma como ele pode se mover a cada passo pelo espaço de trabalho e a restrição de só se mover para uma célula da região de interferência (região onde pode haver elementos do ambiente), quando não existe um agente do ambiente ocupando-a. Vale destacar ainda que, inicialmente, foi tratado o caso mais simples de robô, o sem possibilidade de falha. Na seção 5.4 são discutidos os aspectos associados a ocorrência de falhas.

De forma geral, a dinâmica de um robô do sistema é modelada a partir da relação de vizinhança entre as células do espaço de trabalho. Assim, no modelo de um robô são definidas, para cada célula, as posições para as quais ele pode se deslocar no próximo passo. Além disso, também são descritas as restrições de movimentação impostas pelos elementos do ambiente, de forma que, quando a posição para qual o robô está se deslocando pertence a uma região de interferência, é definido que o movimento só pode ocorrer se a célula não estiver ocupada ($C_j \neq Ocupada$).

Dado que a região de interferência corresponde ao conjunto de células $RI = \{X_1, \cdots, X_K\}$, na equação 5.2 é mostrada a modelagem do elemento que representa um robô do sistema.

$$
\begin{array}{l}
\varphi_i^{R_{Si}} : (R_{S_i} = C_{inicial}^{Rsi}) \\
[h] \quad \varphi_t^{R_{Si}} : \left\{ \begin{array}{l} \bigwedge\limits_{j=1}^{N_C} \left\{ (R_{Si} = C_j) \rightarrow \bigcirc \left[(R_{Si} = C_j) \vee \left(\bigvee\limits_{C_h \in Viz(C_j)} (R_{Si} = C_h) \right) \right] \right\} \\ \bigwedge\limits_{j=1}^{N_C} \left\{ \bigcirc (X_j = Ocupada) \rightarrow \bigcirc (R_{Si} \neq X_j) \right\} \end{array} \right. \\
\varphi_g^{R_{Si}} : \Box \Diamond true
\end{array}
\tag{5.2}
$$

onde $C_{inicial}^{Rsi}$ corresponde a posição inicial do i-ésimo robô do sistema (R_{Si}), $C_{ET} = \{C_1, \cdots, C_{N_C}\}$ é o conjunto de células do espaço de trabalho e a função $Viz(C_i) : C_{ET} \rightarrow 2^{C_{ET}}$ define as células na vizinhança da posição C_i.

Na fórmula $\varphi_i^{R_{Si}}$ é definida a posição inicial do robô do sistema R_{Si}. Já na equação $\varphi_t^{R_{Si}}$ é descrita a dinâmica do robô, sendo que, na primeira linha é indicado para quais células o robô pode se deslocar no próximo passo, dada a posição atual. Na segunda linha, são indicadas as restrições de movimentação em cada uma das células da região de interferência, de modo que, se uma posição X_j estiver sendo ocupada por algum elemento do ambiente ($X_j = Ocupada$), o robô não poderá se deslocar para X_j no próximo passo.

É importante destacar que, na restrição de movimentação, é levado em conta o próximo *status* da célula, representado por $\bigcirc(X_j = Ocupada)$. Isso foi feito para permitir que o modelo da planta contenha a possibilidade do *status* do ambiente mudar durante o movimento do robô. Além disso, com o intuito de evitar um aumento desnecessário no espaço de estados da estrutura de jogo, optou-se por não incluir na modelagem as células onde existem obstáculos fixos, ao invés de definir uma restrição de movimentação sobre ela.

Vale ressaltar também que, por serem tratados apenas SMRs de organização centralizada, apesar das especificações estarem associadas aos robôs do sistema, estes agentes por si só não possuem objetivos, de modo que o sistema como um todo é que possui uma especificação. Por conta disso, nos modelos dos robôs do sistema não é definida condição de justiça ($\varphi_g^{R_{Si}} : \Box \Diamond true$).

Para ilustrar a modelagem de um robô do sistema, é utilizado o exemplo mostrado na figura 11(a), onde é considerado um espaço de trabalho com 12 células, um obstáculo fixo na posição 8 e duas portas, nas posições 7 e 9. Para este caso, o modelo de um robô R_{Si} é dado pela equação 5.3.

$$
\begin{array}{l}
\varphi_i^{R_{Si}} : (R_{S_i} = C_{inicial}^{Rsi}) \\
\varphi_t^{R_{Si}} : \left\{ \begin{array}{l}
(R_{Si} = 1) \rightarrow \bigcirc [(R_{Si} = 1) \vee (R_{Si} = 2) \vee (R_{Si} = 4)] \\
\bigwedge (R_{Si} = 2) \rightarrow \bigcirc [(R_{Si} = 2) \vee (R_{Si} = 1) \vee (R_{Si} = 3) \vee (R_{Si} = 5)] \\
\cdots \\
\bigwedge (R_{Si} = 12) \rightarrow \bigcirc [(R_{Si} = 12) \vee (R_{Si} = 11) \vee (R_{Si} = 9)] \\
\bigwedge \bigcirc (P_1 \neq A) \rightarrow \bigcirc (R_{Si} \neq 7) \\
\bigwedge \bigcirc (P_2 \neq A) \rightarrow \bigcirc (R_{Si} \neq 9)
\end{array} \right. \\
\varphi_g^{R_{Si}} : \Box \Diamond true
\end{array}
\tag{5.3}
$$

Sobre a descrição apresentada na equação 5.3, vale destacar a declaração das restrições de movimentação. Para o espaço de trabalho mostrado na figura 11(a), o ambiente é composto pelas portas P_1 e P_2, assim, as células da RI (7 e 9) são consideradas como livres quando a porta associada está aberta.

Mecanismo de não colisão entre agentes

No capítulo 4 são descritas as situações consideradas como colisões entre agentes do sistema: quando estão ocupando a mesma célula ou se dirigindo para uma posição que ainda está sendo liberada por outro robô do sistema. Assim, a fim de evitar a geração de controladores em que haja colisão entre robôs do sistema, foi criado um mecanismo de não colisão.

Para descrever este mecanismo é utilizada uma variável auxiliar *NCol*, a qual é inicializada com *true* e recebe *false* caso alguma das situações de colisão aconteça. Uma vez que *NCol* recebe *false*, ela não pode mais voltar a ser verdade. Isso, associado à inclusão da fórmula $\Box\Diamond(NCol = true)$ na condição de justiça do mecanismo, garante que as estratégias geradas nunca apresentem situações de colisão, pois, uma vez que uma delas ocorresse, a fórmula $\Box\Diamond\,(NCol = true)$ não poderia ser mais satisfeita. Na equação 5.4 é descrito o modelo do mecanismo de não colisão.

$$
\begin{array}{l}
\varphi_i^{NCol} : (NCol = true) \\
\varphi_t^{NCol} : \left\{ \begin{array}{l}
\neg \left[\bigvee_{i=1}^{N_R - 1} \left(\bigvee_{j=i+1}^{N_R} \left[(R_{Si} = R_{Sj}) \vee (\bigcirc R_{Si} = R_{Sj}) \vee (R_{Si} = \bigcirc R_{Sj}) \right] \right) \right] \\
\quad \rightarrow [(\bigcirc NCol) = NCol] \\
\bigwedge \left[\bigvee_{i=1}^{N_R - 1} \left(\bigvee_{j=i+1}^{N_R} \left[(R_{Si} = R_{Sj}) \vee (\bigcirc R_{Si} = R_{Sj}) \vee (R_{Si} = \bigcirc R_{Sj}) \right] \right) \right] \\
\quad \rightarrow \bigcirc (NCol = false)
\end{array} \right. \\
\varphi_g^{NCol} : \Box\Diamond\,(NCol = true)
\end{array}
\tag{5.4}
$$

onde N_R representa a quantidade de robôs do sistema.

Na primeira linha da fórmula φ_t^{NCol} é indicado que, se não houver mais de um robô ocupando a mesma célula ($R_{Si} = R_{Sj}$) e se nenhum dos agentes

se deslocar para uma célula que, até o instante anterior, estava sendo ocupada por outro robô ($\bigcirc R_{Si} = R_{Sj}$ ou $R_{Si} = \bigcirc R_{Sj}$) a variável *NCol* não é alterada. Caso contrário, *NCol* recebe o valor *false*, indicando que houve uma situação de colisão.

Considerando um espaço de trabalho qualquer (como o descrito na figura 11(a)), onde existem dois robôs do sistema, o modelo do mecanismo de não colisão é definido como mostrado na equação 5.5.

$$\begin{array}{l} \varphi_i^{NCol} : (NCol = true) \\ \varphi_t^{NCol} : \left\{ \begin{array}{l} \neg [(R_{S1} = R_{S2}) \vee (\bigcirc R_{S1} = R_{S2}) \vee (R_{S1} = \bigcirc R_{S2})] \\ \qquad \rightarrow [(\bigcirc NCol) = NCol] \\ \bigwedge [(R_{S1} = R_{S2}) \vee (\bigcirc R_{S1} = R_{S2}) \vee (R_{S1} = \bigcirc R_{S2})] \\ \qquad \rightarrow \bigcirc (NCol = false) \end{array} \right. \\ \varphi_g^{NCol} : \Box \Diamond (NCol = true) \end{array} \tag{5.5}$$

5.2.2 Ambiente

5.2.2.1 Porta

De modo geral, uma porta possui dois *status*: aberta e fechada. Além diss, sua abertura e fechamento se dão sem a intervenção do sistema. A única restrição sobre o funcionamento deste elemento é que, caso a posição do espaço de trabalho à qual a porta está associada esteja ocupada, ela não poderá ser fechada. Esta restrição é um mecanismo de segurança que evita a colisão da porta com os robôs do sistema.

Quando se usa o conceito de porta para modelar um elemento do ambiente, não faz sentido considerar que este agente esteja sempre "fechado" (ocupando a posição associada a ele), pois este caso seria equivalente ao comportamento de um obstáculo fixo. Assim, é bastante razoável assumir que uma porta irá abrir e fechar indefinidas vezes durante um ensaio.

Deste modo, na condição de justiça do ambiente é descrita a hipótese de que a porta não estará sempre fechada. Para uma porta P, associada à posição X_P, seu modelo é descrito na equação 5.6, onde são apresentadas as descrições de sua condição inicial (φ_i^P), dinâmica (φ_t^P) e hipótese (φ_g^P).

$$\begin{array}{l} \varphi_i^P : (P = A) \\ \varphi_t^P : (X_P = Ocupada) \rightarrow \bigcirc (P = A) \\ \varphi_g^P : \Box \Diamond (P \neq F) \end{array} \tag{5.6}$$

Neste trabalho, foi assumido que inicialmente a porta está aberta ($P = A$). Contudo, ao adotar que o *status* inicial da porta é fechada, não são geradas mudanças significativas no modelo final da planta.

Na fórmula φ_t^P é indicado apenas que, se a posição associada à porta (X_P) estiver sendo ocupada, ela deve permanecer aberta. Para os casos em que X_P está livre, nada é definido sobre o estado da porta, permitindo que o *status* da porta varie entre aberta ($P = A$) e fechada ($P = F$) irrestritamente.

Para ilustrar o modelo de uma porta, é utilizada a porta P_1, posicionada na célula 7 do espaço de trabalho mostrado na figura 11(a). Na equação 5.7 é apresentado o modelo deste elemento.

$$\begin{array}{l} \varphi_i^{P_1} : (P_1 = A) \\ \varphi_t^{P_1} : [(R_{S1} = 7) \vee (R_{S2} = 7)] \rightarrow \bigcirc (P_1 = A) \\ \varphi_g^{P_1} : \Box\Diamond (P_1 \neq F) \end{array} \tag{5.7}$$

5.2.2.2 Robô adversário

Neste trabalho, o elemento *robô adversário* foi utilizado para representar agentes controlados por outros sistemas que dividem o espaço de trabalho com os robôs do SMR tratado. Assim, o modelo de um adversário foi descrito de forma bastante similar ao robô do sistema. Além disso, a cada robô adversário é associado um conjunto de células do espaço de trabalho sobre o qual ele pode se deslocar: sua Região de Movimentação (RM). Esta região é uma partição do espaço de trabalho dada por $RM = \{X_1, \cdots, X_K\} \subseteq C_{ET}$. Na equação 5.8 é descrito o modelo de um adversário.

$$\begin{array}{l} \varphi_i^{R_A} : (R_A = X_{inicial}) \\ \varphi_t^{R_A} : \left\{ \begin{array}{l} \bigwedge\limits_{j=1}^{K} \left\{ (R_A = X_j) \rightarrow \bigcirc \left[(R_A = X_j) \bigvee\limits_{X_h \in Viz(X_j)} (R_A = X_h) \right] \right\} \\ \bigwedge\limits_{j=1}^{K} \left\{ (X_j = Ocupada) \rightarrow \bigcirc (R_A \neq X_j) \right\} \end{array} \right. \\ \varphi_g^{R_A} : \Box\Diamond true \end{array} \tag{5.8}$$

onde $X_{inicial}$ corresponde a posição inicial do adversário e a função $Viz(X) : RM \rightarrow 2^{RM}$ define as células da região de movimentação do adversário na vizinhança da posição X.

Assim como foi descrito no modelo do robô do sistema, na equação 5.8 é definida a posição inicial do adversário ($\varphi_i^{R_A}$) e sua dinâmica ($\varphi_t^{R_A}$), na qual é indicado como o adversário pode se deslocar por sua região de movimentação e as restrições de não ocupar uma célula que esteja ocupada. Além disso, também não foram definidas condições de justiça ($\varphi_g^{R_A}$) para o adversário.

Considerando o caso apresentado na figura 11(b), onde é mostrado um espaço de trabalho com 12 células, das quais o robô adversário pode ocupar a 7, 8 e 9, e um sistema com dois robôs, o modelo do adversário é descrito como mostrado na equação 5.9.

$$\begin{array}{l} \varphi_i^{R_A} : (R_A = 8) \\ \varphi_t^{R_A} : \left\{ \begin{array}{l} (R_A = 7) \rightarrow \bigcirc[(R_A = 7) \vee (R_A = 8)] \\ \bigwedge (R_A = 8) \rightarrow \bigcirc[(R_A = 8) \vee (R_A = 7) \vee (R_A = 9)] \\ \bigwedge (R_A = 9) \rightarrow \bigcirc[(R_A = 9) \vee (R_A = 8)] \\ \bigwedge [(R_{S1} = 7) \vee (R_{S2} = 7)] \rightarrow \bigcirc(R_A \neq 7) \\ \bigwedge [(R_{S1} = 8) \vee (R_{S2} = 8)] \rightarrow \bigcirc(R_A \neq 8) \\ \bigwedge [(R_{S1} = 9) \vee (R_{S2} = 9)] \rightarrow \bigcirc(R_A \neq 9) \end{array} \right. \\ \varphi_g^{R_A} : \square\Diamond\, true \end{array} \tag{5.9}$$

Na fórmula $\varphi_t^{R_A}$, a condição de célula ocupada ($X_j = Ocupada$) foi definida de modo que, se um dos robôs do sistema estiver ocupando uma células X_j ($(R_{S1} = X_j) \vee (R_{S2} = X_j)$), o adversário não pode se deslocar, no próximo passo, para esta posição.

5.2.2.3 Objeto com posição desconhecida (OPD)

No contexto de observabilidade global, um objeto com posição desconhecida é modelado de forma que a célula em que ele está é informada ao sistema imediatamente após o início da execução. Assim, dado que o objeto está em uma das células do conjunto $C_{Ob} = \{X_1, \cdots, X_K\}$, após o início da execução, o *status* do objeto deve mudar de desconhecido (indicado pelo valor 0) para um valor X_j, que indica a célula em que está o objeto. Deste modo, a modelagem do objeto é feita como descrito na equação 5.10.

$$\begin{array}{l} \varphi_i^{Ob} : (Ob = 0) \\ \varphi_t^{Ob} : \left\{ \begin{array}{l} (Ob = 0) \rightarrow \bigcirc \left(\bigvee_{j=1}^{K} (Ob = X_j) \right) \\ \bigwedge (Ob \neq 0) \rightarrow [(\bigcirc Ob) = Ob] \end{array} \right. \\ \varphi_g^{Ob} : \square\Diamond true \end{array} \tag{5.10}$$

Na equação 5.10, o *status* do OPD é, inicialmente, definido como desconhecido ($Ob = 0$), uma vez que sua real posição só passa a ser conhecida após o início da execução. Quanto à dinâmica, na primeira linha da equação φ_t^{Ob}, é descrito que, se a posição do objeto ainda não é conhecida ($Ob = 0$), no próximo passo a variável Ob deve receber um valor X_j referente à célula em que o objeto está. Na segunda linha, é definido que após ter sido determinada a posição do objeto ($Ob \neq 0$), o valor de Ob não poderá mais ser alterado. Além disso, não é definida nenhuma condição de justiça para este elemento.

Considerando o exemplo mostrado na figura 11(c), onde um objeto representado por um OPD pode estar nas posições 7 ou 9, o modelo do OPD é descrito como mostrado na equação 5.11.

$$\begin{array}{l} \varphi_i^{Ob} : (Ob = 0) \\ \varphi_t^{Ob} : \left\{ \begin{array}{l} (Ob = 0) \rightarrow \bigcirc ((Ob = 7) \vee (Ob = 9)) \\ \bigwedge (Ob \neq 0) \rightarrow [(\bigcirc Ob) = Ob] \end{array} \right. \\ \varphi_g^{Ob} : \Box \Diamond true \end{array} \tag{5.11}$$

É importante ressaltar que, apesar do OPD ser utilizado para representar um objeto imóvel, quando o sistema é composto por este tipo de elemento, o controlador gerado considera cada uma das possibilidades referentes à posição deste objeto. Desta forma, este elemento do ambiente pode ser utilizado, por exemplo, para representar obstáculos fixos, cujas posições não são conhecidas.

5.3 MODELOS CONSIDERANDO OBSERVABILIDADE PARCIAL

Para o caso com observabilidade parcial, os modelos dos elementos do ambiente descrevem como o sistema os percebem e não seu comportamento real e completo. Isto é, são consideradas na modelagem dos elementos do ambiente apenas as variações que podem ser percebidas pelo sistema, através dos sensores dos seus robôs.

De forma geral, a dinâmica de um elemento do ambiente é composta por uma parte referente ao seu comportamento real (descrito na modelagem considerando observabilidade global), mais uma restrição de que o elemento só pode ser detectado numa determinada célula da região de interferência, caso haja pelo menos um robô observando (através de seu sensor) esta célula. Neste sentido, ao considerar observabilidade parcial, é necessário definir o *status Indeterminado*, o qual indica que o sistema não possui qualquer informação sobre o elemento.

5.3.1 Sistema

Quando são considerados modelos com observabilidade parcial, é necessário introduzir a ideia de *sensor*, através do qual as ações do ambiente são "sentidas" pelos robôs e informadas ao sistema. Entretanto, neste trabalho foi considerado que o sensor de cada robô é capaz de detectar uma única célula por vez, modelando, assim, o caso mais restritivo. Deste modo, o modelo gerado para o robô pode ser utilizado mesmo quando o sistema é composto por agentes cujos sensores possuem um alcance maior.

Existem ainda duas considerações importantes sobre o modelo gerado para representar um robô do sistema com observabilidade parcial. O primeiro está associado ao fato de que o uso de sensores para observar células que não fazem parte da região de interferência (RI) não traz nenhum ganho ao sistema, uma vez que os elementos do ambiente não podem ocupar estas células. Além disso, quando são considerados modelos em que o sensor é capaz de detectar o *status* de todas as células do espaço de trabalho (não só as da região de interferência), há um aumento significativo do espaço de estados da estrutura de jogo que representa a aplicação.

Assim, a fim de evitar um aumento desnecessário do espaço de estados da estrutura de jogo, optou-se por definir que o sensor só é "ligado" para verificar o *status* em células da região de interferência. A segunda consideração assumida se refere à imposição de que, para um robô se deslocar para uma célula da RI, é necessário que ele esteja com seu sensor ligado e direcionado para esta posição. Isso evita que o robô colida com um elemento associado

ao ambiente ao se deslocar para uma célula que ele não sabe se está livre.

Na equação 5.12 é definido, de forma geral, o modelo de um robô para o caso em que o sistema opera com observabilidade parcial ($\varphi^{Rp_{Si}}$).

$$\varphi^{Rp_{Si}} = \varphi^{R_{Si}} \wedge \varphi^{s_i} \Rightarrow \begin{cases} \varphi_i^{Rp_{Si}} = \varphi_i^{R_{Si}} \wedge \varphi_i^{s_i} \\ \varphi_t^{Rp_{Si}} = \varphi_t^{R_{Si}} \wedge \varphi_t^{s_i} \\ \varphi_g^{Rp_{Si}} = \Box\Diamond true \end{cases} \tag{5.12}$$

onde a fórmula $\varphi^{R_{Si}}$ descreve o modelo de um robô do sistema considerando observabilidade global. Já na equação φ^{s_i} é definido o modelo do sensor do robô i e a restrição referente ao deslocamento para células da região de região de interferência com o sensor ligado. Vale lembrar que, para os sensores não são definidas condições de justiça, assim, na equação 5.13 são descritas apenas a condição inicial de um sensor e sua dinâmica.

$$\begin{aligned} &\varphi_i^{s_i} : (s_i = 0) \\ &\varphi_t^{s_i} : \begin{cases} \bigwedge_{j=1}^{K} \left\{ \bigcirc \left(\bigwedge_{X_h \in Viz(X_j)} (R_{Si} \neq X_h) \right) \rightarrow \bigcirc (s_i \neq X_j) \right\} \\ \bigwedge_{j=1}^{K} \left[(s_i \neq X_j) \rightarrow \bigcirc (R_{Si} \neq X_j) \right] \end{cases} \end{aligned} \tag{5.13}$$

onde K corresponde ao número de células da região de interferência, a função $Viz(X_j)$ define as células na vizinhaça da posição X_j e s_i indica a posição que está sendo observada pelo sensor do robô R_{Si}.

Neste trabalho foi assumido que os robôs iniciam a execução com seus sensores desativados ($s_i = 0$). Em relação a dinâmica do sensor, na linha 1 da fórmula $\varphi_t^{s_i}$ é definido que, se no próximo passo o robô R_{Si} não estiver na vizinhança de uma célula $X_j \in RI$, o sensor não poderá ser direcionado para esta posição ($s_i \neq X_j$). Por sua vez, na segunda linha da fórmula $\varphi_t^{s_i}$ é definido que, se o sensor de R_{Si} não estiver ligado e direcionado para a célula $X_j \in RI$, este robô não poderá se deslocar, no próximo passo, para esta posição.

Vale ressaltar que, a restrição de movimentação para células da RI associada ao sensor (só pode se movimentar quando o sensor está ligado) tem como objetivo garantir a segurança durante a movimentação dos robôs, impedindo que um agente se mova em direção a uma célula sem ter certeza que ela não está ocupada.

Considerando o exemplo mostrado na figura 11(b), onde espaço de trabalho é compartilhado com um adversário que pode ocupar as células 7, 8 e 9, o modelo do sensor de um robô é descrito como mostrado na equação 5.14.

$$\varphi_i^{s_i} : (s_i = 0)$$
$$\varphi_t^{s_i} : \begin{cases} \bigcirc[(R_{Si} \neq 4) \wedge (R_{Si} \neq 8) \wedge (R_{Si} \neq 10)] \rightarrow \bigcirc(s_i \neq 7) \\ \bigwedge \bigcirc[(R_{Si} \neq 5) \wedge (R_{Si} \neq 7) \wedge (R_{Si} \neq 9) \wedge (R_{Si} \neq 11)] \rightarrow \bigcirc(s_i \neq 8) \\ \bigwedge \bigcirc[(R_{Si} \neq 6) \wedge (R_{Si} \neq 8) \wedge (R_{Si} \neq 12)] \rightarrow \bigcirc(s_i \neq 9) \\ \bigwedge (s_i \neq 7) \rightarrow \bigcirc(R_{Si} \neq 7) \\ \bigwedge (s_i \neq 8) \rightarrow \bigcirc(R_{Si} \neq 8) \\ \bigwedge (s_i \neq 9) \rightarrow \bigcirc(R_{Si} \neq 9) \end{cases} \tag{5.14}$$

5.3.2 Ambiente

5.3.2.1 Porta

Para uma porta, o comportamento percebido pelo sistema pode ser entendido da seguinte forma. Enquanto não há nenhum sensor direcionado para a posição associada a uma porta, seu *status* é dito *não-determinado* ($P = ND$). Uma vez que existe um sensor direcionado para a posição associada a ela, seu *status* deve ser mudado para aberta ($P = A$) ou fechada ($P = F$).

