JONATAS PAVEI

COORDENAÇÃO EM SISTEMAS MULTI-ROBÔS UTILIZANDO MÉTODOS BASEADOS EM AUTÔMATOS

FLORIANÓPOLIS
2011

UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Coordenação em Sistemas Multi-Robôs utilizando Métodos baseados em Autômatos

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia
de Automação e Sistemas.

JONATAS PAVEI

Florianópolis, Dezembro, 2011.

Coordenação em Sistemas Multi-Robôs utilizando Métodos baseados em Autômatos

Jonatas Pavei

'Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia de Automação e Sistemas, Área de Concentração em *Controle, Automação e Sistemas*, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina.'

Jean-Marie Farines, Dr.
Orientador

José Eduardo Ribeiro Cury, Dr.
Co-Orientador

José Eduardo Ribeiro Cury, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Banca Examinadora:

Jean-Marie Farines, Dr.
Orientador

José Eduardo Ribeiro Cury, Dr.
Co-Orientador

Antonio Eduardo Carrilho da Cunha, Dr.

Max Hering de Queiroz, Dr.

Ubirajara Franco Moreno, Dr.

Aos meus pais.

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer àquelas pessoas que fazem sentido a vida. Em especial, meu pai Tavico e minha mãe Luciane, que são responsáveis pelo que sou hoje. Meu filho Eduardo, que é minha maior motivação. Meus irmãos Guilherme e Maiara, companheiros de todas as horas. Thiele, minha noiva, cujo apoio e paciência foram fundamentais nos momentos mais difíceis.

Gostaria de agradecer em especial ao Professor Jean-Marie Farines pela orientação, colaboração, paciência e seus conhecimentos repassados desde a monografia. Sou muito grato pelas oportunidades que tive durante este período. Ao meu co-orientador Professor José Eduardo Ribeiro Cury, que contribuiu ativamente com este trabalho, agradeço também pela colaboração, paciência e seus conhecimentos repassados.

Aos meus colegas Larissa e Renan, pelos desenvolvimentos realizados no decorrer do trabalho.

Agradeço ainda ao CNPq pelo apoio financeiro nesses 24 meses de trabalho. O mesmo para o Departamento de Automação e Sistemas e todos os seus funcionários e colaboradores.

Por fim, aos meus colegas de mestrado agradeço o apoio, as discussões, os estudos em grupo e pelos ótimos momentos de descontração.

A todos, muito obrigado!

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia de Automação e Sistemas.

Coordenação em Sistemas Multi-Robôs utilizando Métodos baseados em Autômatos

Jonatas Pavei

Dezembro/2011

Orientador: Jean-Marie Farines, Dr.
Co-Orientador: Jose Eduardo Ribeiro Cury, Dr.
Área de Concentração: Controle, Automação e Sistemas
Linha de Pesquisa: Automação e Sistemas Mecatrônicos
Palavras-chave: Multi-Robôs, Controle Supervisório, Autômato-Jogo
Número de Páginas: xxxii + 143

O problema da coordenação de um sistema multi-robôs, que pode realizar atividades mais complexas e de forma mais rápida e eficiente que um único robô, é um tema complexo que apresenta um importante desafio de pesquisa. O planejamento de trajetórias e de tarefas são alguns dos problemas fundamentais dos sistemas multi-robôs e se referem a encontrar caminhos livres de colisão para cada robô determinado a realizar uma dada tarefa. Obviamente, para alcançar estes caminhos livres de colisão, algum tipo de coordenação é indispensável. Para garantir propriedades de segurança e alcançabilidade, este trabalho se interessa em estudar o uso de técnicas de modelagem e de verificação formal na solução deste tipo de problema. Para tal, duas diferentes técnicas para a solução do planejamento e coordenação de sistemas multi-robôs foram abordadas. Na primeira técnica, Teoria de Controle Supervisório, o problema consiste em calcular o supervisor ótimo que representa a lógica de controle utilizada para o planejamento de tarefas. Na segunda técnica, Autômato Jogo-Temporizado, uma estratégia é sintetizada para o planejamento de tarefas dos robôs que considera um jogo entre o ambiente e o controlador evidenciando a reatividade destes sistemas e a constante busca da vitória por parte

do controlador em termos de segurança e alcançabilidade. Por fim, uma arquitetura genérica para a implementação de sistemas multi-robôs foi desenvolvida para fornecer desde o planejamento de tarefas ao controle contínuo dos robôs. Desta maneira, este trabalho avalia diferentes métodos formais para o desenvolvimento do planejamento e coordenação de sistemas multi-robôs, além de reproduzir os resultados das técnicas em um ambiente real.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Automation and Systems Engineering.

Coordination in Multi-Robot Systems using Methods based on Automata

Jonatas Pavei

December/2011

Advisor: Jean-Marie Farines, Dr.
Co-advisor: José Eduardo Ribeiro Cury, Dr.
Area of Concentration: Control, Automation and Systems
Research Area: Automation and Mechatronics Systems
Key words: Multi-Robot Systems, Supervisory Control, and Timed Game Automata
Number of Pages: xxxii + 143

The coordination problem of a multi-robot system to accomplish complex activities and in a faster and more efficient way than a single robot is a complex issue that presents a major research challenge. The path and task planning are some of the fundamental problems of multi-robot systems and corresponds to find collision-free paths for each robot to accomplish an individual given task. Obviously, to achieve these collision-free paths, some kind of coordination is essential. For this purpose, two different techniques for solving the planning and the coordination of multi-robot systems have been tackled. In the first technique, the Supervisory Control Theory, the problem consists to calculate the optimal supervisor representing the control logic used for task planning. In the second technique, Timed Game Automata, a strategy is synthesized for the robot task planning that take account a game between the environment and the controller showing the reactivity of these systems and the constant seek for the victory by the controller in terms of safety and reachability. Finally, a generic architecture for the implementation of a multi-robot system is designed to provide resources from task planning to the continuous control. Thus,

this work evaluates different formal methods for the development of planning and coordination of multi-robots systems, besides to reproduce the results of the techniques in a real environment.