Vale destacar ainda que, mesmo que não haja sensores percebendo o *status* de uma porta, se um robô estiver na posição associada a ela, seu *status* não pode ser dito *não-determinado*, permanecendo como *aberta*.

Como os sensores dos robôs são inicializados com 0 (desligados), foi adotado que, inicialmente, uma porta não é detectada. Além disso, a hipótese realizada sobre este elemento continua a mesma que a assumida para o caso global, $\square\Diamond(P \neq F)$. Na equação 5.15 são descritas a condição inicial, a dinâmica e a hipótese de uma porta, considerando o caso com observabilidade parcial.

$$\varphi_i^P : (P = ND)$$
$$\varphi_t^P : \begin{cases} \left\{ \bigwedge_{i=1}^{N_R} \neg[(s_i = X_j) \vee (R_{Si} = X_j)] \right\} \rightarrow \bigcirc(P = ND) \\ \wedge \left\{ \bigvee_{i=1}^{N_R} (s_i = X_j) \right\} \rightarrow \bigcirc[(P = A) \vee (P = F)] \\ \wedge \left\{ \bigvee_{i=1}^{N_R} (R_{Si} = X_j) \right\} \rightarrow \bigcirc(P = A) \end{cases} \tag{5.15}$$
$$\varphi_g^P : \square\Diamond(P \neq F)$$

Dado que a porta está na célula X_j, na primeira linha da fórmula da dinâmica (φ_t^P) é definido que, se não houver nenhum sensor direcionado para a célula X_j e se também não houver nenhum robô do sistema nesta posição,

a porta não pode ser detectada. Na segunda linha desta fórmula é descrito que, se houver pelo menos um sensor observando a posição X_j, a porta é detectada e deve assumir o *status* aberta ($P = A$) ou fechada ($P = F$). Por último é definido o mecanismo de segurança da porta, o qual evita que uma porta feche caso haja um robô do sistema na posição associada a ela.

Para o exemplo mostrado na figura 11(a), onde existe uma porta (P_1) na posição 7 e é considerado um sistema com dois robôs, o modelo da porta é descrito (no caso parcial) na equação 5.16.

$$\begin{array}{l} \varphi_i^{P_1} : (P_1 = ND) \\ \varphi_t^{P_1} : \left\{ \begin{array}{l} \{\neg[(s_1 = 7) \vee (R_{S1} = 7)] \wedge \neg[(s_2 = 7) \vee (R_{S2} = 7)]\} \rightarrow \bigcirc(P_1 = ND) \\ \bigwedge [(s_1 = 7) \vee (s_2 = 7)] \rightarrow \bigcirc[(P_1 = A) \vee (P_1 = F)] \\ \bigwedge [(R_{S1} = 7) \vee (R_{S2} = 7)] \rightarrow \bigcirc(P_1 = A) \end{array} \right. \\ \varphi_g^{P_1} : \Box\Diamond(P_1 \neq F) \end{array} \tag{5.16}$$

5.3.2.2 Robôs adversários

Quando é considerado o caso com observabilidade parcial, o sistema só possui informação sobre um robô adversário caso haja um de seus robôs observando a posição onde o agente do ambiente está. Assim, a dinâmica deste elemento leva em conta, além da relação de vizinhança entre as células que formam a região pela qual ele se movimenta, o fato de que a qualquer momento o adversário pode passar a não ser mais detectado.

Vale destacar ainda que foi assumido que, uma vez que a posição do adversário não é mais conhecida, no próximo passo, ele pode ser detectado em qualquer célula de sua região de movimentação. Além disso, foi adotado que inicialmente o adversário não está sendo detectado. Na equação 5.17 são descritas a condição inicial, a dinâmica e a condição de justiça de um robô adversário.

$$\begin{array}{l} \varphi_i^{R_A} : (R_A = X_{inicial}) \\ \varphi_t^{R_A} : \left\{ \begin{array}{l} \bigwedge_{j=1}^{K} \left\{ (R_A = X_j) \rightarrow \bigcirc \left[(R_A = X_j) \vee (R_A = 0) \vee \left(\bigvee_{X_h \in Viz(X_j)} (R_A = X_h) \right) \right] \right\} \\ \bigwedge (R_A = 0) \rightarrow \bigcirc \left[(R_A = 0) \vee \left(\bigvee_{j=1}^{K} (R_A = X_j) \right) \right] \\ \bigwedge_{j=1}^{K} \left\{ (X_j = Ocupada) \rightarrow \bigcirc (R_A \neq X_j) \right\} \\ \bigwedge_{j=1}^{K} \left\{ \left(\bigwedge_{i=1}^{N_R} (s_i \neq X_j) \right) \rightarrow \bigcirc (R_A \neq X_j) \right\} \end{array} \right. \\ \varphi_g^{R_A} : \Box\Diamond true \end{array} \tag{5.17}$$

onde N_R é o número de robô do sistema, K corresponde ao número de células da região de movimentação do adversário e os valores X_j correspondem a cada uma das posições que compõem esta região.

Sobre a equação 5.17, vale destacar a inclusão (em relação ao modelo do caso com observabilidade global) das linhas 2 e 4 na fórmula $\varphi_t^{R_A}$. Na linha 2, é definido que se a posição do adversário não é conhecida, no próximo passo ele pode ser detectado em qualquer célula de sua região de movimentação. Por sua vez, na linha 4 é definido que se não houver sensores observando uma célula X_j, o adversário não pode ser detectado nela. Além disso, na primeira linha é acrescentada a possibilidade do sistema perder a informação referente à posição do adversário.

Considerando o espaço de trabalho mostrado na figura 11(b), onde a região de movimentação do adversário é composta pelas células 7, 8 e 9 e existem dois robôs do sistema no espaço de trabalho, o modelo do adversário é descrito como mostrado na equação 5.18.

$$
\begin{aligned}
&\varphi_i^{R_A} : (R_A = 0) \\
&\varphi_t^{R_A} : \left\{ \begin{array}{l}
(R_A = 7) \rightarrow \bigcirc [(R_A = 7) \vee (R_A = 0) \vee (R_A = 8)] \\
\bigwedge (R_A = 8) \rightarrow \bigcirc [(R_A = 8) \vee (R_A = 7) \vee (R_A = 9) \vee (R_A = 0)] \\
\bigwedge (R_A = 9) \rightarrow \bigcirc [(R_A = 9) \vee (R_A = 8) \vee (R_A = 0)] \\
\bigwedge (R_A = 0) \rightarrow \bigcirc [(R_A = 0) \vee (R_A = 7) \vee (R_A = 8) \vee (R_A = 9)] \\
\bigwedge [(R_{S1} = 7) \vee (R_{S2} = 7)] \rightarrow \bigcirc (R_A \neq 7) \\
\bigwedge [(R_{S1} = 8) \vee (R_{S2} = 8)] \rightarrow \bigcirc (R_A \neq 8) \\
\bigwedge [(R_{S1} = 9) \vee (R_{S2} = 9)] \rightarrow \bigcirc (R_A \neq 9) \\
\bigwedge [(s_1 \neq 7) \wedge (s_2 \neq 7)] \rightarrow \bigcirc (R_A \neq 7) \\
\bigwedge [(s_1 \neq 8) \wedge (s_2 \neq 8)] \rightarrow \bigcirc (R_A \neq 8) \\
\bigwedge [(s_1 \neq 9) \wedge (s_2 \neq 9)] \rightarrow \bigcirc (R_A \neq 9)
\end{array} \right. \\
&\varphi_g^{R_A} : \square \Diamond true
\end{aligned}
\tag{5.18}
$$

5.3.2.3 Objeto com posição desconhecida (OPD)

Na abordagem parcial, um objeto com posição desconhecida é modelado de forma que a posição real do objeto só encontrada quando o sensor de um dos robôs do sistema o detecta. Além disso, para que o sistema saiba quais das posições em que o objeto pode estar já foram verificadas, é definido também um mecanismo de memória.

Como mecanismo de memória, optou-se por utilizar um conjunto de variáveis, sendo que cada uma delas armazena o *status* ($Visitada \Rightarrow v$ e $NaoVisitada \Rightarrow nv$) de uma posição onde o objeto pode estar. Na equação 5.19 são descritas a condição inicial, a dinâmica e a condição de justiça de um OPD.

$$
\begin{aligned}
&\varphi_i^{Ob} : (Ob = 0) \wedge \left(\bigwedge_{j=1}^{K} (V_{Xj} = nv) \right) \\
&\varphi_t^{Ob} : \begin{cases}
(Ob \neq 0) \rightarrow \left[[(\bigcirc Ob) = Ob)] \wedge \left(\bigcirc \bigwedge_{j=1}^{K} (V_{Xj} = v) \right) \right] \\
\bigwedge_{j=1}^{K} \left\{ (Ob = 0) \wedge \neg \left(\bigvee_{i=1}^{N_R} (s_i = X_j) \right) \rightarrow \bigcirc \left[(\bigcirc V_{Xj}) = V_{Xj} \right] \right\} \\
\bigwedge_{j=1}^{K} \left\{ (Ob = 0) \wedge \left(\bigvee_{i=1}^{N_R} (s_i = X_j) \right) \rightarrow \bigcirc (V_{Xj} = v) \right\} \\
\bigwedge_{j=1}^{K} \left\{ (Ob = 0) \wedge \left[\neg \left(\bigvee_{i=1}^{N_R} (s_i = X_j) \right) \vee (V_{Xj} = v) \right] \rightarrow \bigcirc (Ob \neq X_j) \right\} \\
\bigcirc \left(\bigwedge_{j=1}^{K} (V_{Xj} = v) \right) \rightarrow \bigcirc (Ob \neq 0)
\end{cases} \\
&\varphi_g^{Ob} : \Box \Diamond true
\end{aligned}
\tag{5.19}
$$

onde V_{Xj} é uma variável de estado que indica se a posição X_j já foi visitada, K é o número de células em que o objeto pode estar e N_R é a quantidade de robôs do sistema.

Na primeira linha da fórmula φ_t^{Ob} é definido que, se já foi determinada a posição do objeto ($Ob \neq 0$), o valor da variável Ob não pode mais ser mudado. Além disso, como o objeto só pode estar em um único local, após ele ser encontrado, todas as posições em que o OPD poderia estar são consideradas como visitadas.

Na segunda linha, é definido que, se o objeto ainda não foi encontrado ($Ob = 0$) e existe pelo menos um sensor observando uma das posições possíveis, esta posição é considerada como visitada. Já na linha 3 é dito que, caso o objeto ainda não tenha sido encontrado e nenhum sensor está direcionado para uma posição possível X_j, o *status* da variável V_{Xj} referente a esta posição não é alterado. Vale destacar que estas linhas (2 e 3) definem o mecanismo de memória.

Na quarta linha da fórmula φ_t^{Ob} é definido que, se o objeto ainda não foi encontrado e nenhum sensor está apontado para uma posição X_j, ou caso ela já tenha sido visitada, o objeto não poderá, no próximo passo, ser encontrado nesta posição. Isto porque, se não existe nenhum sensor apontado para esta célula, o sistema não pode detectar o objeto nela. Por sua vez, se o objeto ainda não foi encontrado e a célula já foi visitada, então já se sabe que ele não está nesta posição.

Por último, na linha 5 é definida uma restrição que garante que o objeto sempre será encontrado. Para isto, é descrito que, se no próximo passo todas as células possíveis terão sido visitadas, então, no próximo passo a posição do objeto não poderá mais ser desconhecida ($Ob \neq 0$). Isso significa que, se a última posição em que objeto pode estar será visitada no próximo passo, então o objeto deverá, obrigatoriamente, ser encontrado nela.

Na equação 5.20 é descrita a dinâmica do OPD presente no espaço de estado mostrado na figura 11(c), onde o objeto pode estar nas posições 7 ou 9.

$$
\begin{aligned}
&\varphi_i^{Ob} : (Ob = 0) \wedge (V_7 = nv) \wedge (V_9 = nv) \\
&\varphi_t^{Ob} : \begin{cases}
(Ob \neq 0) \rightarrow \{[(\bigcirc Ob) = Ob)] \wedge \bigcirc [(V_7 = v) \wedge (V_9 = v)]\} \\
\bigwedge (Ob = 0) \wedge \neg [(s_1 = 7) \vee (s_2 = 7)] \rightarrow [(\bigcirc V_7) = V_7] \\
\bigwedge (Ob = 0) \wedge \neg [(s_1 = 9) \vee (s_2 = 9)] \rightarrow [(\bigcirc V_9) = V_9] \\
\bigwedge (Ob = 0) \wedge [(s_1 = 7) \vee (s_2 = 7)] \rightarrow \bigcirc (V_7 = v) \\
\bigwedge (Ob = 0) \wedge [(s_1 = 9) \vee (s_2 = 9)] \rightarrow \bigcirc (V_9 = v) \\
\bigwedge (Ob = 0) \wedge \{\neg [(s_1 = 7) \vee (s_2 = 7)] \vee (V_7 = v)\} \rightarrow \bigcirc (Ob \neq 7) \\
\bigwedge (Ob = 0) \wedge \{\neg [(s_1 = 9) \vee (s_2 = 9)] \vee (V_9 = v)\} \rightarrow \bigcirc (Ob \neq 9) \\
\bigwedge \bigcirc [(V_7 = v) \wedge (V_9 = v)] \rightarrow \bigcirc (Ob \neq 0)
\end{cases} \\
&\varphi_g^{Ob} : \Box \Diamond true
\end{aligned}
\tag{5.20}
$$

5.4 SITUAÇÕES DE FALHA

Neste trabalho, também foram considerados casos onde os robôs do sistema podem sofrer falhas. Deste modo, foi possível avaliar o método proposto em Piterman *et al.*(2006) [56] quanto ao tratamento de situações de falta. A seguir são apresentados os modelos de robôs com possibilidade de falha e do mecanismo utilizado para simular a ocorrência e detecção de uma falha, chamado de *simulador de falha*, que se assume como parte do ambiente.

5.4.1 Robô com possibilidade de falha

Para representar a possibilidade de falha num robô, foi utilizada uma variável st_i que indica se um robô R_{Si} está ativo ($st_i = A$) ou em falha ($st_i = F$). Além disso, também foi modelado (como parte do ambiente) um mecanismo que simula a ocorrência de uma falha. Assim, o sistema percebe, através da leitura da variável $falha$, que houve um falha e em qual robô ela ocorreu.

O modelo de um robô com possibilidade de falha ($\varphi^{RF_{Si}}$), para o caso com observabilidade global, foi obtido pela conjunção do modelo do robô sem falha ($\varphi^{R_{Si}}$) com o modelo da possibilidade de falha (φ^{pf_i}), como mostrado na equação 5.21.

$$\varphi^{RF_{Si}} = \varphi^{R_{Si}} \wedge \varphi^{pf_i} \tag{5.21}$$

Para o caso parcial ($\varphi^{RFp_{Si}}$), o modelo é dado pela equação 5.22, onde é representada a conjunção dos modelos do robô sem falha, do sensor e do modelo da possibilidade de falha.

$$\varphi^{RFp_{Si}} = \varphi^{R_{Si}} \wedge \varphi^{s_i} \wedge \varphi^{pf_i} \tag{5.22}$$

Quanto ao modelo da possibilidade de falha, este foi descrito considerando que o robô só pode se movimentar caso esteja ativo ($st_i = A$). Além disso, também é definido que se o robô R_{Si} estiver ativo e houver uma falha associada a ele ($falha = i$), seu *status* é alterado para *em falha* ($st_i = F$). Caso contrário, o valor de st_i não muda. Assim, o modelo da possibilidade de falha no robô R_{Si} é descrito como mostrado na equação 5.23.

$$
\begin{array}{l}
\varphi_i^{pf_i} : (st_i = A) \\
\varphi_t^{pf_i} : \left\{ \begin{array}{l} (st_i \neq A) \rightarrow [(\bigcirc R_{Si}) = R_{Si}] \\ \bigwedge\ [(st_i = A) \wedge \bigcirc (falha = i)] \rightarrow \bigcirc (st_i = F) \\ \bigwedge\ \neg [(st_i = A) \wedge \bigcirc (falha = i)] \rightarrow [(\bigcirc st_i) = st_i] \end{array} \right. \\
\varphi_g^{pf_i} : \Box \Diamond true
\end{array}
\tag{5.23}
$$

onde $falha$ é uma variável que tem seu valor alterado por um elemento que está associado ao ambiente, o simulador de falha.

5.4.2 Simulador de falha

Como foi discutido no início desta seção, a fim de representar a ocorrência e detecção de uma falha, foi proposta a utilização de um simulador de falha. Entretanto, vale ressaltar que o objetivo neste trabalho foi o tratamento da falta no sistema e não a detecção ou o de tratamento da falha do robô.

De forma geral, o simulador de falha foi utilizado para representar que a qualquer momento da execução pode ocorrer uma falha num dos robôs do sistema. Assim, foi assumido que este mecanismo deve ser descrito como parte do ambiente, para que, agindo como um adversário do sistema, force que os controladores gerados considerem todas as possibilidades de ocorrência de falha.

Além disso, também foi considerado que o projetista pode definir um número máximo de faltas (F_{max}) com que seu sistema deve ser capaz de lidar. Assim, o modelo do simulador de falhas foi descrito como mostrado na equação 5.24

$$
\begin{array}{l}
\varphi_i^F : (falha = 0) \wedge (nF = 0) \\
\varphi_t^F : \left\{ \begin{array}{l} \neg \bigcirc (falha \neq 0) \rightarrow [(\bigcirc nF) = nF] \\ \bigwedge\ \bigcirc (falha \neq 0) \rightarrow [(\bigcirc nF) = nF + 1] \\ \bigwedge\ [(falha = 0) \wedge (nF < F_{max})] \rightarrow \bigcirc \left[(falha = 0) \vee \left(\bigvee_{i=1}^{N_R} (falha = i) \right) \right] \\ \bigwedge_{N_R}\ \neg [(falha = 0) \wedge (nF < F_{max})] \rightarrow \bigcirc (falha = 0) \\ \bigwedge_{i=1}^{N_R}\ \{(st_i = F) \rightarrow \bigcirc (falha \neq i)\} \end{array} \right. \\
\varphi_g^F : \Box \Diamond true
\end{array}
\tag{5.24}
$$

onde nF é uma variável utilizada para contar o número de falhas ocorridas e $falha$ é uma variável que indica em qual robô está ocorrendo a falha.

A dinâmica definida na equação φ_t^F pode ser dividida em duas partes: contador e ocorrência de falhas. Nas duas primeiras linhas da fórmula é descrito o contador, definindo que ao ocorrer uma falha ($falha \neq 0$), a variável

nF é incrementada em 1. Caso contrário, nF não é alterada. Na segunda parte da dinâmica é definido como o valor da variável $falha$ pode ser alterado, a fim de indicar a falha num robô. Nas linhas 3 e 4 é descrito que a variável só pode indicar a ocorrência de falha num robô ($falha \neq 0$) se não estiver sendo indicada outra falha e se o número máximo de faltas não tiver sido excedido ($nF < F_{max}$). Por último, é declarado na linha 5 que, se um robô R_{Si} não estiver ativo ($st_i = F$), não pode ser indicada uma falha neste robô.

Para ilustrar a modelagem de um simulador de falhas, foi considerado um exemplo onde o sistema é formado por dois robôs. Na equação 5.25 é definido o modelo do simulador para este caso.

$$
\begin{aligned}
&\varphi_i^F : (falha = 0) \\
&\varphi_t^F : \begin{cases}
\neg \bigcirc (falha \neq 0) \rightarrow [(\bigcirc nF) = nF] \\
\bigwedge \bigcirc (falha \neq 0) \rightarrow [(\bigcirc nF) = nF + 1] \\
\bigwedge [(falha = 0) \wedge (nF < 1)] \rightarrow \\
\qquad \bigcirc [(falha = 0) \vee (falha = 1) \vee (falha = 2)] \\
\bigwedge \neg [(falha = 0) \wedge (nF < 1)] \rightarrow \bigcirc (falha = 0) \\
\bigwedge (st_1 \neq A) \rightarrow \bigcirc (falha \neq 1) \\
\bigwedge (st_2 \neq A) \rightarrow \bigcirc (falha \neq 2)
\end{cases} \\
&\varphi_g^F : \Box \Diamond true
\end{aligned}
\tag{5.25}
$$

Sobre o exemplo descrito na equação 5.25 , vale destacar que o número máximo de falhas (F_{max}) assumido foi 1, uma vez que só existem dois robôs.

5.5 MODELAGEM DOS OBJETIVOS BÁSICOS DO SISTEMA

Quando se trabalha com coordenação de SMRs, várias são as possibilidades de tarefas a serem executadas por estes sistemas, conforme foi visto nos capítulos 2 e 4. Todavia, existem alguns tipos básicos de especificações que, combinados, são suficientemente gerais para representar a maioria dos comportamentos desejados para os sistemas. Esses tipos básicos são: alcançabilidade, cobertura e sequenciamento.

No método proposto em [56] é utilizado apenas um fragmento de LTL para descrever a especificações de um sistema, sendo possível usar apenas expressões no formato de condições de justiça, mostrado na equação 5.26.

$$\phi \;=\; \Box\Diamond E \tag{5.26}$$

onde E é uma propriedade que será satisfeita sempre no futuro.

Assim, foi proposta neste trabalho a utilização de variáveis indicadoras para representar especificações de alcançabilidade ($\Diamond E$) a partir de fórmulas de justiça. Esse tipo de variável é inicializada com *false* e sua dinâmica é modelada de forma que, após receber *true*, seu valor não pode mais ser alterado.

Proposição 5.1. *Dado que V_{ind} é uma variável indicadora, a equivalência mostrada na equação 5.27 é válida.*

$$\Box\Diamond(V_{ind} = true) \;\Leftrightarrow\; \Diamond(V_{ind} = true) \tag{5.27}$$

Demonstração. Sejam as especificações $\Diamond E$ e $\Box\Diamond E$. Da definição dos operadores de lógica temporal $\Diamond$ e $\Box\Diamond$, se a fórmula $\Diamond E$ é satisfeita, isso indica que em algum momento no futuro a propriedade E é válida. Por sua vez, se $\Box\Diamond E$ é satisfeita, temos que E ocorre, no futuro, infinitas vezes. Assim, se $\Box\Diamond E$ é satisfeita, então $\Diamond E$ também é satisfeita. Logo, tem-se que:

$$\Box\Diamond E \;\Rightarrow\; \Diamond E \tag{5.28}$$

Porém, como o cumprimento da fórmula $\Diamond E$ não implica que $\Box\Diamond E$ também será satisfeita, não se pode dizer que, no caso geral, $\Diamond E \;\Leftrightarrow\; \Box\Diamond E$.