Sumário

Lista de Acrônimos

SMR	Sistemas Multi-Robôs
RAM	Robôs Autônomos Móveis
SED	Sistema a Eventos Discretos
CTL	*Computacional Tree Logic*
LTL	*Linear Temporal Logic*
TCS	Teoria de Controle Supervisório
AJT	Autômato-Jogo Temporizado
ADEF	Autômato Determinístico de Estados Finitos
ANDEF	Autômato Não-Determinístico de Estados Finitos
AT	Autômato Temporizado
TCTL	*Timed Computation Tree Logic*

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O estudo de sistemas multi-robôs naturalmente estende a pesquisa sobre robôs móveis unitários, em especial os robôs móveis autônomos (RMA) [45]. Um robô móvel autônomo é um sistema fisicamente independente, equipado com sensores e atuadores, suficiente para realizar uma dada atividade. Os sistemas multi-robôs são então compostos de robôs móveis autônomos que operam compartilhadamente em um ambiente realizando uma ou mais atividades específicas.

O estudo de sistemas multi-robôs estende-se para uma gama de tópicos, nos quais cada um foca em uma determinada funcionalidade do sistema. Dentre estes tópicos podemos citar, além dos tópicos relacionados à RMA [45], a comunicação entre robôs; a localização, o mapeamento e exploração; a manipulação e transporte de objetos; o planejamento de trajetórias e de tarefas; entre outros. Para este sistema como para os robôs individuais é indispensável atender requisitos de segurança. Uma forma literária destes se encontram nas três leis da robótica, elaboradas pelo escritor Isaac Asimov [6].

1.1 Motivação

Tradicionalmente os SMR possuem diversas aplicações, desde as militares à aeroportos e as áreas de manufatura. Entretanto, novas aplicações que potencialmente podem ser beneficiadas por SMR estão emergindo, entre elas estão as aplicações espaciais, os resgates em ambientes perigosos, etc. Em todas essas áreas, os sistemas multi-robôs podem lidar com problemas que são difíceis para serem executados por apenas um robô ou podem resolvê-los com uma maior eficiência tanto na execução das tarefas como na robustez e confiabilidade do sistema.

Quanto à robustez, duas características são de extrema importância, a adaptabilidade e a tolerância a falhas. Adaptabilidade se refere à capacidade do sistema multi-robôs de mudar seu comportamento devido às mudanças do ambiente dinâmico ou de objetivo a fim de garantir a execução da tarefa desejada. Já a tolerância a falhas se refere a habilidade do SMR em lidar com problemas individuais de cada robô, tais como problemas de comunicação ou problemas estruturais que podem ocorrer durante a execução de uma tarefa [27].

As consequências dos problemas causados com a falta de adaptabilidade e de tolerância a falhas podem acarretar diversos tipos de danos, tais como financeiros e ambientais. Além disso, uma vez que estes robôs compartilham o espaço físico com pessoas de uma maneira autônoma, torna-se crítico a necessidade de se evitar colisões. Assim, a adaptabilidade do sistema é indispensável. Neste sentido, a garantia de segurança e confiabilidade na execução das tarefas tem sido de muito interesse nas pesquisas sobre o planejamento e coordenação de sistemas multi-robôs. Desta maneira, pode-se afirmar que planejamento e coordenação são de fundamental importância para um sistema SMR que é composto de vários robôs móveis interagindo entre si.

Os planejamentos de trajetórias e de tarefas são alguns dos problemas fundamentais dos SMR e se referem a atender os objetivos fixados na atividade a realizar e a encontrar caminhos livres de colisão

para cada robô determinado a realizar uma dada atividade. Obviamente, para alcançar estes caminhos livres de colisão, algum tipo de coordenação é indispensável. No ponto de vista da engenharia, os SMR são sistemas inerentemente complexos. O controle destes sistemas complexos apresentam desafios que vão além das abordagens de controle teórico clássico. Um destes desafios é a necessidade de controlar, coordenar e sincronizar a operação de vários robôs com segurança e confiabilidade para realizarem uma dada atividade. Desta forma, faz-se necessário a utilização de novos métodos e estratégias que permitam especificar o comportamento do sistema desejado de uma maneira formal e concisa.

Tradicionalmente, as técnicas de planejamento de trajetória são concentradas na geração de controladores que levam em conta a dinâmica contínua de robôs enquanto por sua vez, o planejamento das tarefas é mais focado na determinação das sequências de ações discretas. Nesta dissertação, o problema de planejamento será abordado apenas do ponto de vista das tarefas, onde o controle, a coordenação e a sincronização serão determinados em nível discreto.

Métodos formais como autômatos temporizados, autômatos híbridos e redes de Petri têm sido utilizados na literatura para representar o comportamento de sistemas com vários robôs cooperando em um ambiente único [5, 14, 49]. Além disto, fórmulas de lógica temporal representando as propriedades desejadas deste tipo de sistemas têm servido para a verificação através de métodos de *model-checking*. Neste sentido, a utilização da lógica temporal na definição desses sistemas pode ser usada para realizar a síntese de controladores, indicando um poderoso método para a garantia de segurança e confiabilidade em sistemas multi-robôs.

1.2 Objetivos

O objetivo principal desta dissertação é realizar um estudo sobre o projeto de coordenadores (controladores) através de duas abordagens diferentes, a Teoria de Controle Supervisório (TCS) e o Autômato-Jogo Temporizado (AJT). Em ambas as abordagens, a síntese dos controladores é analisada em uma topologia centralizada e em uma topologia distribuída.

O foco do estudo é em relação à coordenação de sistemas multi-robôs, principalmente no que se refere à segurança e à confiabilidade. Desta maneira, através da combinação das diferentes abordagens com as diferentes topologias, pode-se traçar as potencialidades e as restrições que tais métodos para síntese de controladores possam possuir.