Contudo, devido ao fato da variável indicadora V_{ind} passar a ser sempre *true*, após receber esse valor, se a propriedade $\Diamond(V_{ind} = true)$ é satisfeita, então $\Box\Diamond(V_{ind} = true)$ também o é. Logo:

$$\Diamond(V_{ind} = true) \Rightarrow \Box\Diamond(V_{ind} = true) \tag{5.29}$$

Substituindo a propriedade E por $(V_{ind} = true)$ temos que:

$$\Box\Diamond(V_{ind} = true) \Rightarrow \Diamond(V_{ind} = true) \tag{5.30}$$

Assim, das equações (5.29) e (5.30), temos que:

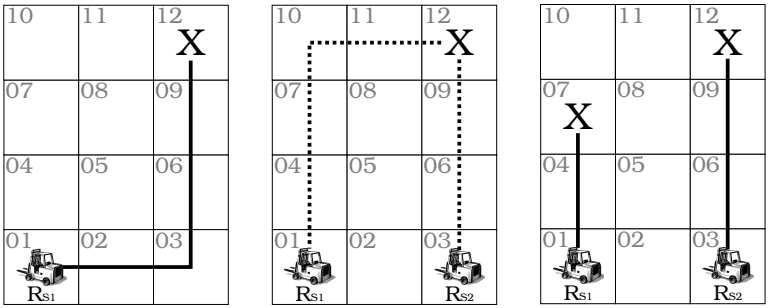
$$\Diamond(V_{ind} = true) \Leftrightarrow \Box\Diamond(V_{ind} = true)$$

Por tanto, a equivalência é válida. □

Devido à equivalência apresentada na equação 5.27, é possível utilizar variáveis indicadoras para representar os tipos básicos de objetivos, bastando para isso que a mudança do *status* da variável para *true* esteja vinculada ao cumprimento do objetivo correspondente. A seguir são discutidas em detalhes as descrições de cada tipo de objetivo.

5.5.1 Alcançabilidade

Alcançabilidade é o tipo mais comum em aplicações de robótica móvel, sendo utilizado para definir a posição de destino de um agente do sistema. Nos SMRs tratados neste trabalho, foram consideradas três tipos de alcançabilidade: simples, suficiente e em conjunto. Na figura 12 são apresentados exemplos de cada um dos tipos de alcançabilidade.



(a) Simples (b) Suficiente (c) Em conjunto

Figura 12: Tipos de alcançabilidade

Alcançabilidade simples

Na alcançabilidade simples, a posição de destino deve ser alcançada por um robô específico e de forma independente em relação aos outros objetivos do sistema. No caso apresentado na figura 12(a), a especificação é que o robô R_{S1} deve alcançar a posição 12.

Para o caso do objetivo de alcançabilidade simples, sua modelagem é feita como mostrado na equação 5.31.

$$\begin{array}{l} \varphi_i^{alcS} : (alcS = false) \\ \varphi_t^{alcS} : \left\{ \begin{array}{l} \neg \bigcirc (R_{Si} = D) \rightarrow [(\bigcirc alcS) = alcS] \\ \bigwedge \bigcirc (R_{Si} = D) \rightarrow \bigcirc (alcS = true) \end{array} \right. \\ \varphi_g^{alcS} : \Box \Diamond (alcS = true) \end{array} \tag{5.31}$$

A dinâmica da variável indicadora *alcS* (φ_t^{alcS}), associada ao objetivo do robô R_{Si} alcançar a posição *D*, pode ser entendida da seguinte forma. Se o R_{Si} não está na posição *D*, *alcS* não pode ter seu valor alterado. Por sua vez, na segunda linha é descrito que, se no próximo passo a posição *D* for alcançada por R_{Si}, será atribuído o valor *true* à variável *alcS*.

Para o exemplo mostrado na figura 12(a), o modelo do objetivo básico de alcançabilidade simples é dada por:

$$\begin{array}{l} \varphi_i^{alcS} : (alcS = false) \\ \varphi_t^{alcS} : \left\{ \begin{array}{l} \neg \bigcirc (R_{S1} = 12) \rightarrow [(\bigcirc alcS) = alcS] \\ \bigwedge \bigcirc (R_{S1} = 12) \rightarrow \bigcirc (alcS = true) \end{array} \right. \\ \varphi_g^{alcS} : \Box \Diamond (alcS = true) \end{array}$$

Alcançabilidade suficiente

Numa especificação de alcançabilidade suficiente, é definido um conjunto de robôs que podem ser utilizados para alcançar uma determinada posição. Ressaltando que o objetivo é considerado como realizado se qualquer um dos robôs alcançar a posição de destino. Na figura 12(b), por exemplo, a especificação é satisfeita se um dos robôs (R_{S1} ou R_{S2}) alcançar a posição 12.

Para o caso das especificações de alcançabilidade suficiente, o modelo do objetivo é descrito como mostrado na equação 5.32.

$$\begin{array}{l} \varphi_i^{alcC} : (alcC = false) \\ \varphi_t^{alcC} : \left\{ \begin{array}{l} \neg \bigcirc \left(\bigvee_{i=1}^{N} (R_{Si} = D) \right) \rightarrow [(\bigcirc alcC) = alcC] \\ \bigwedge \bigcirc \left(\bigvee_{i=1}^{N} (R_{Si} = D) \right) \rightarrow \bigcirc (alcC = true) \end{array} \right. \\ \varphi_g^{alcC} : \Box \Diamond (alcC = true) \end{array} \tag{5.32}$$

onde *N* é o número de robôs que podem ser utilizados para satisfazer a especificação de alcançar a posição *D*.

Na fórmula φ_t^{alcC} é indicado que, se nenhum robô do sistema estiver na posição *D*, a variável *alcC* permanece inalterada. Uma vez que um dos robôs alcança a célula de destino *D*, *alcC* recebe *true*, indicando que a especificação foi satisfeita.

Para o exemplo mostrado na figura 12(b), o objetivo de alcançabilidade suficiente é modelado por:

$$\varphi_i^{alcC} : (alcC = false)$$
$$\varphi_t^{alcC} : \begin{cases} \neg \bigcirc [(R_{S1} = 12) \vee (R_{S2} = 12)] \rightarrow [(\bigcirc alcC) = alcC] \\ \bigwedge \bigcirc [(R_{S1} = 12) \vee (R_{S2} = 12)] \rightarrow \bigcirc (alcC = true) \end{cases}$$
$$\varphi_g^{alcC} : \Box \Diamond (alcC = true)$$

Alcançabilidade em conjunto

Na alcançabilidade em conjunto, o objetivo é posicionar um subconjunto de agentes do sistema em células específicas. Deste modo, a especificação só é cumprida quando todos os agentes se encontram em suas respectivas posições, simultaneamente. No exemplo mostrado na figura 12(c), a especificação só é satisfeita quando os robôs R_{S1} e R_{S2} estão nas posições 7 e 12, respectivamente, no mesmo instante.

No caso do objetivo de alcançabilidade em conjunto, o modelo é descrito como mostrado na equação 5.33.

$$\varphi_i^{alcA} : (alcA = true)$$
$$\varphi_t^{alcA} : \begin{cases} \neg \bigcirc \left(\bigwedge_{i=1}^{N} (R_{Si} = D_i) \right) \rightarrow [(\bigcirc alcA) = alcA] \\ \bigwedge \bigcirc \left(\bigwedge_{i=1}^{N} (R_{Si} = D_i) \right) \rightarrow \bigcirc (alcA = true) \end{cases} \tag{5.33}$$
$$\varphi_g^{alcA} : \Box \Diamond (alcA = true)$$

onde N é o número de robôs que compõem o conjunto.

Na fórmula φ_t^{alcA} é indicado que, se algum dos N robôs que formam o arranjo não estiver em sua respectiva posição D_i, a variável $alcA$ permanece inalterada. Uma vez que todos os robôs estão em suas células de destino, $alcA$ recebe $true$, indicando que a especificação foi satisfeita.

Para o exemplo mostrado na figura 12(c), o objetivo de alcançabilidade em conjunto é modelado por:

$$\varphi_i^{alcA} : (alcA = false)$$
$$\varphi_t^{alcA} : \begin{cases} \neg \bigcirc [(R_{S1} = 7) \wedge (R_{S2} = 12)] \rightarrow [(\bigcirc alcA) = alcA] \\ \bigwedge \bigcirc [(R_{S1} = 7) \wedge (R_{S2} = 12)] \rightarrow \bigcirc (alcA = true) \end{cases}$$
$$\varphi_g^{alcA} : \Box \Diamond (alcA = true)$$

5.5.2 Cobertura

Quando a especificação do sistema é formada por um conjunto de objetivos básicos independentes, ou seja, a realização de um objetivo não de-

pende do cumprimento de outro, tem-se uma especificação de cobertura. Entretanto, quando a realização de um dos objetivos é suficiente para satisfazer a especificação, esta é chamada de *especificação de cobertura de objetivos suficientes*. Por outro lado, quando é necessário que todos os objetivos sejam satisfeitos, tem-se um a *especificação de cobertura de objetivos necessários*.

Vale destacar ainda que, a composição destes dois tipos de cobertura pode ser utilizada para representar especificações onde existem conjuntos de objetivos que são suficientes ou necessários. Por exemplo, em situações onde é suficiente a realização de dois objetivos básicos, este tipo de abordagem poderia ser utilizada. A seguir são descritos os dois tipos de especificações de cobertura.

Cobertura com objetivos suficientes

Dado um sistema onde existam várias especificações $\varphi^1, \cdots, \varphi^N$. Se, para que o objetivo do sistema seja satisfeito, basta que apenas uma das especificações φ^i seja satisfeita (ou seja, cada especificação é suficiente, mas não necessária), tem-se uma especificação de cobertura com objetivos suficientes.

Para este tipo de especificação, o modelo é descrito como mostrado na equação 5.34.

$$\begin{array}{l} \varphi_i^{cSuf} : (cSuf = false) \\ \varphi_t^{cSuf} : \left\{ \begin{array}{l} \neg \bigcirc \left(\bigvee_{i=1}^{N} (\varphi^i = true) \right) \rightarrow [(\bigcirc cSuf) = cSuf] \\ \bigwedge \bigcirc \left(\bigvee_{i=1}^{N} (\varphi^i = true) \right) \rightarrow \bigcirc (cSuf = true) \end{array} \right. \\ \varphi_g^{cSuf} : \Box \Diamond (cSuf = true) \end{array} \tag{5.34}$$

onde N é o número de objetivos que compõem a especificação e $cSuf$ é a variável que indica que a especificação de cobertura foi realizada.

Sobre as especificações φ^i, vale destacar que elas podem ser fórmulas lógicas ou outros objetivos. No segundo caso, a fórmula $\varphi^i = true$ corresponde a $V_{ind}^i = true$, onde V_{ind}^i é a variável indicadora associada ao cumprimento do objetivo φ^i. Vale destacar também que o objetivo de alcançabilidade suficiente é um caso particular deste tipo especificação, onde os objetivos são os de cada robô chegar a posição de destino D.

Considerando o caso em que o objetivo final é que o robô R_{S1} esteja na posição 11 ou que uma das especificações mostradas nas figuras 12(b) e 12(c) seja satisfeita, tem-se uma especificação de cobertura com objetivos suficientes, dada por:

$$\begin{array}{l}\varphi_i^{cSuf} : (cSuf = false)\\ \varphi_t^{cSuf} : \left\{\begin{array}{l} \neg \bigcirc [(R_{S1} = 11) \vee (alcC = true) \vee (alcA = true)] \rightarrow [(\bigcirc cSuf) = cSuf]\\ \bigwedge \bigcirc [(R_{S1} = 11) \vee (alcC = true) \vee (alcA = true)] \rightarrow \bigcirc (cSuf = true)\end{array}\right.\\ \varphi_g^{cSuf} : \Box\Diamond (cSuf = true)\end{array}$$

Cobertura com objetivos necessários

Dado um sistema onde existem várias especificações $\varphi^1, \cdots, \varphi^N$. Quando o objetivo do sistema é que, ao final, todas as especificações φ^i sejam realizadas (ou seja, cada especificação é necessária), tem-se uma especificação de cobertura com objetivos necessários.

Para este caso, a modelagem é feita como mostrado na fórmula 5.35.

$$\begin{array}{l}\varphi_i^{cNes} : (cNes = false)\\ \varphi_t^{cNes} : \left\{\begin{array}{l} \neg \bigcirc \left(\bigwedge_{i=1}^{N} (\varphi^i = true) \right) \rightarrow [(\bigcirc cNes) = cNes]\\ \bigwedge \bigcirc \left(\bigwedge_{i=1}^{N} (\varphi^i = true) \right) \rightarrow \bigcirc (cNes = true)\end{array}\right.\\ \varphi_g^{cNes} : \Box\Diamond (cNes = true)\end{array} \tag{5.35}$$

onde *cNes* é a variável que indica que a especificação de cobertura de objetivos necessários foi realizada.

Assim como ocorreu no caso de cobertura com objetivos suficientes, as especificações podem ser fórmulas lógicas ou outros objetivos. Quando todos os itens φ^i são descritos através de variáveis indicadoras, não há necessidade de declarar uma especificação de cobertura com objetivos necessários, podendo ser declaradas as condições de justiça de cada objetivo diretamente na condição de justiça do sistema (φ_g^s).

Considerando o caso em que, além das especificações φ^{alcC} e φ^{alcA}, o robô R_{S1} deve estar na posição 11, o modelo da especificação de cobertura com objetivos necessários é dado por:

$$\begin{array}{l}\varphi_i^{cNes} : (cNes = false)\\ \varphi_t^{cNes} : \left\{\begin{array}{l} \neg \bigcirc [(R_{S1} = 11) \wedge (alcC = true) \wedge (alcA = true)] \rightarrow [(\bigcirc cNes) = cNes]\\ \bigwedge \bigcirc [(R_{S1} = 11) \wedge (alcC = true) \wedge (alcA = true)] \rightarrow \bigcirc (cNes = true)\end{array}\right.\\ \varphi_g^{cNes} : \Box\Diamond (cNes = true)\end{array}$$

5.5.3 Sequenciamento

Quando o objetivo final do sistema é dividido em várias especificações e estas devem ser realizadas respeitando uma ordem pré-determinada (por exemplo, φ^{i+1} só pode ser realizada depois que φ^{i} tenha sido), tem-se uma especificação de sequenciamento.

Neste caso, é necessário alterar ligeiramente a dinâmica da variável indicadora de cada especificação, de modo que, para a variável receber o valor *true* é necessário que o objetivo de índice menor tenha sido cumprido. Assim, a modelagem de um objetivo φ^{j}é feita como mostrado na equação5.36.

$$\begin{array}{l} \varphi_i^j : (V_{ind}^j = false) \\ \varphi_t^j : \left\{ \begin{array}{l} \neg[(V_{ind}^{j-1} = true) \wedge (\varphi^j = true)] \rightarrow [(\bigcirc V_{ind}^j) = V_{ind}^j] \\ \wedge[(V_{ind}^{j-1} = true) \wedge (\varphi^j = true)] \rightarrow \bigcirc(V_{ind}^j = true) \end{array} \right. \\ \varphi_g^j : \Box\Diamond (V_{ind}^j = true) \end{array} \tag{5.36}$$

onde V_{ind}^j é a variável indicadora associada ao cumprimento do objetivo φ^j e a fórmula $(\varphi^j = true)$ indica que a condição, sem levar em conta a ordem, em que o objetivo φ^j é realizado está sendo satisfeita.

Na fórmula φ_t^j é definido que, se o objetivo predecessor (φ^{j-1}) não tiver sido realizado ou se a condição em que φ^j é cumprido não estiver sendo satisfeita, a variável indicadora (V_{ind}^j) associada a esta especificação não é alterada. Caso contrário, V_{ind}^j recebe $true$.

Considerando o caso de um robô que deve se deslocar pelo espaço de trabalho a fim de alcançar a posição 12 e, em seguida, alcançar a 3, tem-se uma especificação do sistema composta por dois objetivos de alcançabilidade simples em sequência. Para este caso, os modelos são descritos como mostrado nas equações abaixo.

$$\begin{array}{l} \varphi_i^{alcS_1} : (alcS_1 = false) \\ \varphi_t^{alcS_1} : \left\{ \begin{array}{l} \neg[(true) \wedge \bigcirc(R_{S1} = 12)] \rightarrow [(\bigcirc alcS_1) = alcS_1] \\ \bigwedge[(true) \wedge \bigcirc(R_{S1} = 12)] \rightarrow \bigcirc(alcS_1 = true) \end{array} \right. \\ \varphi_g^{alcS_1} : \Box\Diamond true \end{array}$$

$$\begin{array}{l} \varphi_i^{alcS_2} : (alcS_2 = false) \\ \varphi_t^{alcS_2} : \left\{ \begin{array}{l} \neg[(alcS_1 = true) \wedge \bigcirc(R_{S1} = 3)] \rightarrow [(\bigcirc alcS_2) = alcS_2] \\ \bigwedge[(alcS_1 = true) \wedge \bigcirc(R_{S1} = 3)] \rightarrow \bigcirc(alcS_2 = true) \end{array} \right. \\ \varphi_g^{alcS_2} : \Box\Diamond (alcS_2 = true) \end{array}$$

Sobre a descrição do primeiro objetivo, vale destacar que não há um predecessor, deste modo, só é levado em conta a condição de realização deste

objetivo. Além disso, a condição de justiça numa especificação de sequenciamento é a associada ao último objetivo da sequência. No caso acima, usado como exemplo, a condição de justiça seria $\varphi_g^{alcS_2}$. Isso ocorre porque, se o último objetivo foi satisfeito, num sequência, isso implica que todos os predecessores também foram realizados.

5.6 METODOLOGIA PARA COMPOSIÇÃO DE ELEMENTOS DE MODELAGEM

Neste trabalho, foi proposta uma metodologia para descrição de SMRs em estruturas de jogos baseada em três passos: identificação dos elementos que compõem a aplicação, obtenção do modelo de cada elemento isoladamente e conjunção dos modelos obtidos. A seguir, estas etapas são descritas.

1. Identificação dos elementos. Inicialmente, os componentes e as tarefas de uma aplicação são identificados, sendo definido também se eles estão associados ao ambiente ou ao sistema.

2. Obtenção do modelo de cada elemento. Uma vez identificados os componentes de uma aplicação, o modelo de cada um deles é descrito de forma isolada.

3. Conjunção dos modelos. Após modelados todos os elementos que compõem uma aplicação, os modelos do ambiente e do sistema (que definem a estrutura de jogo) são obtidos pela conjunção dos elementos associados a eles.

Como foi discutido no capítulo 3, no método proposto em Piterman *et al.* (2006) o ambiente e o sistema são descritos por $\varphi^e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$ e $\varphi^s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$, respectivamente. Nessas fórmulas, os índices i, t e g indicam que a fórmula descreve a condição inicial, dinâmica e condição de justiça, respectivamente.

Para o ambiente, como descrito no passo 3, o modelo é obtido pela conjunção das fórmulas que descrevem cada um dos componentes associados a ele, como mostrado na equação 5.37.

$$\begin{bmatrix} \varphi_i^e : \bigwedge_{j=1}^{N_{EA}} \varphi_i^{ElAmb_j} \\ \varphi_t^e : \bigwedge_{j=1}^{N_{EA}} \varphi_t^{ElAmb_j} \\ \varphi_g^e : \bigwedge_{j=1}^{N_{EA}} \varphi_g^{ElAmb_j} \end{bmatrix} \tag{5.37}$$

onde N_{EA} é o número de elementos do ambiente e as fórmulas $\varphi_i^{ElAmb_j}$, $\varphi_t^{ElAmb_j}$ e $\varphi_g^{ElAmb_j}$ correspondem a descrição da condição inicial, dinâmica e hipótese realizada sobre o j-ésimo elemento do ambiente.

De forma similar ao que foi feito para o ambiente, o modelo do sistema é obtido pela conjunção das fórmulas que descrevem cada um dos seus componentes, como mostrado na equação 5.38. Entretanto, neste trabalho foi proposta a utilização de uma única condição de justiça para o sistema, indicando que sua especificação é que, ao final da execução, todos os objetivos sejam satisfeitos. Assim, a condição de justiça do sistema é declarada como mostrado na fórmula φ_g^s da equação 5.38.

$$\begin{aligned}
\varphi_i^s &: \left[\left(\bigwedge_{j=1}^{N_R} \varphi_i^{R_{Sj}} \right) \wedge \varphi_i^{NCol} \wedge \left(\bigwedge_{k=1}^{N_{obj}} \varphi_i^{obj_k} \right) \right] \\
\varphi_t^s &: \left[\left(\bigwedge_{j=1}^{N_R} \varphi_t^{R_{Sj}} \right) \wedge \varphi_t^{NCol} \wedge \left(\bigwedge_{k=1}^{N_{obj}} \varphi_t^{obj_k} \right) \right] \\
\varphi_g^s &: \Box\Diamond \left[(NCol = true) \wedge \left(\bigwedge_{k=1}^{N_{obj}} (V_{ind}^{obj_k} = true) \right) \right]
\end{aligned} \tag{5.38}$$

onde $\varphi^{R_{Si}}$ corresponde à descrição do i-ésimo robô, φ^{NCol} corresponde a especificação de não colisão entre robôs, φ^{Obj_k} corresponde a descrição do k-ésimo objetivo básico, $V_{ind}^{Obj_k}$ é a variável indicadora associada à realização deste objetivo e N_{obj} é o número de objetivos básicos.

É importante destacar que, se tivessem sido declaradas condições de justiça ($\Box\Diamond$) para cada objetivo do sistema, o controlador gerado implementaria várias estratégias, sendo uma para cada objetivo a ser satisfeito. Dessa forma, apenas um objetivo seria tratado de cada vez, sendo que, após sua realização, o controlador mudaria de estratégia para atender ao próximo objetivo.

Entretanto, este tipo de solução não é eficiente e, assim, optou-se por considerar apenas uma única condição de justiça. Vale destacar que esta forma de modelar a especificação do sistema não interfere, por exemplo, na ordem de realização dos objetivos de uma especificação de sequência. Isso porque é definido, através da dinâmica da variável indicadora associada a cada objetivo, que ele só pode ser satisfeito após seu objetivo predecessor ter sido realizado.

Exemplo

Para ilustrar o processo de modelagem de uma aplicação, será apresentada em detalhes a descrição da planta mostrada na figura 11(a), para o caso com observabilidade global. Neste exemplo, o espaço de trabalho possui 12 células e um obstáculo fixo na posição 8. Por sua vez, o ambiente é composto por duas portas, situadas nas posições 7 e 9 e que chamaremos de P_1 e P_2, respectivamente. Já o sistema possui dois robôs, R_{S1} e R_{S2}, inicialmente nas

posições 1 e 3. Como especificação do sistema, foram adotados os objetivos (de alcançabilidade simples) do robô R_{S1} deve atingir a posição 12 e o R_{S2} a célula 10.

Modelagem do ambiente

Nas equações 5.39, 5.40 são apresentadas a descrição da condição inicial, dinâmica e condição de justiça das portas P_1 e P_2, que compõem o ambiente.

$$\begin{array}{l} \varphi_i^{P1} : (P_1 = A) \\ \varphi_t^{P1} : [(R_{S1} = 7) \vee (R_{S2} = 7)] \rightarrow \bigcirc (P_1 = A) \\ \varphi_g^{P1} : \Box\Diamond (P_1 \neq F) \end{array} \tag{5.39}$$

$$\begin{array}{l} \varphi_i^{P2} : (P_2 = A) \\ \varphi_t^{P2} : [(R_{S1} = 9) \vee (R_{S2} = 9)] \rightarrow \bigcirc (P_2 = A) \\ \varphi_g^{P2} : \Box\Diamond (P_2 \neq F) \end{array} \tag{5.40}$$

Uma vez que o ambiente é composto apenas pelas portas P_1 e P_2, seu modelo é descrito como mostrado na equação 5.41

$$\begin{array}{l} \varphi_i^{e} : (\varphi_i^{P1} \wedge \varphi_i^{P2}) \\ \varphi_t^{e} : (\varphi_t^{P1} \wedge \varphi_t^{P2}) \\ \varphi_g^{e} : (\varphi_g^{P1} \wedge \varphi_g^{P2}) \end{array} \tag{5.41}$$

Modelagem dos robôs sistema

Para os dois robôs do sistema, os modelos são descritos nas equações 5.42 e 5.43.

$$\begin{array}{l} \varphi_i^{R_{S1}} : (R_{S1} = 1) \\ \varphi_t^{R_{S1}} : \left\{ \begin{array}{l} (R_{S1} = 1) \rightarrow \bigcirc [(R_{S1} = 1) \vee (R_{S1} = 2) \vee (R_{S1} = 4)] \\ \bigwedge (R_{S1} = 2) \rightarrow \bigcirc [(R_{S1} = 2) \vee (R_{S1} = 1) \vee (R_{S1} = 3) \vee (R_{S1} = 5)] \\ \quad \vdots \\ \bigwedge (R_{S1} = 12) \rightarrow \bigcirc [(R_{S1} = 12) \vee (R_{S1} = 11) \vee (R_{S1} = 9)] \\ \bigwedge \bigcirc (P_1 \neq A) \rightarrow \bigcirc (R_{S1} \neq 7) \\ \bigwedge \bigcirc (P_2 \neq A) \rightarrow \bigcirc (R_{S1} \neq 9) \end{array} \right. \\ \varphi_g^{R_{S1}} : \Box\Diamond true \end{array} \tag{5.42}$$

$$\begin{array}{l} \varphi_i^{R_{S2}} : (R_{S2} = 3) \\ \varphi_t^{R_{S2}} : \left\{ \begin{array}{l} (R_{S2} = 1) \rightarrow \bigcirc [(R_{S2} = 1) \vee (R_{S2} = 2) \vee (R_{S2} = 4)] \\ \bigwedge (R_{S2} = 2) \rightarrow \bigcirc [(R_{S2} = 2) \vee (R_{S2} = 1) \vee (R_{S2} = 5) \vee (R_{S2} = 3)] \\ \quad \vdots \\ \bigwedge (R_{S2} = 12) \rightarrow \bigcirc [(R_{S2} = 12) \vee (R_{S2} = 11) \vee (R_{S2} = 9)] \\ \bigwedge \bigcirc (P_1 \neq A) \rightarrow \bigcirc (R_{S2} \neq 7) \\ \bigwedge \bigcirc (P_2 \neq A) \rightarrow \bigcirc (R_{S2} \neq 9) \end{array} \right. \\ \varphi_g^{R_{S2}} : \Box\Diamond true \end{array} \tag{5.43}$$

Modelagem do mecanismo de não colisão

Como o sistema é composto por mais de um robô, é necessário definir o mecanismo de não colisão entre os agentes. Na equação 5.44 é mostrada a descrição deste mecanismo.

$$\begin{array}{l} \varphi_i^{NCol} : (NCol = true) \\ \varphi_t^{NCol} : \left\{ \begin{array}{l} \neg [(R_{S1} = R_{S2}) \vee (\bigcirc R_{S1} = R_{S2}) \vee (R_{S1} = \bigcirc R_{S2})] \\ \quad \rightarrow [(\bigcirc NCol) = NCol] \\ \bigwedge [(R_{S1} = R_{S2}) \vee (\bigcirc R_{S1} = R_{S2}) \vee (R_{S1} = \bigcirc R_{S2})] \\ \quad \rightarrow \bigcirc (NCol = false) \end{array} \right. \\ \varphi_g^{NCol} : \Box \Diamond (NCol = true) \end{array} \tag{5.44}$$

Modelagem dos objetivos básicos do sistema

Por último, nas equações 5.45 e 5.46 são descritos os modelos dos objetivos (de alcançabilidade simples) de R_{S1} alcançar a célula 12 e R_{S2} chegar na posição 10.

$$\begin{array}{l} \varphi_i^{alcS1} : (alcS_1 = false) \\ \varphi_t^{alcS1} : \left\{ \begin{array}{l} \neg \bigcirc (R_{S1} = 12) \rightarrow [(\bigcirc alcS_1) = alcS_1] \\ \bigwedge \bigcirc (R_{S1} = 12) \rightarrow \bigcirc (alcS_1 = true) \end{array} \right. \\ \varphi_g^{alcS1} : (alcS_1 = true) \end{array} \tag{5.45}$$

$$\begin{array}{l} \varphi_i^{alcS2} : (alcS_2 = false) \\ \varphi_t^{alcS2} : \left\{ \begin{array}{l} \neg \bigcirc (R_{S2} = 10) \rightarrow [(\bigcirc alcS_2) = alcS_2] \\ \bigwedge \bigcirc (R_{S2} = 10) \rightarrow \bigcirc (alcS_2 = true) \end{array} \right. \\ \varphi_g^{alcS2} : (alcS_2 = true) \end{array} \tag{5.46}$$

Vale destacar que, como foi adotado que só existe uma condição de justiça associada ao sistema, os operadores $\Box\Diamond$ foram retirados da representação da condição de justiça dos objetivos básicos, sendo definido apenas no modelo do sistema (equação 5.47).

Modelagem do sistema

Uma vez definidos os modelos dos robôs do sistema, do mecanismo de não colisão e dos objetivos básicos, o modelo do sistema é descrito como mostrado na equação 5.47.

$$\varphi^s : \left\{ \begin{array}{l} \varphi_i^s = (\varphi_i^{R_{S1}} \wedge \varphi_i^{R_{S2}} \wedge \varphi_i^{NCol} \wedge \varphi_i^{alcS1} \wedge \varphi_i^{alcS2}) \\ \varphi_t^s = (\varphi_t^{R_{S1}} \wedge \varphi_t^{R_{S2}} \wedge \varphi_t^{NCol} \wedge \varphi_t^{alcS1} \wedge \varphi_t^{alcS2}) \\ \varphi_g^s = \Box\Diamond\ [(NCol = true) \wedge (alcS_1 = true) \wedge (alcS_2 = true)] \end{array} \right. \tag{5.47}$$

5.7 CONCLUSÃO

Neste capítulo foram apresentadas algumas ferramentas que implementam o método proposto em Piterman *et al.* (2006)[56]. Também neste capítulo, foram apresentadas as descrições dos elementos de modelagem e a metodologia utilizada para realizar a composição deles. Ao final do capítulo, foi apresentado um exemplo de modelagem utilizando os princípios propostos.

6 RESOLUÇÃO DE DIFERENTES PROBLEMAS DE COORDENAÇÃO

Neste capítulo são apresentadas as especificações para as classes de problemas discutidas no capítulo 4. Em seguida, exemplos de cada uma destas classes são resolvidos com base nos princípios de modelagem propostos no capítulo 5 e utilizando a implementação dos algoritmos realizada pelos próprios autores do método proposto em Piterman *et al.*(2006)[56].

Na sequência, são apresentados os experimentos realizados, bem como algumas informações relevantes sobre a realização destes. Por último, são discutidos os resultados obtidos na resolução dos exemplos de cada classe de problemas e nos experimentos realizados a partir dos controladores obtidos. Vale destacar ainda que foram considerados casos com ambientes compostos pelos elementos de modelagem descritos no capítulo 5.

6.1 PROBLEMA DE ALCANÇABILIDADE SIMPLES

Como já foi discutido no capítulo 5, especificações de alcançabilidade simples são recorrentes em aplicações de robótica móvel, de modo que vários dos comportamentos de interesse para SMRs podem ser descritos através da composição deste tipo básico de objetivos.

6.1.1 Modelagem das especificações

A especificação de um Problema de Alcançabilidade Simples (PAS) pode ser descrita através da conjunção de objetivos básicos de alcançabilidade simples. Assim, dado que $\varphi^{pAS} = \varphi_i^{pAS} \wedge \varphi_t^{pAS} \wedge \varphi_g^{pAS}$ é a fórmula que descreve a especificação deste tipo de problema, ela é definida como mostrado na equação 6.1.

$$\begin{aligned} \varphi_i^{pAS} &: \bigwedge_{j=1}^{N_{pAS}} \varphi_i^{alcS_j} \\ \varphi_t^{pAS} &: \bigwedge_{j=1}^{N_{pAS}} \varphi_t^{alcS_j} \\ \varphi_g^{pAS} &: \bigwedge_{j=1}^{N_{pAS}} (alcS_j = true) \end{aligned} \tag{6.1}$$

Na equação 6.1, as fórmulas φ_i^{pAS}, φ_t^{pAS} e φ_g^{pAS} descrevem a condição inicial, a dinâmica e a condição de justiça da especificação, respectivamente. N_{pAS} indica o número de objetivos de alcançabilidade simples que compõem a especificação e φ^{alcS_j} é a fórmula que descreve cada um deste objetivos.

Vale destacar a ausência do operador $\Box\Diamond$ na fórmula referente à condição de justiça (φ_g^{pAS}). Como foi discutido na seção 5.6, foi adotado que o sistema possui apenas uma condição de justiça. Assim, o operador $\Box\Diamond$ só é declarado uma vez (veja nas equações 5.47, no capítulo 5, e 6.3, definida na sequência deste capítulo).

6.1.2 Exemplo

Na seção 5.6, onde é apresentada a metodologia proposta, é utilizado um exemplo cuja especificação do sistema é formada por dois objetivos de alcançabilidade simples. Vale destacar que este é o exemplo utilizado nos trabalhos apresentados na dissertação de mestrado [54], desenvolvida no PPGEAS, e no artigo [55]. Assim, o exemplo utilizado na seção 5.6 também será resolvido para o caso com observabilidade parcial (caso tratado em [55, 54]) e os resultados serão utilizados para comparação com os obtidos nestes trabalhos.

Neste exemplo, é considerado um espaço de trabalho composto por 12 células, onde estão inseridos dois robôs do sistema, um ambiente composto por duas portas e um obstáculo fixo. Na figura 13 é apresentado este espaço de trabalho, onde já são indicadas as posições iniciais dos robôs, a localização das portas e do obstáculo fixo e as células a serem alcançadas pelos agentes do sistema.

Modelos

Para o exemplo mostrado na figura 13, os modelos do ambiente e do sistema, considerando a abordagem com observabilidade global já foram apresentados na seção 5.6, nas equações 5.41 e 5.47. Assim, eles não serão discutidos novamente neste capítulo.

Paro o caso em que é considerada observabilidade parcial, os modelos do ambiente (φ^e), composto pelas portas P_1 e P_2, e do sistema (φ^s) são descritos pelas equações 6.2 e 6.3, respectivamente. Nestas equações, φ^{P_1} e φ^{P_2} correspondem aos modelos das portas, $\varphi^{Rp_{S1}}$ e $\varphi^{Rp_{S2}}$ são os modelos dos dois robôs do sistema e φ^{NCol} descreve o mecanismo de não colisão entre agentes

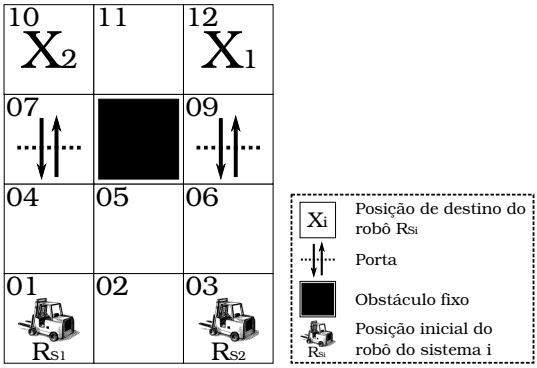


Figura 13: Espaço de trabalho do caso de alcançabilidade, considerando portas não controladas

$$\begin{aligned} \varphi_i^e &: (\varphi_i^{P_1} \wedge \varphi_i^{P_2}) \\ \varphi_t^e &: (\varphi_t^{P_1} \wedge \varphi_t^{P_2}) \\ \varphi_g^e &: (\varphi_g^{P_1} \wedge \varphi_g^{P_2}) \end{aligned} \tag{6.2}$$

$$\begin{aligned} \varphi_i^s &: \left(\varphi_i^{Rp_{S1}} \wedge \varphi_i^{Rp_{S2}} \wedge \varphi_i^{NCol} \wedge \varphi_i^{pAS}\right) \\ \varphi_t^s &: \left(\varphi_t^{Rp_{S1}} \wedge \varphi_t^{Rp_{S2}} \wedge \varphi_t^{NCol} \wedge \varphi_t^{pAS}\right) \\ \varphi_g^s &: \Box\Diamond \left[(NCol = true) \wedge \varphi_g^{pAS}\right] \end{aligned} \tag{6.3}$$

Nas equações acima, o modelo da porta P_1 é dada pela equação 6.4. Uma vez que a descrição de P_2 é similar à mostrada nesta equação, ela não será apresentada. Vale destacar ainda que a variável R_{Si} indica a posição ocupada pelo i-ésimo robô do sistema e s_i corresponde ao sensor deste robô.

$$\begin{aligned} \varphi_i^{P_1} &: (P_1 = ND) \\ \varphi_t^{P_1} &: \begin{cases} \{\neg[(s_1 = 7) \vee (R_{S1} = 7)] \wedge [(s_2 = 7) \vee (R_{S2} = 7)]\} \\ \quad \rightarrow \bigcirc(P_1 = ND) \\ \bigwedge [(s_1 = 7) \vee (s_2 = 7)] \rightarrow \bigcirc[(P_1 = A) \vee (P_1 = F)] \\ \bigwedge [(R_{S1} = 7) \vee (R_{S2} = 7)] \rightarrow \bigcirc(P_1 = A) \end{cases} \\ \varphi_g^{P_1} &: \Box\Diamond (P_1 \neq F) \end{aligned} \tag{6.4}$$

Por sua vez, um robô do sistema é dado pela conjunção das fórmulas que descrevem seu modelo para o caso com observabilidade global (equação 6.5) e o modelo de um sensor (equação 6.6), como foi descrito na seção 5.3.1. Vale ressaltar que a variável R_{Si} indica a posição atual do i-ésimo robô do sistema, tanto na abordagem com observabilidade global quanto na parcial.

$$\varphi_i^{R_{Si}} : (R_{Si} = P_{inicial}^{Rsi})$$
$$\varphi_t^{R_{Si}} : \begin{cases} (R_{Si} = 1) \rightarrow \bigcirc [(R_{Si} = 1) \vee (R_{Si} = 2) \vee (R_{Si} = 4)] \\ \bigwedge (R_{Si} = 2) \rightarrow \bigcirc [(R_{Si} = 2) \vee (R_{Si} = 1) \vee (R_{Si} = 3) \vee (R_{Si} = 5)] \\ \cdots \\ \bigwedge (R_{Si} = 12) \rightarrow \bigcirc [(R_{Si} = 12) \vee (R_{Si} = 11) \vee (R_{Si} = 9)] \\ \bigwedge \bigcirc (P_1 \neq A) \rightarrow \bigcirc (R_{Si} \neq 7) \\ \bigwedge \bigcirc (P_2 \neq A) \rightarrow \bigcirc (R_{Si} \neq 9) \end{cases} \tag{6.5}$$

$$\varphi_i^{s_i} : (s_i = 0)$$
$$\varphi_t^{s_i} : \begin{cases} \bigcirc [(R_{Si} \neq 4) \wedge (R_{Si} \neq 10)] \rightarrow \bigcirc (s_i \neq 7) \\ \bigwedge \bigcirc [(R_{Si} \neq 6) \wedge (R_{Si} \neq 12)] \rightarrow \bigcirc (s_i \neq 9) \\ \bigwedge (s_i \neq 7) \rightarrow \bigcirc (R_{Si} \neq 7) \\ \bigwedge (s_i \neq 9) \rightarrow \bigcirc (R_{Si} \neq 9) \end{cases} \tag{6.6}$$

Por último, nas equações 6.7 e 6.8 são definidas a condição inicial, dinâmica e condição de justiça associadas aos objetivos (de alcançabilidade simples) de R_{S1} alcançar a célula 12 e R_{S2} a célula 10.

$$\varphi_i^{alcS_1} : (alcS_1 = false)$$
$$\varphi_t^{alcS_1} : \begin{cases} (\neg \bigcirc (R_{S1} = 12) \rightarrow [(\bigcirc alcS_1) = alcS_1] \\ \bigwedge \bigcirc (R_{S1} = 12) \rightarrow \bigcirc (alcS_1 = true) \end{cases} \tag{6.7}$$
$$\varphi_g^{alcS_1} : (alcS_1 = true)$$

$$\varphi_i^{alcS_2} : (alcS_2 = false)$$
$$\varphi_t^{alcS_2} : \begin{cases} (\neg \bigcirc (R_{S2} = 10) \rightarrow [(\bigcirc alcS_2) = alcS_2] \\ \bigwedge \bigcirc (R_{S2} = 10) \rightarrow \bigcirc (alcS_2 = true) \end{cases} \tag{6.8}$$
$$\varphi_g^{alcS_2} : (alcS_2 = true)$$

É importante destacar que os modelos dos objetivos não mudam em relação ao tipo de observabilidade do sistema. Assim, a especificação do problema de alcançabilidade simples em SMRs, considerando observabilidade global e parcial, é descrita pela equação 6.9. Além disso, é válido destacar que o operador $\square\Diamond$ só é definido na condição de justiça do sistema, como foi discutido na seção 5.6.

$$\begin{aligned} \varphi_i^{pAS} &: \left(\varphi_i^{alcS_1} \wedge \varphi_i^{alcS_2} \right) \\ \varphi_t^{pAS} &: \left(\varphi_t^{alcS_1} \wedge \varphi_t^{alcS_2} \right) \\ \varphi_g^{pAS} &: ((alcS_1 = true) \wedge (alcS_2 = true)) \end{aligned} \tag{6.9}$$

Como a descrição do mecanismo de não colisão entre agentes (para dois robôs) já foi apresentada na equação 5.44 da seção 5.6, ela não será mostrada novamente neste capítulo. Além disso, vale lembrar que o modelo completo da aplicação, incluindo as especificações, é descrito nas equações 6.3 (sistema) e 6.2 (ambiente).

Controlador considerando observabilidade global

Para o caso considerando observabilidade global, utilizado como exemplo para ilustrar a metodologia de composição de modelos, o controlador obtido é representado pelo sistema de transições mostrado na figura 14

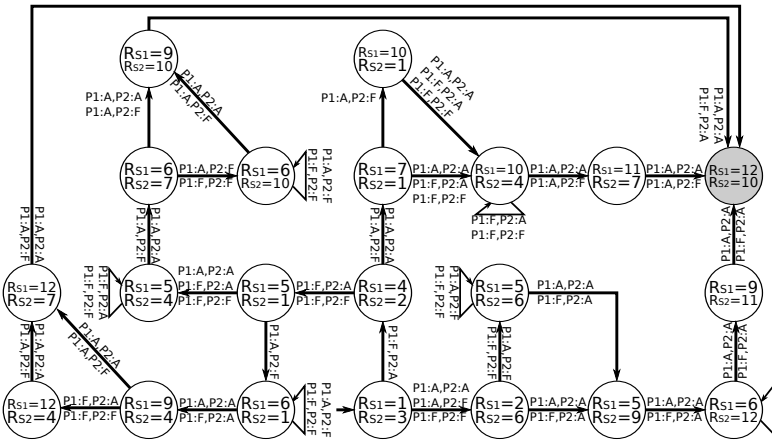


Figura 14: Representação do controlador obtido para o PAS com observabilidade global

Vale ressaltar que o controlador mostrado na figura 14 não corresponde a uma versão exata do controlador (obtido em autômato textual) gerado pelos algoritmos propostos em [29, 56]. O controlador mostrado nesta figura foi descrito de forma que, em cada estado são representadas apenas as posições que cada robô do sistema está ocupando.

A existência de mais de uma possibilidade de evolução no autômato ocorre apenas porque é necessário representar cada uma das evoluções possíveis para as variáveis do ambiente. Deste modo, as transições mostradas na figura 14 estão relacionadas com a mudança de valor das variáveis do ambiente.

Assim, o sistema de transição mostrado na figura 14 é descrito de forma que, um estado identificado pelo rótulo $\begin{smallmatrix} R_{S1}=2 \\ R_{S2}=6 \end{smallmatrix}$ indica que o robô 1 está na célula 2 e o robô 2 na 6. Já uma transição com um rótulo $\begin{smallmatrix} P_1:A, P_2:A \\ P_1:F, P_2:A \end{smallmatrix}$ indica, de forma similar a um *guard*, que a transição é executada quando os valores de P_1 e P_2 (portas 1 e 2) são $P_1 = A$ e $P_2 = A$ (as duas abertas) ou $P_1 = F$ e $P_2 = A$ (porta 1 fechada e porta 2 aberta).

Por exemplo, se no estado em que $R_{S1} = 2$ e $R_{S2} = 6$ a porta 2 está fechada, o controlador evolui para o estado que indica que R_{S2} deve permanecer em 6 e R_{S1} deve ir para 5.

Além disso, esta representação gráfica do controlador foi criada a partir do *software* de desenho vetorial *Inkscape*[1] e tem como objetivo apresentar, de forma concisa, a estratégia obtida. Contudo, o controlador real a ser implementado é obtido a partir do autômato textual gerado pelos algoritmos propostos em [29, 56].

No controlador mostrado na figura 14, é possível notar a existência de três possibilidades de movimentação para os dois robôs do sistema. A seguir são apresentados cada um destes comportamentos, indicando as rotas tomadas por cada um dos robôs para realização das tarefas de alcançabilidade simples associadas a eles.

- Comportamento 1 $\begin{cases} R_{S1} : 1 \mapsto 2 \mapsto 5 \mapsto 6 \mapsto 9 \mapsto 12 \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \mapsto 11 \mapsto 10 \end{cases}$
- Comportamento 2 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 7 \mapsto 10 \mapsto 11 \mapsto 12 \\ R_{S2} : 3 \mapsto 2 \mapsto 1 \mapsto 4 \mapsto 7 \mapsto 10 \end{cases}$
- Comportamento 3 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 5 \mapsto 6 \mapsto 9 \mapsto 12 \\ R_{S2} : 3 \mapsto 2 \mapsto 1 \mapsto 4 \mapsto 7 \mapsto 10 \end{cases}$

Controlador considerando observabilidade parcial

Para o caso considerando observabilidade parcial, cujo modelo é apresentado nesta seção, o controlador obtido é representado pelo sistema de transições mostrado na figura 15

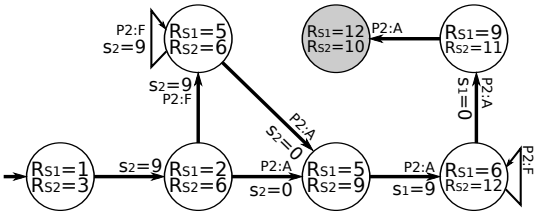


Figura 15: Representação do controlador obtido para o PAS com observabilidade parcial

Do mesmo modo que foi feito para o controlador do caso global, no parcial, também é utilizada uma representação gráfica do controlador, a fim de facilitar o entendimento por parte do leitor. Contudo, existem alguns aspectos que são particulares do caso parcial, são eles o sensor e o *status ND* das portas.

[1] http://inkscape.org/

Como uma porta só é "sentida" quando existe um sensor ligado e "apontado" para ela, optou-se por omitir o *status* de uma porta na representação do controlador quando ela não está sendo detectada. Com isso, é possível deixar a representação mais simples e legível. De forma similar, só é indicado o valor de um sensor quando ele está ligado ($s_i = \{7,9\}$) ou na transição em que ele está desligado ($s_1 = 0$).

Analisando o controlador mostrado na figura 15, é possível perceber que o comportamento definido nele é muito mais simples que o do caso com observabilidade global, possuindo apenas um fluxo de execução, indicado abaixo.

$$\begin{cases} R_{S1} : 1 \mapsto 2 \mapsto 5 \mapsto 6 \mapsto 9 \mapsto 12 \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \mapsto 11 \mapsto 10 \end{cases}$$

Devido à falta de informação que é inerente à observabilidade parcial, o número de comportamentos possíveis, associados às possibilidades de evolução das variáveis do ambiente, diminui bastante. Isso ocorre porque, se o sistema não pode "sentir" um determinado elemento do ambiente, ele não pode perceber mudanças no *status* deste elemento e, assim, não são gerados outros comportamentos.

Outros aspectos associados à complexidade e eficiência dos controladores estão ligados aos algoritmos utilizados para obtê-los. No final deste capítulo será apresentada uma discussão sobre estes aspectos.