Por fim, objetiva-se, ao fim deste trabalho, desenvolver uma arquitetura para a implementação de sistemas multi-robôs que suporte os controladores discretos aqui desenvolvidos.

1.3 Resultados do Trabalho

Os resultados deste trabalho focaram principalmente no planejamento de tarefas e coordenação de sistemas multi-robôs [34]. Entretanto, eles se estendem aos níveis de implementação e estudo das diferentes topologias destes tipos de sistema. Essas contribuições são discriminadas conforme apresentado abaixo.

- O planejamento de tarefas e coordenação para sistemas multi-robôs são desenvolvidos considerando a dinamicidade do ambiente, i.e., obstáculos móveis durante a execução das tarefas dos robôs podem interferir no sistema. Esta consideração torna essencial a necessidade de desenvolver um plano de tarefas que admita a reatividade dos sistemas em que estão inseridos. Um grande

problema nestes tipos de sistema é a necessidade de replanejamento do plano de tarefas de forma *online*. Um dos resultados deste trabalho foi garantir que o plano de tarefas já incorpore informações que permitam tratar a dinâmica do ambiente evitando, assim, o indesejável replanejamento.

- A síntese dos controladores discretos que representam o plano de tarefas para o sistema multi-robôs foi realizada, nesta dissertação, utilizando métodos formais. Em especial, duas metodologias foram tratadas, a Teoria de Controle Supervisório (TCS) e o Autômato-Jogo Temporizado (AJT). Um dos resultados deste trabalho foi a comparação e análise das sínteses dos controladores entre os diferentes métodos e topologias aqui tratadas.

- O sistema multi-robôs foi tratado para duas diferentes topologias: uma considerando um controlador central e outra, controladores distribuídos (embarcados nos robôs). Este trabalho permitiu identificar as características de ambas estruturas, compará-las e evidenciar as vantagens e desvantagens de cada topologia.

- Uma arquitetura genérica para a implementação de sistemas multi-robôs foi desenvolvida com objetivo de fornecer desde o planejamento de tarefas ao controle de trajetória dos robôs. Esta arquitetura é capaz de suportar diferentes metodologias para o desenvolvimento do planejamento de tarefas e coordenação de sistemas multi-robôs. Além disso, existe a possibilidade de se acrescentar novas trajetórias aos robôs quando desejado. Um dos resultados foi a definição e teste de um arcabouço para a implementação de sistemas multi-robôs.

1.4 Estrutura da Dissertação

Os capítulos posteriores estão organizados conforme apresentado abaixo.

Capítulo 2 compreende a definição dos sistemas multi-robôs destacando mais precisamente o problema de planejamento e coordenação. Também nesse capítulo, a caracterização do problema a ser tratado e os trabalhos relacionados a este trabalho são apresentados.

Capítulo 3 apresenta uma revisão dos tópicos relacionados aos sistemas a eventos discretos;

Capítulo 4 apresenta primeiramente a teoria de controle supervisório em duas abordagens diferentes: Monolítica e Modular Local. Após, o projeto de um coordenador de sistemas multi-robôs é realizado para cada abordagem de controle supervisório na configuração centralizada e distribuída; posteriormente será apresentada uma análise dos resultados obtidos em todos os casos.

Capítulo 5 introduz inicialmente o conceito do autômato-jogo temporizado e a síntese de controladores utilizando esse autômato. O projeto de um coordenador de sistemas multi-robôs é apresentado a seguir na configuração centralizada e distribuída; os resultados serão apresentados e discutidos posteriormente.

Capítulo 6 apresenta a arquitetura definida para sistemas robóticos para o caso centralizado e distribuído. Também é apresentada a implementação dos módulos da arquitetura assim como a definição das tecnologias utilizadas;

Capítulo 7 apresenta as conclusões e perspectivas do trabalho aqui desenvolvido.

Capítulo 2

Planejamento e Coordenação em Sistemas Multi-Robôs

Neste capítulo são apresentados os principais problemas e definições para o planejamento e coordenação de sistemas multi-robôs (SMR). Neste contexto, são abordados aspectos tais como planejamento de trajetória e de tarefas, e a coordenação destes sistemas em ambientes dinâmicos. Também são descritos a caracterização do problema e o estudo de caso a ser tratado nesta dissertação. Por fim, alguns trabalhos já realizados na área de planejamento e coordenação de SMR serão apresentados.

2.1 Sistemas Multi-Robôs

Em termos da aplicação dos SMR, uma especial atenção tem sido dada em atividades onde o ambiente em que estes sistemas estão inse-

ridos é dinâmico, sendo motivo de muitas pesquisas na área de robótica [1]. Nesta situação, geralmente as mudanças do ambiente são inesperadas e desconhecidas dos robôs presentes neste [27]. Portanto, um dos problemas fundamentais para os sistemas multi-robôs é a necessidade de planejar e controlar as tarefas e as trajetórias dos robôs tal que eles não colidam entre si nem com os eventuais obstáculos. Para tal, é imprescindível a necessidade de algum tipo de coordenação.

Em Iocchi *et al.* (2001) [27], uma taxonomia para a classificação de sistemas de coordenação de SMR foi proposta. A taxonomia é divida em 4 dimensões (Tabela 2.1), representando as diferentes características da coordenação de sistemas multi-robôs.

Dimensões de Coordenação
Cooperação
Conhecimento
Orquestração
Organização

Tabela 2.1: Dimensões de coordenação para SMR.

O primeiro nível da taxonomia, a cooperação, é referente à habilidade do sistema em cooperar entre si a fim de atingir um objetivo específico. No caso dos sistemas multi-robôs, os robôs são *cooperativos*, interagem entre si como um time a fim de buscar um objetivo global; ou são *não-cooperativos*, possuem diferentes objetivos dentro do mesmo sistema, portanto agem independentes um dos outros.