6.2 PROBLEMA DE ALCANÇABILIDADE COOPERATIVA

Quando se utiliza especificações de alcançabilidade simples para descrever os objetivos do sistema, é feita uma associação entre objetivo e robô que deve realizá-lo, definindo, mesmo que involuntariamente, uma diferenciação entre agentes.

Contudo, se não é relevante para o sistema definir qual agente deve realizar cada objetivo, podem ser utilizadas especificações de alcançabilidade suficiente para descrever o objetivo global do sistema. Como foi discutido no capítulo 5, na especificação de alcançabilidade suficiente é definido um conjunto de agentes que podem ser utilizados para satisfazer o objetivo. Desta forma, a especificação é considerada como realizada se qualquer um destes robôs alcançar a posição de destino.

Assim, dado um conjunto de células de destino $D = \{D_1, \cdots, D_{N_{pAC}}\}$ e um conjunto de robôs do sistema $R' = \{R_{S1}, \cdots, R_{S_{N'}}\}$ que podem ser utilizados para alcançar estas células, tem-se um Problema de Alcançabilidade Cooperativa (PAC), cuja especificação é definida como segue.

6.2.1 Modelagem das especificações

Quando se tem um PAC, a especificação do sistema é dada como uma conjunção de objetivos básicos de alcançabilidade suficiente, associados a cada uma das células de destino. Assim, dado que o objetivo básico de alcançabilidade suficiente associado a uma célula $D_j \in D$ é descrito como mostrado na equação 6.10, a especificação do PAC é dada pela equação 6.11.

$$
\begin{aligned}
&\varphi_i^{alcC_j} : (alcC_j = false) \\
&\varphi_t^{alcC_j} : \begin{cases} \neg \bigcirc \left(\bigvee_{i=1}^{N'} (R_{Si} = D_j) \right) \rightarrow \left[(\bigcirc alcC_j) = alcC_j \right] \\ \wedge \bigcirc \left(\bigvee_{i=1}^{N'} (R_{Si} = D_j) \right) \rightarrow \bigcirc (alcC_j = true) \end{cases} \\
&\varphi_g^{alcC_j} : (alcC_j = true)
\end{aligned}
\tag{6.10}
$$

$$\begin{aligned}
\varphi_i^{pAC} &: \bigwedge_{j=1}^{N_{pAC}} \varphi_i^{alcC_j} \\
\varphi_t^{pAC} &: \bigwedge_{j=1}^{N_{pAC}} \varphi_t^{alcC_j} \\
\varphi_g^{pAC} &: \bigwedge_{j=1}^{N_{pAC}} (alcC_j = true)
\end{aligned} \tag{6.11}$$

A seguir são apresentados dois exemplos, ilustrando a modelagem de uma aplicação carcterizada pela presença de um problema de alcançabilidade cooperativa.

6.2.2 Exemplo 1: Robô sem Falha

Para ilustrar o processo de modelagem de um problema de alcançabilidade cooperativa, é utilizado o exemplo mostrado na figura 16. Neste, um sistema composto por dois robôs R_{S1} e R_{S2} tem como objetivo que as posições 10 e 12 sejam alcançadas. Além disso, ainda existe um robô adversário circulando pelo espaço de trabalho, pelas células 7, 8 e 9.

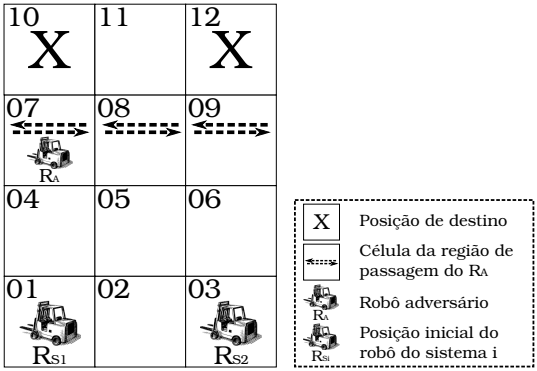


Figura 16: Espaço de trabalho considerando um robô adversário

Modelos

Como, no exemplo mostrado na figura 16, o ambiente é composto apenas por um robô adversário, o modelo do adversário já corresponde ao do ambiente. Assim, para o caso em que é considerada observabilidade glo-

bal, o modelo do ambiente é descrito na equação 6.12. Neste exemplo foi considerado que, inicialmente o adversário está na posição 7.

$$\begin{array}{l}
\varphi_i^{R_A} : (R_A = 7) \\
\varphi_t^{R_A} : \left\{ \begin{array}{l}
(R_A = 7) \rightarrow \bigcirc[(R_A = 7) \vee (R_A = 8)] \\
\bigwedge (R_A = 8) \rightarrow \bigcirc[(R_A = 8) \vee (R_A = 7) \vee (R_A = 9)] \\
\bigwedge (R_A = 9) \rightarrow \bigcirc[(R_A = 9) \vee (R_A = 8)] \\
\bigwedge [(R_{S1} = 7) \vee (R_{S2} = 7)] \rightarrow \bigcirc(R_A \neq 7) \\
\bigwedge [(R_{S1} = 8) \vee (R_{S2} = 8)] \rightarrow \bigcirc(R_A \neq 8) \\
\bigwedge [(R_{S1} = 9) \vee (R_{S2} = 9)] \rightarrow \bigcirc(R_A \neq 9)
\end{array} \right. \\
\varphi_g^{R_A} : \Box\Diamond \, true
\end{array} \tag{6.12}$$

Para o caso com observabilidade parcial, o modelo do robô adversário é descrito na equação 6.13.

$$\begin{array}{l}
\varphi_i^{R_A} : (R_A = 0) \\
\varphi_t^{R_A} : \left\{ \begin{array}{l}
(R_A = 7) \rightarrow \bigcirc[(R_A = 7) \vee (R_A = 0) \vee (R_A = 8)] \\
\bigwedge (R_A = 8) \rightarrow \bigcirc[(R_A = 8) \vee (R_A = 7) \vee (R_A = 9) \vee (R_A = 0)] \\
\bigwedge (R_A = 9) \rightarrow \bigcirc[(R_A = 9) \vee (R_A = 8) \vee (R_A = 0)] \\
\bigwedge (R_A = 0) \rightarrow \bigcirc[(R_A = 0) \vee (R_A = 7) \vee (R_A = 8) \vee (R_A = 9)] \\
\bigwedge [(R_{S1} = 7) \vee (R_{S2} = 7)] \rightarrow \bigcirc(R_A \neq 7) \\
\bigwedge [(R_{S1} = 8) \vee (R_{S2} = 8)] \rightarrow \bigcirc(R_A \neq 8) \\
\bigwedge [(R_{S1} = 9) \vee (R_{S2} = 9)] \rightarrow \bigcirc(R_A \neq 9) \\
\bigwedge [(s_1 \neq 7) \wedge (s_2 \neq 7)] \rightarrow \bigcirc(R_A \neq 7) \\
\bigwedge [(s_1 \neq 8) \wedge (s_2 \neq 8)] \rightarrow \bigcirc(R_A \neq 8) \\
\bigwedge [(s_1 \neq 9) \wedge (s_2 \neq 9)] \rightarrow \bigcirc(R_A \neq 9)
\end{array} \right. \\
\varphi_g^{R_A} : \Box\Diamond true
\end{array} \tag{6.13}$$

Por sua vez, os modelos do sistema, considerando observabilidade global e parcial, são apresentados nas equações 6.14 e 6.15, respectivamente.

$$\begin{array}{c}
\varphi_i^s : (\varphi_i^{R_{S1}} \wedge \varphi_i^{R_{S2}} \wedge \varphi_i^{NCol} \wedge \varphi_i^{pAC}) \\
\varphi_t^s : (\varphi_t^{R_{S1}} \wedge \varphi_t^{R_{S2}} \wedge \varphi_t^{NCol} \wedge \varphi_t^{pAC}) \\
\varphi_g^s : \Box\Diamond \left[(NCol = true) \wedge \varphi_g^{pAC} \right]
\end{array} \tag{6.14}$$

$$\begin{array}{l}
\varphi_i^s : \left[(\varphi_i^{R_{S1}} \wedge \varphi_i^{s_1}) \wedge (\varphi_i^{R_{S2}} \wedge \varphi_i^{s_2}) \wedge \varphi_i^{NCol} \wedge \varphi_i^{pAC} \right] \\
\varphi_t^s : \left[(\varphi_t^{R_{S1}} \wedge \varphi_t^{s_1}) \wedge (\varphi_t^{R_{S2}} \wedge \varphi_t^{s_2}) \wedge \varphi_t^{NCol} \wedge \varphi_t^{pAC} \right] \\
\varphi_g^s : \Box\Diamond \left[(NCol = true) \wedge \varphi_g^{pAC} \right]
\end{array} \tag{6.15}$$

Nas equações acima, a fórmula $\varphi_i^{R_{Si}}$ corresponde ao modelo do i-ésimo robô do sistema, considerando observabilidade global. Por sua vez, para descrever o sistema no caso com observabilidade parcial, o modelo do robô do sistema é descrito pela conjunção $\left(\varphi_i^{R_{Si}} \wedge \varphi_i^{s_i} \right)$, onde $\varphi_i^{s_i}$ representa a descrição do sensor do robô R_{Si}.

Nas equações 6.16 e 6.17 são apresentadas as descrições de um robô do sistema e de seu sensor.

$$\varphi_i^{R_{Si}} : (R_{Si} = P_{inicial}^{Rsi})$$
$$\varphi_t^{R_{Si}} : \begin{cases} (R_{Si} = 1) \rightarrow \bigcirc[(R_{Si} = 1) \vee (R_{Si} = 2) \vee (R_{Si} = 4)] \\ \bigwedge (R_{Si} = 2) \rightarrow \bigcirc[(R_{Si} = 2) \vee (R_{Si} = 1) \vee (R_{Si} = 3) \vee (R_{Si} = 5)] \\ \cdots \\ \bigwedge (R_{Si} = 12) \rightarrow \bigcirc[(R_{Si} = 12) \vee (R_{Si} = 11) \vee (R_{Si} = 9)] \\ \bigwedge \bigcirc(R_A = 7) \rightarrow \bigcirc(R_{Si} \neq 7) \\ \bigwedge \bigcirc(R_A = 8) \rightarrow \bigcirc(R_{Si} \neq 8) \\ \bigwedge \bigcirc(R_A = 9) \rightarrow \bigcirc(R_{Si} \neq 9) \end{cases} \tag{6.16}$$

$$\varphi_i^{s_i} : (s_i = 0)$$
$$\varphi_t^{s_i} : \begin{cases} \bigcirc[(R_{Si} \neq 4) \wedge (R_{Si} \neq 8) \wedge (R_{Si} \neq 10)] \rightarrow \bigcirc(s_i \neq 7) \\ \bigwedge \bigcirc[(R_{Si} \neq 5) \wedge (R_{Si} \neq 7) \wedge (R_{Si} \neq 9) \wedge (R_{Si} \neq 11)] \rightarrow \bigcirc(s_i \neq 8) \\ \bigwedge \bigcirc[(R_{Si} \neq 6) \wedge (R_{Si} \neq 8) \wedge (R_{Si} \neq 12)] \rightarrow \bigcirc(s_i \neq 9) \\ \bigwedge (s_i \neq 7) \rightarrow \bigcirc(R_{Si} \neq 7) \\ \bigwedge (s_i \neq 8) \rightarrow \bigcirc(R_{Si} \neq 8) \\ \bigwedge (s_i \neq 9) \rightarrow \bigcirc(R_{Si} \neq 9) \end{cases} \tag{6.17}$$

Por último, nas equações 6.18 e 6.19 são descritos os objetivos de alcançabilidade cooperativa referente às posições 10 e 12. Na equação 6.20 é descrita a especificação do PAC. Vale ressaltar que a modelagem da especificação independe do tipo de observabilidade considerado.

$$\begin{aligned} &\varphi_i^{alcC_1} : (alcC_1 = false) \\ &\varphi_t^{alcC_1} : \begin{cases} \neg \bigcirc[(R_{S1} = 10) \vee (R_{S2} = 10)] \rightarrow [(\bigcirc alcC_1) = alcC_1] \\ \bigwedge \bigcirc[(R_{S1} = 10) \vee (R_{S2} = 10)] \rightarrow \bigcirc(alcC_1 = true) \end{cases} \\ &\varphi_g^{alcC_1} : (alcC_1 = true) \end{aligned} \tag{6.18}$$

$$\begin{aligned} &\varphi_i^{alcC_2} : (alcC_2 = false) \\ &\varphi_t^{alcC_2} : \begin{cases} \neg \bigcirc[(R_{S1} = 12) \vee (R_{S2} = 12)] \rightarrow [(\bigcirc alcC_2) = alcC_1] \\ \bigwedge [(R_{S1} = 12) \vee (R_{S2} = 12)] \rightarrow \bigcirc(alcC_2 = true) \end{cases} \\ &\varphi_g^{alcC_2} : (alcC_2 = true) \end{aligned} \tag{6.19}$$

$$\begin{aligned} \varphi_i^{pAC} &: \left(\varphi_i^{alcC_1} \wedge \varphi_i^{alcC_2}\right) \\ \varphi_t^{pAC} &: \left(\varphi_t^{alcC_1} \wedge \varphi_t^{alcC_2}\right) \\ \varphi_g^{pAC} &: \left(\varphi_g^{alcC_1} \wedge \varphi_g^{alcC_2}\right) \end{aligned} \tag{6.20}$$

Vale destacar que, para o caso com observabilidade global, a estrutura de jogo é obtida através das equações 6.12 (ambiente) e 6.14 (sistema). Por sua vez, para observabilidade parcial, o sistema é descrito na equação 6.15 e o ambiente na 6.13.

Controlador considerando observabilidade global

Considerando observabilidade global, o controlador obtido é representado pelo sistema de transições mostrado na figura 17

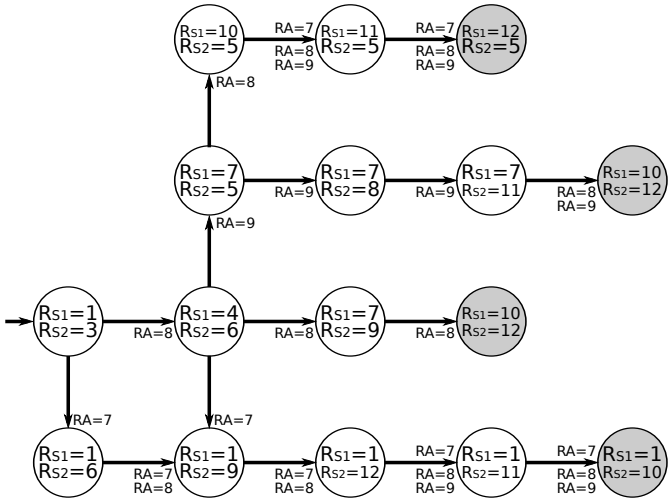


Figura 17: Representação do controlador obtido para o PAC, observabilidade global

De forma similar ao que foi feito para o problema de alcançabilidade simples, também foi utilizada uma forma gráfica (vide figura 17) que facilitasse o entendimento, por parte do leitor, sobre o comportamento definido no controlador gerado pelos algoritmos propostos em [29, 56]. Além disso, o sistema de transições utilizado para representar o controlador é o mesmo que o usado no PAS.

Neste sistema de transição, os estados indicam a posição que está sendo ocupada por cada robô do sistema. Por sua vez, o rótulo de uma transição indica a condição para sua execução (similar a um guarda).

No controlador mostrado na figura 17, é possível notar a existência de cinco comportamentos que levam a realização da especificação do sistema, são eles:

- Comportamento 1 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 7 \mapsto 10 \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \end{cases}$

- Comportamento 2 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 1 \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \mapsto 11 \mapsto 10 \end{cases}$

- Comportamento 3 $\begin{cases} R_{S1} : 1 \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \mapsto 11 \mapsto 10 \end{cases}$
- Comportamento 4 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 7 \mapsto 10 \\ R_{S2} : 3 \mapsto 6 \mapsto 5 \mapsto 8 \mapsto 11 \mapsto 12 \end{cases}$
- Comportamento 5 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 7 \mapsto 10 \mapsto 11 \mapsto 12 \\ R_{S2} : 3 \mapsto 6 \mapsto 5 \end{cases}$

Controlador considerando observabilidade parcial

Considerando observabilidade parcial, o controlador obtido é representado pelo sistema de transições mostrado na figura 18

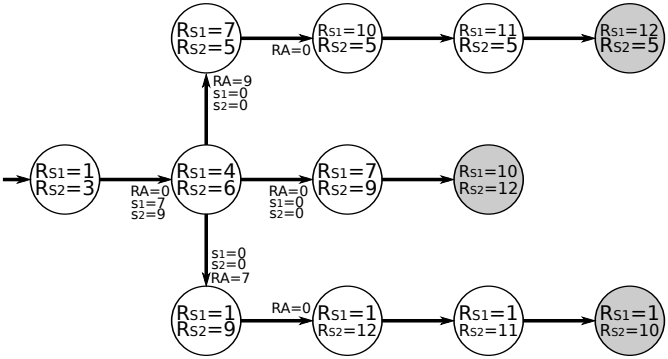


Figura 18: Representação do controlador obtido para o PAC, observabilidade parcial

Para o controlador do caso com observabilidade parcial, além dos *guards* associados às variáveis de estado do ambiente, também são representados nas transições a ativação/desativação dos sensores dos robôs, de forma similar ao que foi feito no PAS.

No controlador mostrado na figura 18, é possível notar a existência de 3 comportamentos que levam a realização da especificação do sistema, são eles:

- Comportamento 1 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 7 \mapsto 10 \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \end{cases}$
- Comportamento 2 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 1 \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \mapsto 11 \mapsto 10 \end{cases}$
- Comportamento 3 $\begin{cases} R_{S1} : 1 \mapsto 4 \mapsto 7 \mapsto 10 \mapsto 11 \mapsto 12 \\ R_{S2} : 3 \mapsto 6 \mapsto 5 \end{cases}$

Vale observar que os comportamentos presentes no controlador do caso com observabilidade parcial também estão no caso global. Sendo que os comportamentos 1, 2 e 3 do caso parcial correspondem aos 1, 2 e 5 do caso global, respectivamente. Vale destacar também que a diminuição da quantidade de comportamentos está associada à diminuição da quantidade de informação que o sistema, considerando observabilidade parcial, tem sobre o adversário.

6.2.3 Exemplo 2: Robôs com possibilidade de falha

Quando são consideradas aplicações em que os objetivos não estão associados a robôs específicos, é possível definir também o aspecto de tolerância à falta, de modo que, se um robô entrar em falha, o objetivo que havia sido designado para ele é realocado para outro robô.

Assim, aplicações com especificações de alcançabilidade cooperativa podem conter também mecanismos de tolerância à falta. Neste sentido, com o intuito de simplificar a vizualização dos resultados, é utilizado um exemplo bastante simples, com o espaço de trabalho e posições de destino mostrados na figura 19. Contudo, o tipo de robô do sistema considera a possibilidade de falha.

É importante ressaltar que a utilização de um exemplo onde não havia outro elemento do ambiente, além do *simulador de falha*, tem como objetivo focar na capacidade do método proposto em Piterman *et al.*(2006)[56] e dos princípios de modelagem, propostos neste trabalho, em tratar problemas que envolvam tolerância à falta. Optou-se então por utilizar um exemplo mais simples, evitando a geração de um controlador muito complexo, cuja representação seria de difícil análise para o leitor.

Além disso, para o exemplo apresentado na figura 16, não é possível gerar um controlador que garanta a realização da especificação do sistema, quando são considerados robôs com possibilidade de falha. Isso ocorre neste exemplo porque, se um dos dois robôs falhar, não se tem garantia de que o outro, sozinho, poderia passar pelo adversário caso o agente do ambiente adotasse um comportamento de perseguição.

Vale destacar que esta não é uma limitação do método proposto em Piterman *et al.* (2006) [56] ou dos princípios de modelagem propostos neste trabalho, sendo inerente a existência de adversários no espaço de trabalho.

Consequentemente, para este exemplo, o ambiente é composto apenas pelo *simulador de falha* e considerando apenas observabilidade global. Na equação 6.21 é descrito o modelo do simulador de falha, que corresponde ao modelo ambiente.

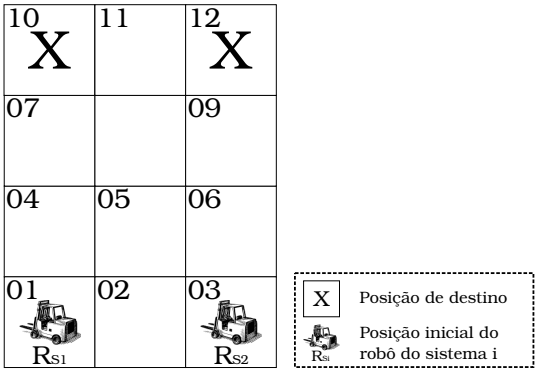


Figura 19: Espaço de trabalho referente ao experimento considerando tolerância à falta

$$
\begin{aligned}
&\varphi_i^F : (falha = 0) \\
&\varphi_t^F : \begin{cases} \neg \bigcirc (falha \neq 0) \rightarrow [(\bigcirc nF) = nF] \\ \bigwedge \bigcirc (falha \neq 0) \rightarrow [(\bigcirc nF) = nF + 1] \\ \bigwedge [(falha = 0) \wedge (nF < 1)] \rightarrow \\ \qquad \bigcirc [(falha = 0) \vee (falha = 1) \vee (falha = 2)] \\ \bigwedge \neg [(falha = 0) \wedge (nF < 1)] \rightarrow \bigcirc (falha = 0) \\ \bigwedge (st_1 \neq A) \rightarrow \bigcirc (falha \neq 1) \\ \bigwedge (st_2 \neq A) \rightarrow \bigcirc (falha \neq 2) \end{cases} \\
&\varphi_g^F : \Box \Diamond true
\end{aligned}
\tag{6.21}
$$

Por sua vez, o modelo do sistema é dado pela conjunção dos modelos de dois robôs com possibilidade de falha ($\varphi^{RF_{S1}}$ e $\varphi^{RF_{S2}}$), da especificação de não colisão entre eles e da especificação do problema de alcançabilidade cooperativa. Assim, na equação 6.22 é descrito o modelo do sistema para o caso com possibilidade de falha. Vale destacar que especificação do problema de alcançabilidade cooperativa (φ^{pAC}) é a mesma do exemplo anterior (robô sem falha).

$$
\begin{aligned}
&\varphi_i^s : (\varphi_i^{RF_{S1}} \wedge \varphi_i^{RF_{S2}} \wedge \varphi_i^{NCol} \wedge \varphi_i^{pAC}) \\
&\varphi_t^s : (\varphi_t^{RF_{S1}} \wedge \varphi_t^{RF_{S2}} \wedge \varphi_t^{NCol} \wedge \varphi_t^{pAC}) \\
&\varphi_g^s : \Box \Diamond \left[(objNCol = true) \wedge \varphi_g^{pAC} \right]
\end{aligned}
\tag{6.22}
$$

Por sua vez, o modelo de um robô com possibilidade de falha é dado pela equação 6.23.

$$
\varphi^{RF_{Si}} = \varphi^{R_{Si}} \wedge \varphi^{pf_i} \tag{6.23}
$$

onde $\varphi^{R_{Si}}$ corresponde à descrição de um robô sem falha e o modelo da possibilidade de falha (φ^{pf_i}) é descrito na equação 6.25.

Nas equações 6.24 e 6.25 são descritos os modelos do robô R_{Si} e da possibilidade de falha associada a ele.

$$\begin{array}{l} \varphi_i^{R_{Si}} : (R_{Si} = P_{inicial}^{Rsi}) \\ \varphi_t^{R_{Si}} : \left\{ \begin{array}{l} (R_{Si} = 1) \rightarrow \bigcirc [(R_{Si} = 1) \vee (R_{Si} = 2) \vee (R_{Si} = 4)] \\ \bigwedge (R_{Si} = 2) \rightarrow \bigcirc [(R_{Si} = 2) \vee (R_{Si} = 1) \vee (R_{Si} = 3) \vee (R_{Si} = 5)] \\ \cdots \\ \bigwedge (R_{Si} = 12) \rightarrow \bigcirc [(R_{Si} = 12) \vee (R_{Si} = 11) \vee (R_{Si} = 9)] \end{array} \right. \end{array} \tag{6.24}$$

$$\begin{array}{l} \varphi_i^{pf_i} : (st_i = A) \\ \varphi_t^{pf_i} : \left\{ \begin{array}{l} (st_i \neq A) \rightarrow [(\bigcirc R_{Si}) = R_{Si}] \\ \bigwedge [(st_i = A) \wedge \bigcirc (falha \neq i)] \rightarrow \bigcirc (st_i = F) \\ \bigwedge \neg [(st_i = A) \wedge \bigcirc (falha \neq i)] \rightarrow [(\bigcirc st_i) = st_i] \end{array} \right. \\ \varphi_g^{pf_i} : \Box \Diamond true \end{array} \tag{6.25}$$

Controladores

Uma vez que neste exemplo só é abordado o caso com observabilidade global, para o problema de alcançabilidade cooperativa e considerando robôs com possibilidade de falha, o controlador obtido é representado pelo sistema de transições mostrado na figura 20.

No controlador representado na figura 20, é possível notar a existência de 3 comportamentos que levam a realização da especificação do sistema, são eles:

- Comportamento 1 $\left\{ \begin{array}{l} R_{S1} : 1 \mapsto 4 \mapsto 7 \mapsto 10 \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \end{array} \right.$
- Comportamento 2 $\left\{ \begin{array}{l} R_{S1} : 1 \mapsto 4 \mapsto 7 \mapsto 10 \mapsto 11 \mapsto 12 \\ R_{S2} : (falha : 2) \end{array} \right.$
- Comportamento 3 $\left\{ \begin{array}{l} R_{S1} : (falha : 1) \\ R_{S2} : 3 \mapsto 6 \mapsto 9 \mapsto 12 \mapsto 11 \mapsto 10 \end{array} \right.$

Vale destacar que nos comportamentos 2 e 3, a variável $falha$ está indicando que houve falha nos robôs R_{S2} e R_{S1}, respectivamente. Deste modo, é indicado que se o robô R_{S2} falhar, o robô R_{S1} realizará os objetivos de alcançar a célula 10 e a 12 (comportamento 2). Por outro lado, se R_{S1} falhar, ambos os objetivos serão realizados por R_{S2} (comportamento 3).

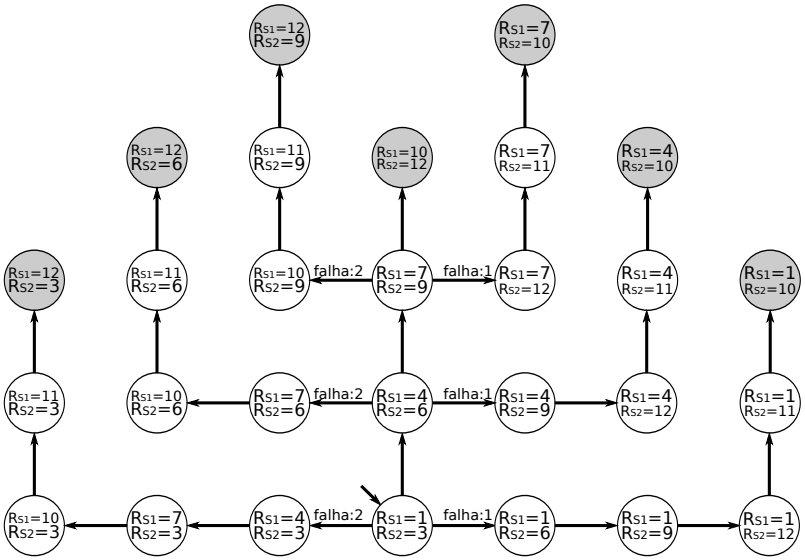


Figura 20: Representação do controlador obtido para o PAC, considerando falta

6.3 PROBLEMA DE BUSCA E RESGATE

Outra classe de problemas bastante relevante quando se trata de sistemas multi-robôs é a que engloba as aplicações de busca e coleta (*foraging*). O interesse nesta classe de problemas está associado a sua capacidade de representar diferentes tipos de tarefas, tais como busca e resgate, limpeza de resíduos tóxicos e busca e retirada de minas[25].

Assim, esta classe de aplicações também foi utilizada para validar o uso da teoria de estrutura de jogos e dos princípios de modelagem propostos na solução do problema de coordenação de SMRs. Nas seções a seguir, é apresentada a forma como uma especificação de busca e resgate é descrita, bem como um exemplo em que este tipo de problema é modelado e resolvido utilizando os princípios de modelagem propostos neste trabalho.

6.3.1 Modelagem das especificações

De forma geral, uma especificação de busca e resgate de um objeto por um robô pode ser vista como uma sequência de dois objetivos de alcan-

çabilidade simples. Ou seja, o robô deve alcançar a posição onde o objeto está e, depois de coletá-lo, alcançar a célula onde ele deverá ser deixado. Na equação 6.26 é apresentada a descrição deste objetivo.

$$
\begin{aligned}
&\varphi_i^{ob_k^i} : (obC_k^i = false) \wedge (obE_k^i = false) \\
&\varphi_t^{ob_k^i} : \begin{cases} \neg \bigcirc (R_{Si} = P_k^{obj}) \rightarrow \left[(\bigcirc obC_k^i) = obC_k^i \right] \\ \wedge \left[(obC_k^i \neq true) \vee \bigcirc (R_{S1} \neq D_k^{obj}) \right] \rightarrow \left[(\bigcirc obE_k^i) = obE_k^i \right] \\ \wedge \bigcirc (R_{Si} = P_k^{obj}) \rightarrow \bigcirc (obC_k^i = true) \\ \wedge \neg \left[(obC_k^i \neq true) \vee \bigcirc (R_{Si} \neq D_k^{obj}) \right] \rightarrow \bigcirc (obE_k^i = true) \end{cases} \\
&\varphi_g^{ob_k^i} : (obE_k^i = true)
\end{aligned}
\tag{6.26}
$$

Na equação 6.26, obC_k^i e obE_k^i são as variáveis indicadoras que indicam que o objeto k foi coletado e entregue, respectivamente, pelo i-ésimo robô do sistema. Os valores P_k^{obj} e D_k^{obj} correspondem à posição do objeto k e sua célula de destino, respectivamente. Vale destacar ainda que $\varphi^{ob_k^i}$ corresponde ao objetivo de busca e resgate do objeto k pelo i-ésimo robô do sistema.

Considerando que existem N_{col} robôs do sistema que podem coletar e entregar o objeto k, a especificação de busca e resgate associada ao k-ésimo objeto é descrita na equação 6.27.

$$
\begin{aligned}
&\varphi_i^{obj_k} : (Obj_k = false) \\
&\varphi_t^{obj_k} : \begin{cases} \neg \left(\bigvee_{i=1}^{N_{col}} \varphi_g^{ob_k^i} \right) \rightarrow \left[(\bigcirc Obj_k) = Obj_k \right] \\ \wedge \left(\bigvee_{i=1}^{N_{col}} \varphi_g^{ob_k^i} \right) \rightarrow \bigcirc (Obj_k = true) \end{cases} \\
&\varphi_g^{obj_k} : (Obj_k = true)
\end{aligned}
\tag{6.27}
$$

Na equação 6.27, a variável indicadora Obj_k indica que o objeto k foi coletado e entregue por algum dos robôs do sistema. Assim, na fórmula φ^{obj_k} é descrita uma especificação de cobertura de objetivos suficientes (de objetivos de busca e resgate), de modo que, se algum robô coletar o objeto e levá-lo a sua posição de destino, a especificação é satisfeita.

Por último, considerando que existem N_{Ob} objetos no espaço de trabalho e N_{col} robôs do sistema que podem ser utilizados para coletá-los e levá-los a suas respectivas posições de destino, a especificação do problema de busca e resgate (PBR) pode ser descrita pela equação 6.28.

$$\begin{aligned}
\varphi_i^{BeR} &: \bigwedge_{k=1}^{N_{Ob}} \varphi_i^{obj_k} \\
\varphi_t^{BeR} &: \bigwedge_{k=1}^{N_{Ob}} \varphi_t^{obj_k} \\
\varphi_g^{BeR} &: \bigwedge_{k=1}^{N_{Ob}} \varphi_g^{obj_k}
\end{aligned} \tag{6.28}$$

6.3.2 Exemplo

Para ilustrar a modelagem e obtenção do controlador em problemas busca e resgate, é utilizado o exemplo mostrado na figura 21. Neste exemplo, é utilizado um espaço de trabalho com 16 células onde estão inseridos dois robôs do sistema, que podem ser utilizados para buscar, coletar e entregar dois objetos (Ob_1 e Ob_2) em posições pré-definidas (5 e 8, respectivamente).

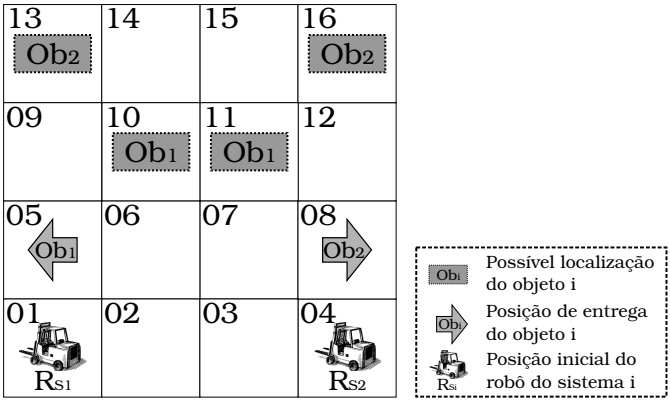


Figura 21: Espaço de trabalho do caso de busca e resgate

Vale destacar que, neste exemplo, os objetos são representados por *objetos com posições desconhecidas* (OPD). Assim, o sistema não conhece, antes do início da execução, a posição exata dos objetos. Na figura 21 é indicado que o objeto 1 pode estar nas células 10 ou 11 e Ob_2 nas células 13 e 16. Além disso, o Ob_1 deve ser entregue na posição 5 e Ob_2 na célula 8.

A seguir são apresentados os modelos utilizados para descrever este exemplo e o controlador obtido.

Modelagem considerando observabilidade global

No exemplo mostrado na figura 21, o ambiente é descrito pela conjunção dos modelos dos objetos 1 e 2. Na equação 6.29 é apresentada a descrição do ambiente, para o caso considerando observabilidade global.

$$\begin{array}{l} \varphi_i^e : (\varphi_i^{OBJ_1} \wedge \varphi_i^{OBJ_2}) \\ \varphi_t^e : (\varphi_t^{OBJ_1} \wedge \varphi_t^{OBJ_2}) \\ \varphi_g^e : (\varphi_g^{OBJ_1} \wedge \varphi_g^{OBJ_2}) \end{array} \tag{6.29}$$

Na equação 6.29, φ^{OBJ_1} e φ^{OBJ_2} correspondem aos modelos dos OPDs, considerando observabilidade global, que representam os objetos 1 e 2, respectivamente. Na equação 6.30 é descrito o modelo de φ^{OBJ_1} (a descrição do objeto 2 é análoga).

$$\begin{array}{l} \varphi_i^{OBJ_1} : (Ob_1 = 0) \\ \varphi_t^{OBJ_1} : \left\{ \begin{array}{l} (Ob_1 = 0) \rightarrow \bigcirc[(Ob_1 = 10) \vee (Ob_1 = 11)] \\ \bigwedge (Ob_1 \neq 0) \rightarrow [(\bigcirc Ob_1) = Ob_1] \end{array} \right. \\ \varphi_i^{OBJ_1} : \Box\Diamond true \end{array} \tag{6.30}$$

Para o caso com observabilidade global o modelo do sistema é descrito pela equação 6.31, onde $\varphi^{R_{S1}}$ e $\varphi^{R_{S2}}$ correspondem aos modelos dos robôs do sistema, φ_i^{NCol} descreve o mecanismo de não colisão e φ_i^{BeR} descreve a especificação de busca e resgate.

$$\begin{array}{l} \varphi_i^s : (\varphi_i^{R_{S1}} \wedge \varphi_i^{R_{S2}} \wedge \varphi_i^{NCol} \wedge \varphi_i^{BeR}) \\ \varphi_t^s : (\varphi_t^{R_{S1}} \wedge \varphi_t^{R_{S2}} \wedge \varphi_t^{NCol} \wedge \varphi_t^{BeR}) \\ \varphi_g^s : \Box\Diamond \left[(NCol = true) \wedge \varphi_g^{BeR}\right] \end{array} \tag{6.31}$$

Na equação 6.32 é apresentada a descrição de um robô do sistema.

$$\begin{array}{l} \varphi_i^{R_{Si}} : (R_{Si} = P_{inicial}^{Rsi}) \\ \varphi_t^{R_{Si}} : \left\{ \begin{array}{l} (R_{Si} = 1) \rightarrow \bigcirc[(R_{Si} = 1) \vee (R_{Si} = 2) \vee (R_{Si} = 5)] \\ \bigwedge (R_{Si} = 2) \rightarrow \bigcirc[(R_{Si} = 2) \vee (R_{Si} = 1) \vee (R_{Si} = 3) \vee (R_{Si} = 6)] \\ \quad \vdots \\ \bigwedge (R_{Si} = 16) \rightarrow \bigcirc[(R_{Si} = 16) \vee (R_{Si} = 12) \vee (R_{Si} = 15)] \end{array} \right. \\ \varphi_t^{R_{Si}} : \Box\Diamond true \end{array} \tag{6.32}$$

Na equação 6.32, é importante notar a ausência de restrições de movimentação no robô. Isso ocorre porque os elementos do ambiente (OPDs) não representam obstáculos, mas objetos a serem coletados.

Modelagem considerando observabilidade parcial

Para o caso parcial, o modelo do ambiente é descrito na equação 6.33.

$$\begin{aligned}
\varphi_i^e &: (\varphi_i^{OBJp_1} \wedge \varphi_i^{OBJp_2}) \\
\varphi_t^e &: (\varphi_t^{OBJp_1} \wedge \varphi_t^{OBJp_2}) \\
\varphi_g^e &: (\varphi_g^{OBJp_1} \wedge \varphi_g^{OBJp_2})
\end{aligned} \tag{6.33}$$

Na equação 6.33, φ^{OBJp_1} e φ^{OBJp_2} correspondem aos modelos dos OPDs, considerando observabilidade parcial, que representam os objetos 1 e 2, respectivamente. Na equação 6.34 é descrito o modelo de φ^{OBJ_2} (a descrição do objeto 1 é análoga).

$$\begin{aligned}
&\varphi_i^{OBJp_2} : (Ob_2 = 0) \wedge (V_{13} = nv) \wedge (V_{16} = nv) \\
&\varphi_t^{OBJp_2} : \begin{cases}
(Ob_2 \neq 0) \rightarrow \{[(\bigcirc Ob_2) = Ob_2)] \wedge \bigcirc [(V_{13} = v) \wedge (V_{16} = v)]\} \\
\bigwedge (Ob_2 = 0) \wedge \neg [(s_1 = 13) \vee (s_2 = 13)] \rightarrow [(\bigcirc V_{13}) = V_{13}] \\
\bigwedge (Ob_2 = 0) \wedge \neg [(s_1 = 16) \vee (s_2 = 16)] \rightarrow [(\bigcirc V_{16}) = V_{16}] \\
\bigwedge (Ob_2 = 0) \wedge [(s_1 = 13) \vee (s_2 = 13)] \rightarrow \bigcirc (V_{13} = v) \\
\bigwedge (Ob_2 = 0) \wedge [(s_1 = 16) \vee (s_2 = 16)] \rightarrow \bigcirc (V_{16} = v) \\
\bigwedge (Ob_2 = 0) \wedge \{\neg [(s_1 = 13) \vee (s_2 = 13)] \vee (V_{13} = v)\} \rightarrow \bigcirc (Ob_2 \neq 13) \\
\bigwedge (Ob_2 = 0) \wedge \{\neg [(s_1 = 16) \vee (s_2 = 16)] \vee (V_{16} = v)\} \rightarrow \bigcirc (Ob_2 \neq 16) \\
\bigwedge \bigcirc [(V_{13} = v) \wedge (V_{16} = v)] \rightarrow \bigcirc (Ob_2 \neq 0)
\end{cases} \\
&\varphi_g^{OBJp_2} : \Box \Diamond true
\end{aligned} \tag{6.34}$$

Quando é considerada observabilidade parcial, é necessário incluir o modelo do sensor na descrição dos robôs. Assim, a descrição do sistema, para o caso parcial, é dada pela equação 6.35

$$\begin{aligned}
\varphi_i^s &: \left[(\varphi_i^{R_{S1}} \wedge \varphi_i^{s_1}) \wedge (\varphi_i^{R_{S2}} \wedge \varphi_i^{s_2}) \wedge \varphi_i^{NCol} \wedge \varphi_i^{BeR}\right] \\
\varphi_t^s &: \left[(\varphi_t^{R_{S1}} \wedge \varphi_t^{s_1}) \wedge (\varphi_t^{R_{S2}} \wedge \varphi_t^{s_2}) \wedge \varphi_t^{NCol} \wedge \varphi_t^{BeR}\right] \\
\varphi_g^s &: \Box \Diamond \left[(NCol = true) \wedge \varphi_g^{BeR}\right]
\end{aligned} \tag{6.35}$$

Uma vez que o modelo do robô já foi apresentado em 6.32, a seguir, só é descrito o modelo de seu sensor (φ^{s_i}) na equação 6.36, onde vale destacar a restrição (definida nas linhas 5, 6, 7 e 8) do robô só se deslocar para uma célula onde um OPD pode estar, quando seu sensor está observando essa posição.

$$\begin{aligned}
&\varphi_i^{s_i} : (s_i = 0) \\
&\varphi_t^{s_i} : \begin{cases}
\neg \bigcirc [(R_{Si} = 6) \vee (R_{Si} = 9) \vee (R_{Si} = 11) \vee (R_{Si} = 14)] \rightarrow \bigcirc (s_i \neq 10) \\
\bigwedge \neg \bigcirc [(R_{Si} = 7) \vee (R_{Si} = 10) \vee c(R_{Si} = 12) \vee (R_{Si} = 15)] \rightarrow \bigcirc (s_i \neq 11) \\
\bigwedge \neg \bigcirc [(R_{Si} = 9) \vee (R_{Si} = 14)] \rightarrow \bigcirc (s_i \neq 13) \\
\bigwedge \neg \bigcirc [(R_{Si} = 12) \vee (R_{Si} = 15)] \rightarrow \bigcirc (s_i \neq 16) \\
\bigwedge (s_i \neq 10) \rightarrow \bigcirc (R_{Si} \neq 10) \\
\bigwedge (s_i \neq 11) \rightarrow \bigcirc (R_{Si} \neq 11) \\
\bigwedge (s_i \neq 13) \rightarrow \bigcirc (R_{Si} \neq 13) \\
\bigwedge (s_i \neq 16) \rightarrow \bigcirc (R_{Si} \neq 16)
\end{cases}
\end{aligned} \tag{6.36}$$

Modelagem das especificações

Por último, na equação 6.37 é definida a especificação do problema de busca e resgate referente ao exemplo mostrado na figura 21

$$\begin{array}{l} \varphi_i^{BeR} : \left(\varphi_i^{Obj_1} \wedge \varphi_i^{Obj_2}\right) \\ \varphi_t^{BeR} : \left(\varphi_t^{Obj_1} \wedge \varphi_t^{Obj_2}\right) \\ \varphi_g^{BeR} : \left(\varphi_g^{Obj_1} \wedge \varphi_g^{Obj_2}\right) \end{array} \tag{6.37}$$

Na equação 6.37, as fórmulas φ^{Obj_1} e φ^{Obj_2} correspondem às descrições das especificações de busca e resgate dos objetos 1 e 2, respectivamente. Nas equações 6.38 e 6.39 são apresentadas as descrições destas especificações.

$$\begin{array}{l} \varphi_i^{Obj_1} : (Obj_1 = false) \\ \varphi_t^{Obj_1} : \left\{ \begin{array}{l} \neg \left[(obE_1^1 = true) \vee (obE_1^2 = true)\right] \rightarrow \left[(\bigcirc Obj_1) = Obj_1\right] \\ \bigwedge \left[(obE_1^1 = true) \vee (obE_1^2 = true)\right] \rightarrow \bigcirc (Obj_1 = true) \end{array} \right. \\ \varphi_g^{Obj_1} : (Obj_1 = true) \end{array} \tag{6.38}$$

$$\begin{array}{l} \varphi_i^{Obj_2} : (Obj_2 = false) \\ \varphi_t^{Obj_2} : \left\{ \begin{array}{l} \neg \left[(obE_2^1 = true) \vee (obE_2^2 = true)\right] \rightarrow \left[(\bigcirc Obj_2) = Obj_2\right] \\ \bigwedge \left[(obE_2^1 = true) \vee (obE_2^2 = true)\right] \rightarrow \bigcirc (Obj_2 = true) \end{array} \right. \\ \varphi_g^{Obj_2} : (Obj_2 = true) \end{array} \tag{6.39}$$

Nas equações 6.40, 6.41, 6.42 e 6.43 são descritas as fórmulas $\varphi_t^{ob_1^1}$, $\varphi_t^{ob_1^2}$, $\varphi_t^{ob_2^1}$ e $\varphi_t^{ob_2^2}$, respectivamente. Na equação 6.40, é descrito o modelo referente ao objetivo do robô 1 buscar e coletar o objeto 1.

$$\begin{array}{l} \varphi_i^{ob_1^1} : (obC_1^1 = false) \wedge (obE_1^1 = false) \\ \varphi_t^{ob_1^1} : \left\{ \begin{array}{l} \neg \bigcirc (R_{S1} = Obj_1) \rightarrow \left[(\bigcirc obC_1^1) = obC_1^1\right] \\ \bigwedge \left[(obC_1^1 \neq true) \vee \bigcirc (R_{S1} \neq 5)\right] \rightarrow \left[(\bigcirc obE_1^1) = obE_1^1\right] \\ \bigwedge \bigcirc (R_{S1} = Obj_1) \rightarrow \bigcirc (obC_1^1 = true) \\ \bigwedge \neg \left[(obC_1^1 \neq true) \vee \bigcirc (R_{S1} \neq 5)\right] \rightarrow \bigcirc (obE_1^1 = true) \end{array} \right. \\ \varphi_g^{ob_1^1} : (obE_1^1 = true) \end{array} \tag{6.40}$$

Na equação 6.41, é descrito o modelo referente ao objetivo do robô 2 buscar e coletar o objeto 1.

$$
\begin{aligned}
&\varphi_i^{ob_1^2} : (obC_2^1 = false) \wedge (obE_2^1 = false) \\
&\varphi_t^{ob_1^2} : \begin{cases} \neg \bigcirc (R_{S2} = Ob_1) \rightarrow \left[(\bigcirc obC_1^2) = obC_1^2\right] \\ \wedge \left[(obC_1^2 \neq true) \vee \bigcirc (R_{S2} \neq 5)\right] \rightarrow \left[(\bigcirc obE_1^2) = obE_1^2\right] \\ \wedge \bigcirc (R_{S2} = Ob_1) \rightarrow \bigcirc (obC_1^2 = true) \\ \wedge \neg \left[(obC_1^2 \neq true) \vee \bigcirc (R_{S2} \neq 5)\right] \rightarrow \bigcirc (obE_1^2 = true) \end{cases} \\
&\varphi_g^{ob_1^2} : (obE_2^1 = true)
\end{aligned}
\tag{6.41}
$$

Na equação 6.42, é descrito o modelo referente ao objetivo do robô 1 buscar e coletar o objeto 2.

$$
\begin{aligned}
&\varphi_i^{ob_2^1} : (obC_2^1 = false) \wedge (obE_2^1 = false) \\
&\varphi_t^{ob_2^1} : \begin{cases} \neg \bigcirc (R_{S1} = Ob_2) \rightarrow \left[(\bigcirc obC_2^1) = obC_2^1\right] \\ \wedge \left[(obC_2^1 \neq true) \vee \bigcirc (R_{S1} \neq 8)\right] \rightarrow \left[(\bigcirc obE_2^1) = obE_2^1\right] \\ \wedge \bigcirc (R_{S1} = Ob_2) \rightarrow \bigcirc (obC_2^1 = true) \\ \wedge \neg \left[(obC_2^1 \neq true) \vee \bigcirc (R_{S1} \neq 8)\right] \rightarrow \bigcirc (obE_2^1 = true) \end{cases} \\
&\varphi_g^{ob_2^1} : (obE_1^2 = true)
\end{aligned}
\tag{6.42}
$$

Na equação 6.43, é descrito o modelo referente ao objetivo do robô 2 buscar e coletar o objeto 2.

$$
\begin{aligned}
&\varphi_i^{ob_2^2} : (obC_2^2 = false) \wedge (obE_2^2 = false) \\
&\varphi_t^{ob_2^2} : \begin{cases} \neg \bigcirc (R_{S2} = Ob_2) \rightarrow \left[(\bigcirc obC_2^2) = obC_2^2\right] \\ \wedge \left[(obC_2^2 \neq true) \vee \bigcirc (R_{S2} \neq 8)\right] \rightarrow \left[(\bigcirc obE_2^2) = obE_2^2\right] \\ \wedge \bigcirc (R_{S2} = Ob_2) \rightarrow \bigcirc (obC_2^2 = true) \\ \wedge \neg \left[(obC_2^2 \neq true) \vee \bigcirc (R_{S2} \neq 8)\right] \rightarrow \bigcirc (obE_2^2 = true) \end{cases} \\
&\varphi_g^{ob_2^2} : (obE_2^2 = true)
\end{aligned}
\tag{6.43}
$$

Vale destacar que nas equações 6.40, 6.41, 6.42 e 6.43, as variáveis obC_k^i e obE_k^i indicam que o objeto k foi coletado e entregue pelo robô i. Além disso, a variável Ob_k corresponde à variável de estado do ambiente associada ao k-ésimo OPD e indica a posição onde o objeto k está.