O segundo nível, o conhecimento, trata do conhecimento que cada robô tem da presença e da ação dos outros robôs. Este nível é divido em *ciente*, no qual os robôs têm conhecimento a respeito dos outros robôs, i.e do estado do sistema, ou *não-ciente*, caso contrário.

No terceiro nível, a orquestração, leva-se em consideração o mecanismo que é utilizado para realizar a cooperação do sistema, e desta maneira, definir se a orquestração é *forte*, *fraca* ou *não-existe*. A or-

questração existe quando as ações realizadas por cada robô levam em consideração as ações executadas pelos outros robôs. Assim, a diferença entre a orquestração forte e fraca está na existência ou não de um protocolo pré-definido, ou seja, se existem regras que estabelecem a interação entre robôs. Caso não exista nenhuma interação entre eles, a orquestração é considerada *não-existente*.

Por fim, o quarto nível, a organização, lida com a maneira com a qual o sistema de decisão do SMR é implementado. Ele pode ser dividido em *centralizado*, no qual o sistema possui um único agente líder (robô, PC, ...) responsável pela organização e controle de todos os robôs, ou *distribuído*, onde o sistema é organizado com robôs que possuem a implementação de seu próprio controle levando em consideração a coordenação com outros robôs.

Dentro desta taxonomia, diversos são os desafios quando se trata de sistemas multi-robôs. Em cada dimensão e dependendo da topologia do sistema, várias possibilidades de implementação podem ser realizadas. Neste trabalho, o sistema multi-robôs é tratado em duas topologias diferentes de acordo com a taxonomia de Iocchi *et al.* (2001) [27], alterando-se apenas na dimensão Organização. Nas outras dimensões as características do SMR são consideradas as mesmas: *Cooperativo*, *Ciente* e *Forte*.

A Figura 2.1 expõe a estrutura hierárquica para as dimensões de coordenação da taxonomia apresentadas em Iocchi *et al.* (2001) [27]. Os quadrados em branco representam as características das dimensões de coordenação utilizadas nesta dissertação, ressaltando que as duas topologias a serem abordadas diferem no nível da Organização: *Centralizado* e *Distribuído*.

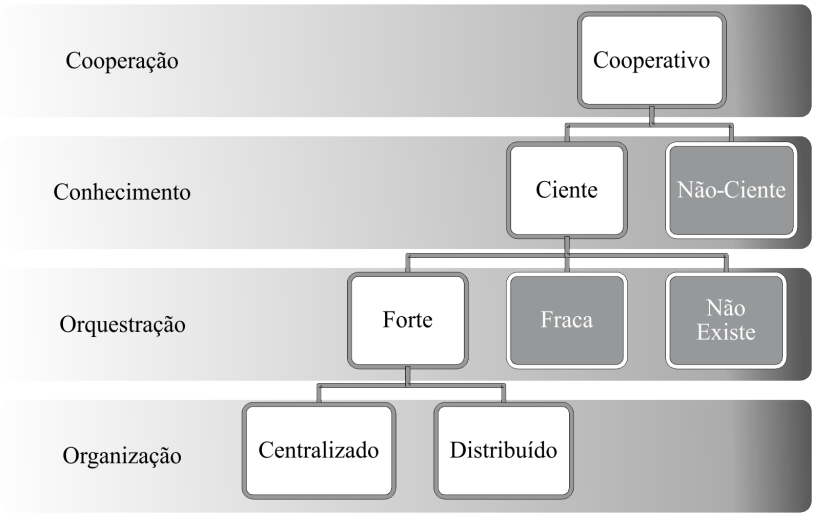


Figura 2.1: Dimensões de coordenação em Sistemas Multi-Robôs.

2.2 Planejamento e Coordenação

O planejamento de trajetória e o planejamento das tarefas são dois problemas fundamentais para a concepção de sistemas autônomos móveis e, mais particularmente, para o caso dos robôs. Tradicionalmente, as técnicas de planejamento de trajetória são concentradas na geração de controladores que levam em conta a dinâmica contínua de robôs enquanto por sua vez, o planejamento das tarefas é mais focado na determinação das sequências de ações discretas. Entretanto, trabalhos mais recentes [14, 21] implementam, numa perspectiva de sistemas híbridos, a integração de ações discretas de alto nível com controladores contínuos de baixo nível, permitindo uma abordagem unificada com o mesmo *framework*.

No âmbito deste trabalho, o problema de planejamento será abordado apenas do ponto de vista da coordenação das tarefas e, portanto, o sistema a ser tratado é visto como um Sistema a Eventos Discretos (SEDs) [11, 48]. Por sua vez, um controlador final híbrido poderá

ser construído a partir de controladores contínuos simples, compostos em acordo com a execução do autômato que representa o controlador discreto.

Como já comentado na seção anterior, o problema de planejamento tem sido focado em garantir a alcançabilidade dos objetivos do sistema, evitando colisões entre robôs e com obstáculos do ambiente, sendo este estático ou dinâmico. O planejamento quando o ambiente é estático considera todas as informações dos obstáculos no momento da síntese, não podendo ser alteradas na execução de uma tarefa. Porém, quando se trata de um ambiente dinâmico, o planejamento não tem conhecimento *a priori* sobre a situação dos obstáculos móveis, o que muitas vezes envolvem um replanejamento em tempo-real, o que pode dificultar sua implementação. Com isto, busca-se cada vez mais a geração *offline* de planos que levam em conta a dinâmica do ambiente.