Para o exemplo de busca e resgate mostrado nesta seção, a estrutura de jogo que representada a aplicação é obtida, considerando observabilidade global, através das equações 6.31 (sistema) e 6.29. Para o caso com observabilidade parcial, o sistema e o ambiente são descritos nas equações 6.35 e 6.33, respectivamente.

Controlador considerando observabilidade global

Considerando observabilidade global, o controlador obtido para o PBR é representado pelo sistema de transições mostrado na figura 22. Nesta representação o termo $Ob_{\{1,2\}} : X$ indica que o ambiente definiu que o objeto (1 ou 2) está na posição X.

Por sua vez, os termos $obC_{k,i}$ e $obE_{k,i}$ indicam que o objeto k foi coletado e entregue, respectivamente, pelo robô R_{Si}.

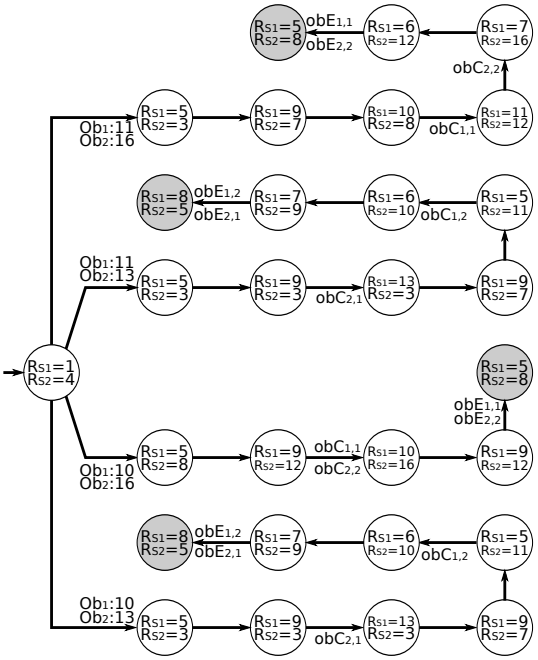


Figura 22: Representação do controlador obtido para o PBR com observabilidade global

No controlador mostrado na figura 22 é possível perceber a existência de quatro comportamentos, sendo um para cada combinação de células em que os OPDs podem estar. Estes comportamentos são:

- Comportamento 1 $\begin{cases} R_{S1} : 1 \mapsto 5 \mapsto 9 \mapsto 10 \mapsto \underline{11} \mapsto 7 \mapsto 6 \mapsto \mathbf{5} \\ R_{S2} : 4 \mapsto 3 \mapsto 7 \mapsto 8 \mapsto 12 \mapsto \underline{16} \mapsto 12 \mapsto \mathbf{8} \end{cases}$
- Comportamento 2 $\begin{cases} R_{S1} : 1 \mapsto 5 \mapsto 9 \mapsto \underline{10} \mapsto 9 \mapsto \mathbf{5} \\ R_{S2} : 4 \mapsto 8 \mapsto 12 \mapsto \underline{16} \mapsto 12 \mapsto \mathbf{8} \end{cases}$
- Comportamento 3 $\begin{cases} R_{S1} : 1 \mapsto 5 \mapsto 9 \mapsto \underline{13} \mapsto 9 \mapsto 5 \mapsto 6 \mapsto 7 \mapsto \mathbf{8} \\ R_{S2} : 4 \mapsto 3 \mapsto 7 \mapsto 11 \mapsto \underline{10} \mapsto 9 \mapsto \mathbf{5} \end{cases}$
- Comportamento 4 $\begin{cases} R_{S1} : 1 \mapsto 5 \mapsto 9 \mapsto \underline{13} \mapsto 9 \mapsto 5 \mapsto 6 \mapsto 7 \mapsto \mathbf{8} \\ R_{S2} : 4 \mapsto 3 \mapsto 7 \mapsto \underline{11} \mapsto 10 \mapsto 9 \mapsto \mathbf{5} \end{cases}$

Nas sequências acima, os números subscritos indicam as células onde os robôs coletaram um objeto e, em negrito, a posição onde ele foi entregue.

Controlador considerando observabilidade parcial

Para o caso em que é considerada observabilidade parcial, a representação do controlador obtido é mostrada na figura 23.

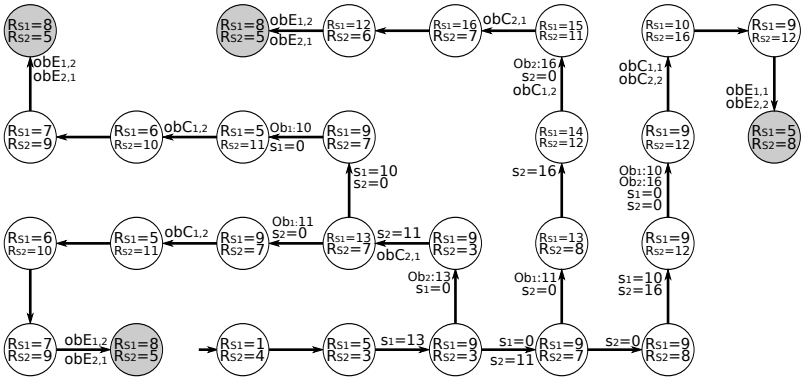


Figura 23: Representação do controlador obtido para o PBR com observabilidade parcial

Uma vez que existem quatro combinação de células onde os objetos 1 e 2 podem estar, no caso parcial também é possível perceber a existência de quatro comportamentos, são eles:

- Comportamento 1 $\begin{cases} R_{S1} : 1 \mapsto 5 \mapsto 9 \mapsto \underline{10} \mapsto 9 \mapsto \mathbf{5} \\ R_{S2} : 4 \mapsto 3 \mapsto 7 \mapsto \overline{8} \mapsto 12 \mapsto \underline{16} \mapsto 12 \mapsto \mathbf{8} \end{cases}$
- Comportamento 2 $\begin{cases} R_{S1} : 1 \mapsto 5 \mapsto 9 \mapsto 13 \mapsto 14 \mapsto 15 \mapsto \underline{16} \mapsto 12 \mapsto \mathbf{8} \\ R_{S2} : 4 \mapsto 3 \mapsto 7 \mapsto 8 \mapsto 12 \mapsto \underline{11} \mapsto 7 \mapsto 6 \mapsto \mathbf{5} \end{cases}$
- Comportamento 3 $\begin{cases} R_{S1} : 1 \mapsto 5 \mapsto 9 \mapsto \underline{13} \mapsto 9 \mapsto 5 \mapsto 6 \mapsto 7 \mapsto \mathbf{8} \\ R_{S2} : 4 \mapsto 3 \mapsto 7 \mapsto \overline{11} \mapsto \underline{10} \mapsto 9 \mapsto \mathbf{5} \end{cases}$
- Comportamento 4 $\begin{cases} R_{S1} : 1 \mapsto 5 \mapsto 9 \mapsto \underline{13} \mapsto 9 \mapsto 5 \mapsto 6 \mapsto 7 \mapsto \mathbf{8} \\ R_{S2} : 4 \mapsto 3 \mapsto 7 \mapsto \overline{11} \mapsto 10 \mapsto 9 \mapsto \mathbf{5} \end{cases}$

No controlador gerado para o caso com observabilidade parcial, alguns comportamentos são ligeiramente diferentes dos apresentados quando se considera observabilidade global. Isso ocorre porque, no caso global, a posição dos objetos é informada ao sistema no início da execução. Assim, o controlador implementa quatro estratégias distintas, uma para cada combinação possível das posições do objeto. Entretanto, os comportamentos 3 e 4, considerando observabilidade global, correpondem aos 3 e 4, na parcial.

No caso com observabilidade parcial, é preciso que o sensor de um robô detecte um objeto para que o sistema passe a saber sua posição. Assim, o sistema assume uma postura de busca dos objetos e, uma vez que eles foram encontrados, os robôs pegam os objetos e levam a suas respectivas posições de destino.

Neste sentido, no controlador com observabilidade parcial aparecem comportamentos como o de um robô detectar um objeto que será coletado por outro agente do sistema, por exemplo, no estado em que $R_{S1} = 13$ e $R_{S2} = 8$, o sensor s_2 detecta que o objeto 2 (coletado posteriormente por R_{S1}) está na célula 16. Vale destacar que estes comportamentos foram gerados "naturalmente", ou seja, não foi definida uma especificação associada a este comportamento.

É muito importante destacar também que, tanto para o caso com observabilidade global quanto parcial, não foi necessário definir uma especificação puramente de busca. Ou seja, não foi necessário criar uma especificação para forçar que os robôs percorressem o espaço de trabalho para encontrar os objetos, e só então coletá-los, sendo esse comportamento gerado naturalmente.

6.4 EXPERIMENTOS REALIZADOS

Neste trabalho, a arquitetura e a estrutura física apresentados na dissertação de mestrado[54] foram utilizadas para implementação dos controladores obtidos. Entretanto, não houve tempo suficiente para a realização de experimentos considerando todos os exemplos apresentados neste capítulo.

Assim, optou-se por implementar apenas o caso que foi tratada na dissertação anterior [54], que corresponde, neste trabalho, ao problema de alcançabilidade simples com observabilidade parcial. De forma geral, o experimento teve como objetivo validar o uso dos controladores obtidos a partir de um método baseado em estruturas de jogos, o apresentado em [56], na coordenação de SMRs. Além disso, os resultados obtidos nos experimentos foram utilizados para comparação da abordagem apresentada neste trabalho com a apresentada em [54].

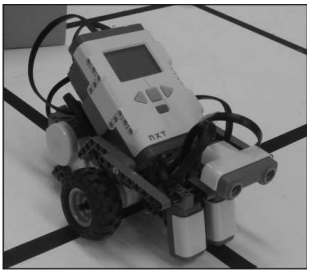
Especificamente, teve-se como objetivo verificar se é possível implementar o tipo de controlador gerado pelo método proposto em Piterman *et al.* (2006)[56] (obtido em formato de autômato textual) num sistema real e a complexidade desta implementação. Além disso, o controlador gerado garante que as especificações do sistema são sempre realizadas, a não ser que existam erros nos modelos de algum dos elementos que compõem a estrutura de jogo (robôs, sensores, etc.). Assim, os experimentos também serviram para validar os princípios de modelagem propostos neste trabalho.

6.4.1 Arquitetura utilizada

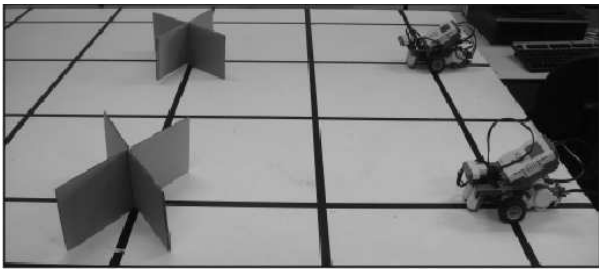
Na dissertação de mestrado [54], desenvolvida no PGEAS, foi apresentada uma arquitetura para SMRs que utiliza coordenadores baseados na teoria de controle supervisório e de autômato-jogo. De forma geral, a arquitetura apresenta uma estrutura de controle híbrida, separando a parte do controle de tarefas/atividades (coordenador) da parte associado ao controle de trajetória entre células.

Uma vez que o objetivo principal da arquitetura foi permitir a avaliação dos coordenadores obtidos, a parte referente ao controle de trajetória foi implementada de forma bastante simples, através da utilização de movimentos de translação (ir para frente) e rotação (girar 90°para a esquerda e 90°para a direita) desacoplados.

Os experimentos foram implementados utilizando robôs LEGO *mindstorms* (vide figura 24(a)), os quais eram compostos por dois tipos de sensores: infravermelho e sonar. Além disso, foi adotado o espaço de trabalho mostrado na figura 24(b), onde cada cruzamento representa uma célula.



(a) Robô LEGO *mindstorms*



(b) Espaço de trabalho

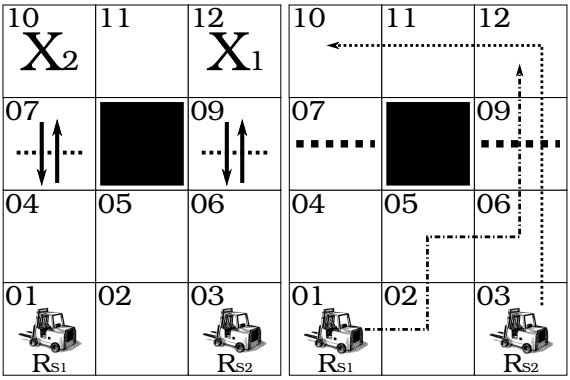
Figura 24: Robô e espaço de trabalho considerados nos experimentos apresentados em [54]

Assim, os sensores infravermelhos foram utilizados para navegação neste espaço de trabalho, indicando que a célula de destino foi alcançada ou que o movimento de rotação terminou. Por sua vez, um sonar foi utilizado como sensor do robô nos casos de observabilidade parcial, possibilitando a detecção de obstáculos e adversários na célula à frente.

6.4.2 Descrição do experimento

Nos experimentos realizados, foi considerado o espaço de trabalho mostrado na figura 25(a), onde existe um obstáculo fixo entre as células 7 e 9, duas portas e dois robôs do sistema, nas posições 1 e 3.

Considerando os objetivos do robô R_{S1} alcançar a posição 12 e o R_{S2} a célula 10, a trajetória executada pelos robôs (de acordo com o controlador mostrado na figura 15) é mostrada na figura 25(b). É importante ressaltar que esse caso foi resolvido na seção 6.1.



(a) Espaço de trabalho (b) Trajetória

Figura 25: Espaço de trabalho e trajetória referentes ao experimento

Durante os experimentos, foram realizados testes considerando quatro situações. Na primeira, a porta situada em 9 (a única utilizada pelos robôs) fica sempre aberta. Na segunda situação, a porta fecha por 5 segundos antes da passagem do robô R_{S2} e depois permanece aberta. Na terceira, a porta inicia aberta e fecha após a passagem de R_{S2}. Após 5 segundos, a porta volta a abrir, permitindo a passagem de R_{S1}.

Na quarta situação, a porta fecha por 5 segundos antes da passagem dos dois robôs, ou seja, antes R_{S2} passar ela fica fechada por 5 segundos. Depois da passagem de R_{S2} a porta volta a fechar, impedindo a passagem de R_{S1}. Após 5 segundos, a porta volta a abrir e este robô a atravessa. Vale destacar ainda que, para todas as situações onde a porta fecha para um dos robôs, foram testado os casos em que a porta fecha enquanto um robô está se deslocando para a posição associada a ela (célula 9).

Ao final do experimento, foi verificado que em todos os casos analisados os robôs conseguem alcançar suas respectivas posições de destino, indicando que os tipos de controladores gerados são realizáveis e que as descrições dos elementos de modelagem (robôs, sensores e portas) condizem com os comportamentos dos objetos que eles representam. Na tabela 1 são apresentados os resultados obtidos em cada uma das situações citadas a cima.

Na tabela 1, o valor "$-$" associado à porta P_1, indica que o funcionamento desta porta não influencia no experimento. Por sua vez, os valores 0 e 5 em P_2, indicam que está porta ficou fechada por 0 ou 5 segundos, para o robô correspondente (R_{S1} ou R_{S2}).

Tabela 1: Resultados dos experimentos

Experimento	P_1	P_2		tempo médio (s)
		R_{S1}	R_{S2}	
1	-	0	0	21,6
2	-	0	5	25,4
3	-	5	0	25,2
4	-	5	5	27,9

Comparando os resultados apresentados na dissertação de mestrado [54] e os obtidos neste trabalho, foi possível perceber que o tempo que os robôs levam para alcançar suas posições de destino diminuiu significativamente (em média 50 segundos para [54] e 22 segundos neste trabalho, considerando as portas abertas), principalmente pelo fato do controlador obtido a partir do método proposto em Piterman *et al.* (2006)[56] considerar o movimento simultâneo de robôs.

Além disso, o tipo de controlador apresentado neste trabalho considera a possibilidade de um robô do sistema detectar, durante a sua movimentação, uma mudança no *status* do ambiente (por exemplo uma porta fecha enquanto um robô tenta atravessá-la), o que não era permitido no trabalho anterior [54]. Vale destacar ainda que a falta de informação sobre os experimentos realizados no trabalho anterior [54] dificultou um pouco a comparação.

6.5 DISCUSSÕES

Sobre os princípios de modelagem propostos

Sobre os elementos de modelagem propostos neste trabalho, existem alguns aspectos que devem ser destacados. O primeiro deles é a hipótese feita sobre a porta. É importante ressaltar que se não fosse assumido que, para ambientes compostos por portas, há cooperação com o sistema, as portas poderiam se manter fechadas e, assim, sempre impedir a passagem dos robôs do sistema. Desta forma, não seria possível gerar controladores.

Outro aspecto que deve ser destacado é a existência de solução em ambientes compostos por robôs adversários. Nestes casos, dependendo do formato do espaço de trabalho e da quantidade de robôs do sistema, pode não ser possível gerar um controlador que garanta o cumprimento da especificação. Isso porque, pode ser necessário que os robôs do sistema cooperem, a fim de que um adversário, assumindo uma postura de perseguição, não consiga impedir que os robôs realizem os objetivos associados a eles.

Por exemplo, no problema de alcançabilidade cooperativa, como o adversário não consegue perseguir os dois robôs ao mesmo tempo, um deles pode atravessar a região de movimentação do adversário e realizar os objetivos. Entretanto, quando é considerado o mesmo exemplo, mas com robôs com possibilidade de falha, não é possível garantir a realização da especificação do sistema. Isso ocorre neste exemplo porque, se um dos robôs falhar, o adversário pode perseguir o outro robô e impedir que este atravesse a região de movimentação e, consequentemente, alcance as posições de destino.

Contudo, é importante ressaltar que esta não é uma restrição associada ao método apresentado em Piterman *et al.* (2006) [56] ou aos princípios de modelagem propostos neste trabalho. Nestes casos, é a própria configuração da aplicação que impede que sejam gerados controladores com garantias de cumprimento das especificações.

Comparação com as abordagens apresentadas em Pavei (2011)

Considerando a abordagem utilizando o método proposto em Piterman *et al.*(2006) [56] e os princípios de modelagem proposto neste trabalho, foi possível constatar que as estratégias geradas, apesar de não considerarem o tempo como parâmetro de escolha do controlador, garantem que as especificações do sistema são realizadas num número mínimo de passos.

Entretanto, o fato de não modelar o tempo, teve como consequência a geração de controladores mais conservadores em relação aos movimentos dos robôs, onde (no caso de ambientes com portas, por exemplo) os robôs esperam que uma porta abra, ao invés de tentar passar pela outra. Contudo, é importante ressaltar que estratégias em que o robô fica alternando entre as

posições adjacentes às portas, até encontrar uma delas aberta, não garantem que o objetivo será realizado. Isso porque poderia haver casos em que o robô sempre encontra as portas fechadas, apesar delas abrirem em alguns momentos.

Neste sentido, é importante ressaltar que os controladores baseados em autômato-jogo, obtidos no trabalho apresentado na dissertação anterior [54], implementam estratégias em que os robôs ficam se movendo pelo espaço de trabalho em busca de uma porta aberta. Além disso, esta forma de cooperação (assumindo que a porta vai abrir quando um robô estiver em sua frente) foi gerada automaticamente pelo método utilizado.

Sobre os controladores, apresentados em [54], obtidos com abordagens de controle supervisório, é importante destacar que a característica permissiva do controlador, torna necessária a utilização de um mecanismo para escolha da próxima ação, que irá definir qual, dentre as ações permitidas pelo supervisor, será tomada a cada passo. Assim, a garantia de realização das tarefas e a eficiência do controlador passam a depender do mecanismo de escolha de ação utilizado.

Escalabilidade

Outro aspecto importante a ser discutido diz respeito à escalabilidade dos controladores gerados. No trabalho apresentado em [54] (considerando as abordagens de TCS monolítica e autômato-jogo), são mostrados os resultados para o problema de alcançabilidade simples, considerando sistemas com um, dois e três robôs e com observabilidade parcial. Neste trabalho, o mesmo problema foi resolvido. Na tabela 2 são apresentados os resultados obtidos em Pavei (2011) [54] e neste trabalho, que consideraram um espaço de trabalho com 16 células e um ambiente composto por duas portas.

Vale ressaltar que a modelagem das aplicações, nas abordagens utilizadas em Pavei (2011) e na considerada neste trabalho, resultaram num espaço de estado de mesma ordem de grandeza. Na tabela 2 é apresentada a ordem de grandeza do número de estados (coluna 2) da aplicação tratada pelas abordagens de controle supervisório (TCS), autômato-jogo e estrutura de jogo, considerando SMRs com 1, 2 e 3 robôs.

Tabela 2: Dados sobre os controladores gerados

Robôs	Espaço de estados da aplicação	Espaço de estados do controlador		
		TCS	AJT	Estrutura de jogo
1	10^3	68	31	8
2	10^5	4360	725	9
3	10^7	$> 10^5$	13735	15

Na tabela 2 também é apresentado o número de estados dos controladores obtidos usando a abordagem de controle supervisório (coluna 3), de autômato-jogo (coluna 4) e usando a abordagem baseada em estrutura de jogo proposta neste trabalho (coluna 5).

Analisando a tabela, é possível perceber que o número de estados dos controladores apresentados em [54] é muito maior que os obtidos neste trabalho e que este número cresce exponencialmente com o aumento da quantidade de robôs do sistema, o que pode inviabilizar a implementação dos coordenadores gerados em robôs ou plataformas com menor poder computacional.

6.6 CONCLUSÃO

Neste capítulo, foram apresentados os modelos de especificação para as classes de problemas abordadas e exemplos de cada uma delas, os quais foram utilizados para ilustrar a modelagem da planta, através dos princípios propostos, e a obtenção do controlador. Na sequência, foram discutidos os principais resultados obtidos neste trabalho, focando na eficiência das estratégias geradas e na escalabilidade do método.