2.3 Caracterização do Problema

Para as topologias *Centralizada* e *Distribuída*, assume-se um SMR onde o espaço de trabalho é *indoor* e dividido por células idênticas nas quais os robôs poderão se locomover, como ilustrado pela Figura 2.2. Estas células servirão para caracterizar posições iniciais, posições a serem atingidas e eventuais posições de obstáculos fixos e móveis. Os obstáculos fixos são restrições permanentes do espaço de trabalho que os robôs sempre deverão respeitar, enquanto os obstáculos móveis, por exemplo representados por portas, poderão ou não obstruir a passagem. Os robôs são considerados idênticos e realizam atividades diferentes dentro do mesmo ambiente, embora haja a necessidade de cooperação. Eles têm dois tipos de sensores: um para identificar os obstáculos móveis e outro para detetar a passagem entre as células. Além disso, cada robô tem atuadores, responsáveis pela locomoção e uma interface de comunicação, pela qual é realizada a troca de informações com outros robôs e/ou com um computador central, depen-

dendo da topologia a ser implementada. Assume-se que o problema de controle de trajetória dos robôs no trânsito de uma célula a qualquer de suas células vizinhas é resolvido, por um movimento uniforme seguindo uma trajetória linear passando pela parte central de cada célula, conforme ilustrado pelas linhas tracejadas na Figura 2.2. Portanto, o foco principal será no planejamento de tarefas e na coordenação de SMR.

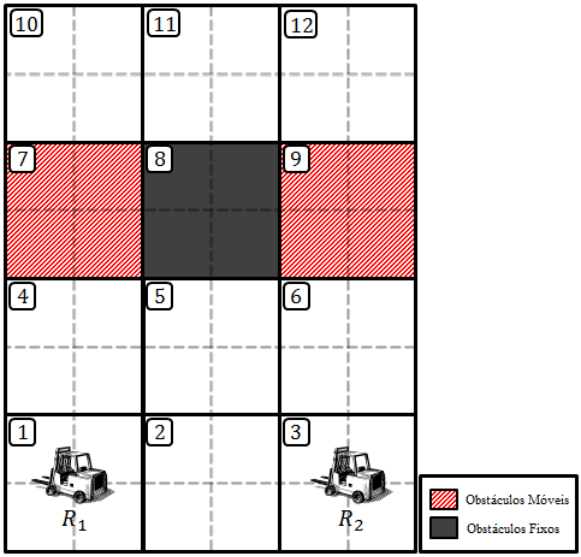


Figura 2.2: Espaço de trabalho para um Sistema Multi-Robôs.

2.3.1 Estudo de Caso

Considera-se como estudo de caso um espaço de trabalho com 12 células onde os robôs possam se locomover, saindo de suas posições iniciais e seguindo até seus objetivos caracterizados por outras posições no espaço de trabalho. Neste ambiente, existe um obstáculo fixo, na posição 8, impedindo a passagem dos robôs. Nas posições 7 e 9 estão os obstáculos móveis, i.e. portas que podem ou não obstruir a passagem do robô. O espaço de trabalho é representado através das células da Figura

2.2, assumidas quadradas e de mesmo tamanho. As áreas hachuradas, representam as portas e as áreas escuras, os obstáculos fixos. Os robôs, definidos como R_1 e R_2 se encontram inicialmente nas posições 1 e 3 do espaço de trabalho, respectivamente. O robô R_1 possui como objetivo atingir a posição 12 do espaço de trabalho enquanto o robô R_2 a posição 10. Portanto, objetiva-se encontrar um plano de tarefas para este caso que garanta o objetivo do sistema multi-robôs, evitando a colisão entre eles e com os obstáculos fixos e móveis.

A caracterização das topologias a serem utilizadas nesta dissertação é feita da seguinte maneira:

- Nível Cooperação (*Cooperativo*): O sistema multi-robôs tem como objetivo a alcançabilidade das posições 12 e 10 pelos robôs R_1 e R_2, respectivamente. Logo, é necessário uma cooperação entre os robôs dentro do espaço de trabalho para que nenhum robô impossibilite o objetivo do outro robô.

- Nível Conhecimento (*Ciente*): Cada robô necessita ter o conhecimento da presença e das ações dos outros robôs presentes no espaço de trabalho. É a partir dessas informações que o controlador discreto poderá garantir a segurança de evitar a colisão entre robôs;

- Nível Orquestração (*Forte*): As ações realizadas por cada robô devem levar em consideração as ações executadas por outros robôs presentes no espaço de trabalho. Desta maneira, pode-se obter uma trajetória que evite a colisão entre robôs e procure o melhor caminho para atingir seu objetivo. Esta interação é estabelecida através de regras que definem como é feita a troca de informações entre os robôs;

- Nível Organização (*Centralizado* e *Distribuído*): Em relação à organização, ambos os casos serão considerados. Quando o controlador for centralizado, as informações serão trocadas entre os

robôs e um computador central, que receberá informações sobre a posição dos robôs e os informará de suas próximas ações. Quando o controlador for distribuído, cada robô será autônomo em suas ações e as informações serão trocadas apenas entre robôs.

2.4 Trabalhos Relacionados

Desde a década de 90, métodos formais para o controle e coordenação de sistemas multi-robôs vêm sido abordados, principalmente no contexto de planejamento de tarefas. A metodologia utilizada por estes métodos formais utiliza uma variedade de análises matemáticas tais como a síntese de controladores e a verificação de modelos (*model checking*).

O projeto de um controlador discreto para um robô móvel pode ser abordado de várias maneiras, seja a partir de síntese de controladores segundo a Teoria de Controle Supervisório (TCS) proposta por Ramadge and Wonham (1989) [43] e algumas extensões [18, 38], onde se obtém uma solução a mais permissiva possível, seja a partir de uma abordagem através de verificação formal de modelos (*model checking*), na qual se apresentam duas diferentes abordagens: a geração de um contra-exemplo obtido a partir da verificação formal do modelo [5, 42] e a síntese a partir de um Autômato-Jogo Temporizado [12, 13, 32, 36].

A verificação formal tem sido utilizada para garantir que os modelos de alto-nível dos sistemas multi-robôs atendam as especificações exigidas de acordo com uma formulação seguindo uma lógica temporal. A lógica temporal é um ramo especial da lógica centrado em proposições cujo valor depende da ordenação temporal, sendo muito utilizada na verificação formal de propriedades do comportamento dos sistemas [23]. Várias são as ferramentas disponíveis para este tipo de abordagem, tais como NuSMV, KRONOS, HyTECH e UPPAAL. Nestes exemplos, os modelos de comportamento podem ser representados por autômatos

temporizados [3] e redes de petri [33].