7 CONCLUSÕES E PERSPECTIVAS

Através do estudo e análise de diversos trabalhos, foram encontrados métodos baseados na teoria de jogos (autômato-jogo e estrutura de jogo) capazes de gerar estratégias para coordenação de SMRs, inclusive em casos onde é necessária a cooperação com o ambiente. Uma vez que o método proposto em Piterman *et al.* (2006)[56] foi o único que permitia definir a forma como a cooperação entre sistema e ambiente ocorre, este foi o escolhido para modelar as classes de aplicações tratadas neste trabalho.

Com base na abordagem apresentada em [56], foram propostos diversos elementos de modelagem, capazes de representar os principais componentes presentes nos SMRs. Além disso, também foi proposta uma metodologia para descrever a estrutura de jogo que representa uma aplicação, através da composição de elementos de modelagem. Esses princípios de modelagem foram validados através de seu uso na solução de problemas de coordenação de SMRs, considerando as classes de tarefas discutidas no capítulo 4 (alcançabilidade simples, cooperativa e busca e resgate). A partir dos resultados obtidos, foi possível constatar que tanto os modelos dos elementos quanto a metodologia para composição deles foram eficazes na modelagem das aplicações, possibilitando a obtenção de coordenadores para realização das tarefas em todos os casos abordados.

Em comparação com os resultados apresentados em outro trabalho de dissertação recente [54], foi possível perceber que os controladores obtidos neste trabalho podem ser implementados de forma mais eficientes, em relação ao tempo de execução e ao no número de estados do controlador. Pois, além de considerar a possibilidade de movimentação simultânea de robôs do sistema, os controladores obtidos minimizam o número de passos necessários para realização dos objetivos do sistema.

Para os controladores apresentados em Pavei (2011)[54], além da obrigatoriedade dos agentes se moverem um de cada vez, nos obtidos pela abordagem de TCS, a garantia de realização dos objetivos e a eficiência dependem do mecanismo para escolha de ação adotado. Por sua vez, na abordagem baseada em autômato-jogo, a hipótese que deve ser satisfeita é a de que em algum momento o robô encontrará uma porta aberta, enquanto que neste trabalho foi adotada apenas a hipótese de que a porta irá abrir. Vale destacar ainda que os tempos de execução nos experimentos apresentados neste trabalho foram bem menores que os apresentados em [54].

É importante ressaltar que, devido à ausência do conceito de tempo no método proposto por Piterman *et al.* (2006)[56], sua minimização não pode ser utilizada como parâmetro na escolha do controlador. Entretanto,

os algoritmos propostos em Piterman *et al.*[29, 56] minimizam o número de passos necessários para levar o sistema de um estado inicial até um estado em que sua especificação esteja sendo satisfeita.

Devido a ausência de tempo na descrição dos elementos de modelagem, foi assumido que todos os robôs (mesmo o adversário) se movem com a mesma velocidade. Durante os experimentos, foi possível notar que em aplicações onde todos os robôs pertencem ao sistema, esse fato não gera problemas, pois os robôs mais rápidos esperam até o mais lento terminar seu movimento, só então é executada uma nova transição no controlador. Entretanto, no caso de ambientes com robôs adversários, esse fato pode ser problemático, assim, ainda deverão ser realizados experimentos considerando ambientes compostos por robôs em diferentes velocidades.

Neste trabalho também foi tratado o aspecto de tolerância à falta em SMRs, tendo sido obtidos controladores que garantem, mesmo em situações de falhas em robôs, que os objetivos do sistema sejam atendidos. Contudo, vale enfatizar que existem situações em que o problema tratado não possui soluções com garantias, como o caso em que os robôs do sistema precisam passar (obrigatoriamente) por uma célula específica da região de movimentação de um adversário. Em casos como este, pode ser necessário definir hipóteses adicionais sobre os elementos do ambiente.

O último aspecto tratado neste trabalho foi a escalabilidade do método proposto em Piterman *et al.* (2006)[56]. Comparando os resultados obtidos neste trabalho com os obtidos na dissertação [54], é possível perceber que os controladores obtidos considerando o método proposto em Piterman *et al.* (2006) [56] e utilizando os princípios propostos neste trabalho, foram muito menores que os obtidos em [54].

PERSPECTIVAS DE TRABALHO

Sobre o método

Como trabalho futuro propõe-se a extensão do método proposto em [56], de modo que se possa modelar o tempo, e o estudo do impacto da modelagem do tempo na eficiência das estratégias geradas. Além disso, também é proposto o desenvolvimento de um método para obtenção de controladores que, ao invés de considerar o ambiente como adversário, adota probabilidades de ocorrência para seus eventos.

Sobre as ferramentas

Outra perspectiva interessante é o desenvolvimento de uma ferramenta gráfica para modelagem de plantas, de modo que toda a descrição em asser-

ções de lógica temporal possa ficar transparente para o projetista. Além disso, seria interessante desenvolver um aplicativo (ou adaptar um existente) para simulação dos controladores obtidos, de modo que se possa considerar, antes da implementação, a influência de aspectos como as características físicas do espaço de trabalho e dos robôs e os tipos de controladores de baixo nível que serão implementados nos agentes.

Outro ponto importante é a realização de uma nova implementação, mais eficiente, dos algoritmos propostos em Piterman *et al.* (2006) [29, 56], permitindo que se trabalhe com espaços de trabalho maiores e com mais elementos do sistema e do ambiente.

Sobre as aplicações

Outras perspectivas, são a resolução de problemas de coordenação de SMRs mais complexos, considerando mais robôs e espaços de trabalho maiores, e a implementação dos controladores gerados.

Neste sentido, também é proposta a melhoria da arquitetura proposta em Pavei (2011) [54], de modo a tornar a implementação de controladores baseados em SED ainda mais fácil.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] R. Alur. Timed automata. In *Computer Aided Verification*, pages 688–688. Springer, 1999.

[2] R. Alur and D. Dill. Automata for modeling real-time systems. *Automata, languages and programming*, pages 322–335, 1990.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[4] M. Andersen, R. Jensen, T. Bak, and M. Quottrup. *Motion planning in multi-robot systems using timed automata*. Department of Control Engineering, Aalborg University, 2004.

[5] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. 1998.

[6] T. Balch. The impact of diversity on performance in multi-robot foraging. In *Proceedings of the third annual conference on Autonomous Agents*, pages 92–99. ACM, 1999.

[7] T. Balch and M. Hybinette. Social potentials for scalable multi-robot formations. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 73–80. IEEE, 2000.

[8] S. Behnke and R. Rojas. A hierarchy of reactive behaviors handles complexity. *Balancing reactivity and social deliberation in multi-agent systems*, pages 125–136, 2001.

[9] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. Uppaal-tiga: Time for playing games! In *Proceedings of the 19th international conference on Computer aided verification*, pages 121–125. Springer-Verlag, 2007.

[10] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime. Uppaal tiga user-manual, 2007.

[11] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, D. Lime, et al. Uppaal-tiga: Timed games for everyone. 2006.

[12] M. Blej and M. Azizi. Modeling and analysis of a real-time system using the networks of extended petri. *Journal of Computers*, 4(7):641–645, 2009.

[13] C. Cassandras and S. Lafortune. *Introduction to discrete event systems*. Springer, 2008.

[14] F. Cassez. Efficient on-the-fly algorithms for partially observable timed games. *Formal Modeling and Analysis of Timed Systems*, pages 5–24, 2007.

[15] F. Cassez, A. David, E. Fleury, K. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. *CONCUR 2005–Concurrency Theory*, pages 66–80, 2005.

[16] E. Clarke. Model checking. In *Foundations of software technology and theoretical computer science*, pages 54–56. Springer, 1997.

[17] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Logics of Programs*, pages 52–71, 1982.

[18] J. Conway. On numbers and games. *New York*, 1976.

[19] J. Cury. Teoria de controle supervisório de sistemas a eventos discretos. *V Simpósio Brasileiro de Automação Inteligente (Minicurso)*, 2001.

[20] A. David, K. Larsen, S. Li, and B. Nielsen. Cooperative testing of timed systems. *Electronic Notes in Theoretical Computer Science*, 220(1):79–92, 2008.

[21] G. Fainekos, A. Girard, H. Kress-Gazit, and G. Pappas. Temporal logic motion planning for dynamic robots. 45(2):343–352, 2009.

[22] G. Fainekos, H. Kress-Gazit, and G. Pappas. Temporal logic motion planning for mobile robots. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2020–2025. IEEE, 2005.

[23] C. Finucane, G. Jing, and H. Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1988–1993. IEEE, 2010.

[24] C. Griffin. Game theory: Penn state math 486 lecture notes, 2011.

[25] L. Iocchi, D. Nardi, and M. Salerno. Reactivity and deliberation: a survey on multi-robot systems. *Balancing reactivity and social deliberation in multi-agent systems*, pages 9–32, 2001.

[26] B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Computer Aided Verification*, pages 258–262. Springer, 2007.

[27] D. Jung and A. Zelinsky. Grounded symbolic communication between heterogeneous cooperating robots. *Autonomous Robots*, 8(3):269–292, 2000.

[28] J.-P. Katoen. *Principles of model checking*. 2004.

[29] Y. Kesten, N. Piterman, and A. Pnueli. Bridging the gap between fair simulation and trace inclusion. *Information and Computation*, 200(1):35–61, 2005.

[30] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. *Automata, Languages and Programming*, pages 1–16, 1998.

[31] E. Klavins and D. Koditschek. A formalism for the composition of concurrent robot behaviors. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 4, pages 3395–3402. IEEE, 2000.

[32] D. Kozen. Results on the propositional [mu]-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

[33] D. Kreps. Game theory and economic modelling. *OUP Catalogue*, 1992.

[34] H. Kress-Gazit, G. Fainekos, and G. Pappas. Temporal logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on*, 25(6):1370–1381, 2009.

[35] S. Kripke. Semantical considerations on modal logic. *Acta philosophica fennica*, 16(1963):83–94, 1963.

[36] C. Kube and E. Bonabeau. Cooperative transport by ants and robots. *Robotics and autonomous systems*, 30(1):85–101, 2000.

[37] X. Liu and S. Smolka. Simple linear-time algorithms for minimal fixed points. *Automata, Languages and Programming*, pages 53–66, 1998.

[38] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *STACS 95*, pages 229–242. Springer, 1995.

[39] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification*, volume 1. springer, 1992.

[40] S. Maoz and Y. Sa'ar. Aspectltl: an aspect language for ltl specifications. In *Proceedings of the tenth international conference on Aspect-oriented software development*, pages 19–30. ACM, 2011.

[41] N. Markey and P. Schnoebelen. Mu-calculus path checking. *Information processing letters*, 97(6):225–230, 2006.

[42] M. Matarić. Learning social behavior. *Robotics and Autonomous Systems*, 20(2):191–204, 1997.

[43] K. McMillan. The smv system, 1992.

[44] K. McMillan. *Symbolic model checking: an approach to the state explosion problem*. PhD thesis, 1992.

[45] K. McMillan. The smv language. *Cadence Berkeley Labs*, pages 1–49, 1999.

[46] L. Molina. Desenvolvimento de uma arquitetura de navegação deliberativa para robôs móveis utilizando a teoria de controle supervisório. Dissertação, Universidade Federal do Rio de Janeiro, 2010.

[47] J. Morrow. Game theory for political scientists. 1994.

[48] R. Myerson. *Game theory: analysis of conflict*. Harvard Univ Pr, 1997.

[49] D. Niwiski and I. Walukiewicz. Games for the [mu]-calculus. *Theoretical Computer Science*, 163(1-2):99–116, 1996.

[50] F. Noreils. Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The International Journal of Robotics Research*, 12(1):79, 1993.

[51] L. Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2):220–240, 1998.

[52] L. Parker. Cooperative robotics for multi-target observation. *Intelligent Automation and Soft Computing*, 5:5–20, 1999.

[53] L. Parker. Lifelong adaptation in heterogeneous multi-robot teams: Response to continual variation in individual robot performance. *Autonomous Robots*, 8(3):239–267, 2000.

[54] J. Pavei. Planejamento e coordenação de sistemas multi-robôs. Master's thesis, 2011.

[55] J. Pavei, J.-M. Farines, and J. Cury. Coordenação de sistemas de robôs utilizando autômato-jogo temporizado. *SBAI 2001*, 2011.

[56] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive (1) designs. In *Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

[57] A. Pneuli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 746–757. IEEE, 1990.

[58] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.

[59] A. Pnueli, Y. Sa'ar, and L. Zuck. Jtlv: A framework for developing verification algorithms. In *Computer Aided Verification*, pages 171–174. Springer, 2010.

[60] M. Quottrup, T. Bak, and R. Zamanabadi. Multi-robot planning: A timed automata approach. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4417–4422. IEEE, 2004.

[61] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[62] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25:206–230, 1987.

[63] V. B. Ribeiro. Concepção de sistemas de robôs e de sensores em redes. PIBIC, Universidade Federal de Santa Catarina, 2010.

[64] V. B. Ribeiro. Using formal verification techniques to coordenate multi-robot systems. PFC, Universidade Federal de Santa Catarina, 2011.

[65] J. Richard Büchi. Symposium on decision problems: On a decision method in restricted second order arithmetic. *Studies in Logic and the Foundations of Mathematics*, 44:1–11, 1966.

[66] J. Weibull. *Evolutionary game theory*. The MIT press, 1997.

[67] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. Murray. Tulip: a software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 313–314. ACM, 2011.

[68] W. Wonham and P. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25:637, 1987.

APÊNDICE A – Modelagem em SMV

SMV é uma linguagem desenvolvida para modelagem sistema em máquina de estados, visando a verificação de especificações descritas em lógica temporal. Proposta por McMillan, esta linguagem é bastante vasta, sendo possível descrever estruturas de decisão e repetição, dentre outras características.

Todavia, como o objetivo deste apêndince é apenas forncer as informações necessárias para o projetista traduzir as asserções em LT, utilizadas para descrever a planta no método proposto em Piterman *et al.* (2006), para o padrão SMV, somente uma parte desta linguagem será descrita. Para mais informações sobre SMV, sugere-se a leitura dos trabalhos apresentados em [43], [44] e [45].

Nas seções a seguir são tratados os aspectos desta linguagem utilizados para modelagem no método proposto em [56]. Além disso, o exemplo apresentado na seção 3.5 para explicar a modelagem em asserções de LT será reutilizada neste capítulo para exemplificar o processo de tradução para SMV.

A linguagem SMV

Considerando o método proposto em Piterman *et al.* (2006), uma planta é descrita com base no esquema mostrado no código A.0.1.

```
1  MODULE main
2    VAR
3      e : env("variáveis de estado do sistema passadas como
4            parâmetros");
5      s : sys("variáveis de estado do ambiente passadas como
6            parâmetros");
7
8  MODULE env("parâmetros")
9    VAR     // Declaração das variáveis de estado do ambiente
10
11   ASSIGN  // Declaração das condições iniciais
12
13   TRANS   // Declaração da dinâmica do ambiente
14
15   JUSTICE // Declaração das hipóteses sobre o ambiente
16
17 MODULE sys("parâmetros")
18   VAR     // Declaração das variáveis de estado do sistema
19
20   ASSIGN  // Declaração das condições iniciais
21
22   TRANS   // Declaração da dinâmica do sistema
23
24   JUSTICE // Declaração dos objetivos do sistema
```

Código A.0.1: Esquema de uma planta em SMV

Analisando o código A.0.1, é possível perceber que o sistema e o ambiente são descritos em módulos distintos e estes são instanciados num módulo principal. Além disso, cada módulo é dividido em quatro partes: VAR, ASSIGN, TRANS e JUSTICE, onde são declaradas as variáveis de estado, a condição inicial, a dinâmica e as condições de justiça. A seguir são discutidas cada uma destas partes.

Variáveis e condição inicial

SMV aceita uma grande quantidade de tipos de variáveis, contudo, na parte desta linguagem utilizada na implementação dos algoritmos de Piterman *et al.*, sem perda de generalidade, são definidos apenas variáveis booleanas e inteiras.

De uma forma geral, as variáveis são declaradas da seguinte forma:

```
1  VAR
2          a : boolean;
3          b : {-1,0,1,12};
4          c : array 0..2 of boolean;
5          d : outroModulo(a,b);
```

Código A.0.2: Declaração de variáveis

No código A.0.2 é apresentada a declaração de quatro variáveis. a é uma variável booleana, b é uma variável definida sobre o conjunto de inteiros $\{-1,0,1,12\}$, c é um *array* com 3 variáveis booleanas ($c[0], c[1], c[2]$) e d é a instanciação de um módulo de nome $outroModulo$.

Uma vez definidas as varipaveis de estado de um módulo, sua inicialização pode ser feita como mostrado no código A.0.3.

```
1  INIT
2    init(a) := TRUE;
3    init(b) := 1;
4    init(c[0]) := FALSE;
5    init(c[1]) := FALSE;
6    init(c[2]) := TRUE;
```

Código A.0.3: Inicialização de variáveis

Vale destacar que, como d é a instanciação de um módulo, não é declara um condição incial para ela.

Dinâmica

Para declarar a dinâmica do sistema ou do ambiente, são utilizados o

Tabela 3: Operadores lógicos e temporais

Operadores			
Conjunção	&	Diferença	!=
Disjunção	|	Implicação	->
Negação	!	*Next*(◯)	next( )
Igualdade	=		

conjunto de operadores mostrados na tabela 3.

Utilizando estes operadores, a dinâmica de cada módulo pode ser descrita através de um conjunto de asserções, as quais indicam os valores possíveis para cada variável no próximo estado, dados os valores atuais. No código A.0.4 é apresentado um exemplo bastante simples de dinâmica.

```
1  TRANS
2    (!a -> next(b=-1)|(b=0)) )&
3    (a & ((b=-1)|(b=0)) -> next(b=1) )&
4    (a & (b=1) -> next(b=12) )&
5    (a & (b=12) -> next(b=0) );
```

Código A.0.4: Exemplo de dinâmica

No código A.0.4 é apresentado um exemplo bastante simples de dinâmica, onde é definido que, se o valor da variável de estado a é falso ($!a$), no próximo passo b só pode assumir os valores -1 ou 0. Caso a seja verdade (indicado por a), é realizada a sequência $0 \mapsto 1 \mapsto 12 \mapsto 0 \mapsto \cdots$, indefinidamente.

É muito importante destacar que, como não é definido como o valor de a é alterado, isso implica que esta varipavel pode ser alterada sem restrições, ou seja, em qualquer passo ela pode passar de falsa para verdade e vice-versa. Além disso, ao final da última asserção que descreve a dinâmica, é utilizado um sinal “;”, como pode ser visto na linha 5 do código A.0.4.

Condição de justiça

Uma condição de justiça é um fómula lógico temporal na forma $\square\lozenge\, p$, onde p é uma fórmula lógica. No código A.0.5 é apresentado um exemplo de declaração de condição de justiça, considerando a implementação do método proposto em [56].

É importante destacar no código A.0.5 a ausência do operador $\square\lozenge$, que já está implicito em cada expressão, de modo que a condição de justiça

```
1  JUSTICE
2    a;
3    (a & (b=12));
```

Código A.0.5: Exemplo de condição de justiça

Tabela 4: Equivalência entre valores

Equivalência de valores							
Asserção	I	J_1	P_E	P_D	J_2	D	C
SMV	1	2	3	4	5	6	0

expressa neste código corresponde à mostrada na equação abaixo.

$$\square\Diamond(a = TRUE) \wedge \square\Diamond[(a = TRUE) \wedge (b = 12)]$$

Exemplo

Com o intuito de exemplificar o processo de tradução de um modelo em asserções em LT para o formato SMV, é utilizado o exemplo descrito na seção 3.5. Nas equações A.1 e A.2 são reescritas as fórmulas que descrevem a condição inicial e dinâmica do sistema e do ambiente, respectivamente.

$$\begin{aligned}
&\varphi_i^{robo} : (robo = I)\\
&\varphi_t^{robo} : \begin{cases}
(robo = I) \rightarrow \bigcirc[(robo = I) \vee (robo = J_1)]\\
\bigwedge(robo = J_1) \rightarrow \bigcirc[(robo = I) \vee (robo = J_1) \vee (robo = P_E) \vee (robo = P_D)]\\
\bigwedge(robo = P_E) \rightarrow \bigcirc[(robo = J_1) \vee (robo = P_E) \vee (robo = J_2)]\\
\bigwedge(robo = P_D) \rightarrow \bigcirc[(robo = J_1) \vee (robo = P_D) \vee (robo = J_2)]\\
\bigwedge(robo = J_2) \rightarrow \bigcirc[(robo = P_E) \vee (robo = P_D) \vee (robo = J_2) \vee (robo = D)]\\
\bigwedge(robo = D) \rightarrow \bigcirc(robo = D)\\
\bigwedge[(robo = P_E) \wedge \bigcirc(adv = P_E)] \rightarrow \bigcirc(robo \neq J_2)\\
\bigwedge[(robo = P_D) \wedge \bigcirc(adv = P_D)] \rightarrow \bigcirc(robo \neq J_2)\\
\bigwedge[(robo = J_2) \wedge \bigcirc(adv = P_E)] \rightarrow \bigcirc(robo \neq P_E)\\
\bigwedge[(robo = J_2) \wedge \bigcirc(adv = P_D)] \rightarrow \bigcirc(robo \neq P_D)
\end{cases}
\end{aligned} \tag{A.1}$$

$$\begin{aligned}
&\varphi_i^{adv} : (adv = C)\\
&\varphi_t^{adv} : \begin{cases}
(adv = C) \rightarrow \bigcirc[(adv = C) \vee (adv = P_E) \vee (adv = P_D)]\\
\bigwedge(adv \neq C) \rightarrow [(\bigcirc adv) = adv]
\end{cases}
\end{aligned} \tag{A.2}$$

Uma vez que não é possível utilizar valores como P_E e D para representar o *status* do ambiente ou do sistema, pode ser feita uma equivalência com valores númericos. Na tabela 4 é apresentada esta equivalência.

Com isso, e dado que não há condições de justiça definidas para o

ambiente e que a definida para o sistema é $\square\lozenge(robo = D)$, é possível obter o modelo da planta como mostrado no código A.0.6.

No código A.0.6 é importante destacar a declaração da condição de justiça do ambiente. Como não são realizadas hipóteses sobre o ambiente, a condição é declarada como TRUE (linha 18).

```
MODULE main
  VAR
    e : env();
    s : sys(e.adv);

MODULE env()
  VAR
    adv : {0,3,4};

  ASSIGN
    init(adv) := 0;

  TRANS
    ( (adv=0) -> next((adv=0)|(adv=3)|(adv=4)) )&
    ( (adv!=0) -> (next(adv)=adv) );

 JUSTICE
    TRUE;

MODULE sys(adv)
  VAR
    robo : {1,2,3,4,5,6};

  ASSIGN
    init(robo) := 1;

  TRANS
    // Dinâmica do robô ////////////////////////////////////
    ( (robo=1) -> next((robo=1)|(robo=2)) ) &
    ( (robo=2) -> next((robo=1)|(robo=2)|(robo=3)|(robo=4)) ) &
    ( (robo=3) -> next((robo=2)|(robo=3)|(robo=5)) ) &
    ( (robo=4) -> next((robo=2)|(robo=4)|(robo=5)) ) &
    ( (robo=5) -> next((robo=3)|(robo=4)|(robo=5)|(robo=6)) ) &
    ( (robo=6) -> next(robo=6) ) &

    ( ((robo=3) & next(adv=3)) -> next(robo!=5) ) &
    ( ((robo=4) & next(adv=4)) -> next(robo!=5) ) &
    ( ((robo=5) & next(adv=3)) -> next(robo!=3) ) &
    ( ((robo=5) & next(adv=4)) -> next(robo!=4) );

  JUSTICE
    (robo=6);
```

Código A.0.6: Esquema da planta em SMV, referente ao exemplo mostrado na figura 6