Um dos primeiros trabalhos utilizando métodos formais para o planejamento e coordenação de SMR foi a abordagem proposta por Alur *et al.* (1999) [4]. Nesta abordagem, foram utilizados autômatos híbridos para a modelagem formal e projeto da estratégia de controle do sistema multi-robôs. A ferramenta de verificação HyTech foi utilizada para a análise dos parâmetros no modelo e para o fornecimento deste como *feedback* para o sistema. Porém, o problema mostrou-se muito complexo devido ao uso combinado do planejamento de trajetórias e de tarefas no mesmo modelo. Em Andersen *et al.* (2004) e Quottrup *et al.* (2004) [5, 42], a verificação formal de modelos também é utilizada, porém apenas para o planejamento de tarefas. Nestes casos, o método proposto faz o uso de uma infra-estrutura de robôs com controles em malha-fechada obedecendo restrições sobre o ambiente e outros robôs. O ambiente é considerado estático, porém, quando tratado como dinâmico, um replanejamento *online* seria necessário. O modelo de comportamento é definido por autômato temporizado e implementado na ferramenta UPPAAL. Através desta ferramenta, a verificação formal das propriedades exigidas para o controle e coordenação do SMR é realizada a partir de fórmulas de lógica temporal *Computacional Tree Logic* (CTL). Desta forma, é possível obter o plano de tarefas necessário para o sistema atingir seus objetivos e atender suas restrições, tais como evitar obstáculos.

Uma outra abordagem para a utilização de verificação formal é uso de lógica temporal para sintetizar o controle discreto algoritmicamente [12, 13, 32, 35, 36] através da satisfatibilidade das expressões lógicas que compõem a especificação de planejamento e coordenação das tarefas.

Em Faneikos *et al.* (2005) [21], o problema de planejamento e coordenação de SMR é utilizado a fim de satisfazer fórmulas expressas em *Linear Temporal Logic* (LTL). Abstrações discretas da movimentação dos robôs baseadas na decomposição do ambiente são consideradas e

definidas na lógica temporal linear LTL. Assim, planos discretos são gerados a partir destas expressões utilizando a ferramenta NuSMV [15]. Por fim, o plano discreto é traduzido em trajetórias contínuas usando controle híbrido. Recentemente, extensões deste trabalho [29, 30] vêm ampliando os objetivos do planejamento, que tradicionalmente é a alcançabilidade, incluindo emergência, cobertura e impedimento de colisão. Além disso, a tratamento da dinamicidade dos obstáculos do ambiente é incorporada.

Em outros trabalhos mais recentes [17, 49], o formalismo de Redes de Petri é incluído para a realização de um novo arcabouço para a representação de SMR, que permite a descrever planos de tarefas em ambientes dinâmicos. Estes trabalhos promovem uma forma prática e intuitiva de modelar as tarefas dos robôs. Porém, o plano de tarefas obtido não é resultado de uma síntese ou de uma verificação formal. Desta maneira, a satisfabilidade das propriedades é somente garantida por análise do plano de tarefas após ele ter sido executado.

Como não existe um método universal para o planejamento e coordenação de sistemas multi-robôs, tampouco existe um algoritmo único para tal, propostas de gerenciamento dos planos [26, 28] têm sido desenvolvidas através de arquiteturas que utilizam um plano de gerenciamento. Estas propostas procuram criar um framework onde todos os aspectos envolvidos em um SMR sejam tratados.

2.5 Conclusão

Em relação às características do problema a ser tratado, objetiva-se a síntese de um controlador discreto para o SMR visando a coordenação dos robôs em um plano de tarefas, considerando que o ambiente no qual evolui é dinâmico. Os casos de organização a serem estudados serão o centralizado e o distribuído. Para tal, duas abordagens são utilizadas nesta dissertação: a Teoria de Controle Supervisório [11, 18, 47]

juntamente com sua extensão modular [38, 39, 40, 41] (Capítulo 4) e o Autômato-Jogo Temporizado (AJT) [32, 36] (Capítulo 5).

Capítulo 3

Modelos para Sistemas a Eventos Discretos

A modernização da tecnologia e o consequente aumento da complexidade de sistemas automatizados faz com que cada vez mais seja necessária a utilização de formalismos para a coordenação e o controle destes sistemas. Entre as suas possíveis aplicações, podemos citar: automação da manufatura, a robótica, a supervisão de tráfego, a logística (canalização e armazenamento de produtos, organização e prestação de serviços), os sistemas operacionais, as redes de comunicação de computadores, a concepção de software, o gerenciamento de bases de dados e a otimização de processos distribuídos.

Estes sistemas compartilham a forma com a qual evoluem, dependendo de um estímulo realizado sobre ou pelo ambiente, percebendo assim, suas reações. Estes estímulos são denominados eventos, que são exemplos de ativação e término de uma tarefa e a percepção da mudança de estado em um sensor. Portanto, as características destes sistemas, que possuem um caráter discreto devido à ocorrência de eventos, levam a denominá-los como Sistemas a Eventos Discretos (SED), em oposição

aos sistemas de variáveis contínuas, tratados pela Teoria de Controle Clássica.

Neste capítulo é inicialmente definida a classe de sistemas abordada neste trabalho (SEDs) com suas principais características. Após, são apresentados os principais formalismos para a representação de SEDs, conforme [11, 18] e suas extensões para o caso temporizado.

3.1 Sistemas a Eventos Discretos

O conceito de sistemas possui diversas definições representativas, o que leva a concluir que um sistema nem sempre pode ser associado com objetos físicos e leis da natureza mas também com atividades vindas de ações humanas. De modo geral, um sistema é uma parte limitada do Universo que interage com o mundo externo através das fronteiras que o delimitam [18]. No presente trabalho, os sistemas utilizados realizam esta interação através de eventos que podem alterar o estado do sistema ou ser uma ação sobre o ambiente, conforme a definição 3.1.

Definição 3.1. *[18] Um sistema a eventos discretos (SED) é um sistema dinâmico que evolui de acordo com a ocorrência abrupta de eventos físicos, em intervalos de tempo em geral irregulares e desconhecidos.*

Em particular, um SED é discreto no tempo e no espaço de estados; é dirigido a evento: isto é, evolui por um evento; e pode ser não-determinístico: capaz de escolher as transições internamente ou por algum motivo externo [11, 48].

Um evento, definido por e, pode representar uma ação específica, uma ocorrência espontânea ou o resultado de várias condições que são verificadas. Desta maneira, podemos afirmar que os eventos podem ser controláveis (i.e., ação de um controlador) ou não-controláveis (i.e., uma reação do ambiente). O conjunto de eventos, é portanto, denotado

por $\Sigma = \Sigma_c \cup \Sigma_u$, sendo Σ_c o conjunto dos eventos controláveis e Σ_u o conjunto dos eventos não-controláveis.

Existem diversos tipos de modelos que são utilizados para modelar SED, tais como Redes de Petri, Teoria das Filas, Teoria de Linguagens e Autômatos, entre outros. Porém, nesta dissertação é utilizado o principio da Teoria de Linguagens e Autômatos para o modelagem do sistema. O comportamento lógico de um SED pode ser então modelado a partir de linguagens, que permitem descrever um modelo comportamental do sistema, pois se baseia na descrição de uma sequência de eventos. A palavra linguagem vem do fato que se pode pensar em um conjunto de eventos como um *alfabeto* e sequências de eventos como *palavras* (*cadeias*). O conteúdo aqui apresentado é fortemente inspirado em Cury (2001) [18] e Cassandras e Lafortune (2006) [11].

3.2 Linguagens

O conjunto de todas as possíveis cadeias finitas compostas com elementos de Σ é denotado por Σ^*, incluindo a cadeia vazia, denotada por ϵ. De acordo com a definição 3.2, uma linguagem é sempre um subconjunto de Σ^*. Em particular, $\emptyset, \Sigma, \Sigma^*$ são linguagens [18]. Como exemplo, considera-se um alfabeto $\Sigma = \{a, b, c\}$. Temos como possíveis linguagens:

- $L_1 = \{\epsilon, a, abb\}$;
- $L_2 = \{aa, ab, ac\}$.

Definição 3.2. *Uma linguagem definida sobre um conjunto de eventos Σ é um conjunto de cadeias de tamanho finito a partir dos eventos em Σ. [11]*

O conjunto de operações habituais, tais como união, intersecção, diferença e complemento com respeito à Σ^* são aplicáveis às lingua-

gens desde que estas sejam também conjuntos. Outras operações serão apresentadas em seguida [11, 18, 48].

Algumas terminologias sobre cadeias são necessárias para a definição destas operações. Se $tuv = s$ com $t, u, v \in \Sigma^*$, então:

- t é chamado prefixo de s;
- u é chamado subcadeia de s;
- v é chamado sufixo de s.

1. Concatenação: Sejam $L_a, L_b \subseteq \Sigma^*$, então

$$L_a L_b := \{s \in \Sigma^* : (s = s_a s_b)\ e\ (s_a \in L_a)\ e\ (s_b \in L_b)\}$$

Logo, uma cadeia que está em $L_a L_b$ pode ser escrita como a concatenação de uma cadeia de L_a com uma cadeia de L_b;

2. Prefixo-Fechamento: Seja uma linguagem $L \subseteq \Sigma^*$, então

$$\overline{L} := \{s \in \Sigma^* \ : (\exists t \in \Sigma^*)[st \in L]\}$$

Logo, $\overline{L}$ é composto de todas as cadeias de Σ^* que são prefixos de L. Em geral, $L \subseteq \overline{L}$. L é um Prefixo-Fechamento se $L = \overline{L}$. Desta maneira, L é dita prefixo-fechada se qualquer prefixo de qualquer cadeia de L também é uma cadeia de L;

3. Fechamento-Kleene: Seja uma linguagem $L \subseteq \Sigma^*$, então

$$L^* := \{\epsilon\} \cup L \cup LL \cup LLL \cup ...$$

Uma cadeia de L^* é formada pela concatenação de um número finito de cadeias de L, incluindo a cadeia vazia ϵ.

3.2.1 Representação de SEDs por linguagens

Uma linguagem pode ser vista como uma maneira formal de descrever o comportamento de um SED. Ela especifica todas as sequências admissíveis de eventos que o SED é capaz de *atuar* ou *perceber*, enquanto evita a necessidade de qualquer estrutura adicional [11]. Este comportamento sequencial dos SEDs pode ser descrito através de um par de linguagens $\mathcal{D} = (L, L_m)$ sobre um alfabeto Σ.

Os elementos do modelo $\mathcal{D}$ são definidos como: $L \subseteq \Sigma^*$, definindo o comportamento gerado pelo sistema, ou seja, o conjunto de todas as cadeias de eventos fisicamente possíveis de ocorrência no sistema. Por outro lado, $L_m \subseteq L$ representa o comportamento marcado do sistema, ou seja, o conjunto de cadeias em L que correspondem as tarefas completas que o sistema pode realizar.

Como estamos tratando de modelos sequenciais, para que um SED possa produzir uma cadeia qualquer z, é necessário que o mesmo tenha produzido anteriormente todos os seus prefixos. Sendo assim, o comportamento gerado pode ser representado por uma linguagem prefixo-fechada, conforme descrito na seção anterior. As propriedades relacionadas às observações acima são formalmente descritas como:

1. $L_m \subset L$, ou seja, o comportamento gerado contém o comportamento marcado de um SED;
2. $L = \overline{L}$, ou seja, o comportamento gerado de um SED é prefixo-fechado.

Conforme apresentado acima, podemos então modelar formalmente o comportamento de um SED utilizando uma linguagem. Ela pode especificar todas as sequências de eventos possíveis (L) ou o conjunto de tarefas que o sistema pode completar (L_m). Porém, a representação de uma linguagem nos moldes apresentados nesta seção não é prática. Por este motivo, um método mais claro, conciso, prático e

sem ambiguidade é necessário para modelar estes sistemas que podem apresentar alto grau de complexidade.

3.2.2 Expressões Regulares

Uma expressão regular para um dado alfabeto Σ é representada da seguinte maneira:

1. **(a)** $\emptyset$ é uma expressão regular que representa a linguagem vazia,
   (b) ε é uma expressão regular denotando a linguagem $\{\varepsilon\}$,
   (c) σ é uma expressão regular denotando $\{\sigma\} \forall e \in \Sigma$;
2. Se r e s são expressões regulares então $rs, r^*, s^*, (r + s)$ são expressões regulares;
3. Toda expressão regular é obtida pela aplicação das regras 1 e 2 um número finito de vezes.

Expressões regulares fornecem um meio de descrição de linguagens, que segundo a definição seguinte, é definida como uma linguagem regular.

Definição 3.3. *Linguagem regular é qualquer linguagem que possa ser descrita por uma expressão regular.*

3.3 Autômatos

Um autômato é um dispositivo que é capaz de representar uma linguagem de acordo com regras bem definidas [11]. Nesta seção, é apresentado uma definição formal de um autômato.

3.3.1 Autômatos Determinísticos de Estados Finitos

Um autômato determinístico de estado finito (ADEF), denominado G, é definido pela quíntupla

$$G = (X, \Sigma, f, x_0, X_m)$$

Onde:

- X é o conjunto finito de estados do autômato;
- Σ é o conjunto de eventos;
- $f : X \times \Sigma \to X$ é uma função de transição, em geral, parcial. Em outras palavras, existe uma transição nomeada pelo evento e do estado x para o estado y, não sendo necessário desta forma, ser definida para todo elemento de Σ em cada estado de X;
- x_0 é o estado inicial;
- X_m é o conjunto de estados marcados, $X_m \subseteq X$. [18]

O autômato é dito determinístico pelo motivo de f ser uma função de $X \times \Sigma$ para X, isto é, não podem existir duas transições nomeadas com o mesmo evento a partir de um estado. Contrariamente, a estrutura de um autômato não-determinístico de estados finitos (ANDEF) é definida por meio de uma função de $X \times \Sigma$ para 2^X; neste caso, é possível múltiplas transições nomeadas com o mesmo evento a partir de um estado [11]. Neste trabalho, utiliza-se como padrão o ADEF.

A representação de um autômato pode ser feita de forma gráfica a partir de um grafo dirigido, onde os nós representam os estados e os arcos etiquetados pelos eventos as transições entre os estados. O estado inicial é representado por um seta no estado em questão e os estados marcados por dois círculos concêntricos.

Exemplo 3.1. *Neste exemplo, apresenta-se um ADEF conforme a figura 3.1, cuja descrição formal é relatada da seguinte maneira:*

- $X = \{1, 2, 3\}$;
- $\Sigma = \{a, b, c\}$;
- $f : X \times \Sigma \to X : \ f(1, a) = 2, \ f(2, a) = 2, \ f(2, b) = 3, \ f(3, a) = 2, \ f(3, c) = 1$;
- $x_0 = 1$;
- $X_m = 2$.

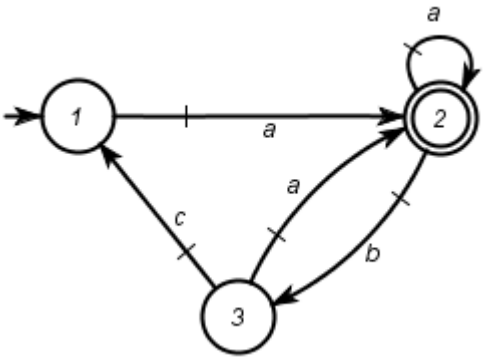


Figura 3.1: Autômato Determinístico de Estados Finitos (ADEF).

3.3.2 Representação por Autômatos

3.3.2.1 Definição Geral

A conexão entre as linguagens e os autômatos é feita inspecionando o diagrama de transição de estado de um autômato. Considerando todos os caminhos no diagrama de estados a partir de estado

inicial e os que, entre eles, chegam aos estados marcados, temos a definição 3.4. Desta forma, podemos também representar um SED por um autômato a partir das definições a seguir.

Definição 3.4. *A linguagem gerada por* $G = (X, \Sigma, f, x_0, X_m)$ *é*

$$\mathcal{L}(G) := \{s \in \Sigma^* : f(x_0, s) \text{ é definida}\}.$$

A linguagem marcada por G *é*

$$\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) : f(x_0, s) \in X_m\}.$$

Desta maneira, a linguagem $\mathcal{L}(G)$ representa todas as cadeias que podem ser exploradas no autômato a partir de um estado inicial, enquanto a linguagem $\mathcal{L}_m(G)$ conta com todas as cadeias, que a partir do estado inicial, chegam a um estado marcado. Portanto, para um SED, $\mathcal{L}(G)$ é o comportamento gerado pelo sistema e $\mathcal{L}_m(G)$ é o comportamento marcado ou conjunto de tarefas completas do sistema.

3.3.2.2 Autômatos Não-Bloqueantes

Um conceito importante utilizado em SEDs se refere ao fato de um autômato ser ou não bloqueante. Isto implica em um possível *deadlock* ou *livelock*, geralmente indesejáveis em um sistema. A definição 3.5 apresenta as condições necessárias para que um autômato seja ou não bloqueante.

Definição 3.5. *[11] Um autômato* G *é dito ser bloqueante se*

$$\overline{\mathcal{L}_m(G)} \neq \mathcal{L}(G).$$

Por outro lado, é dito ser não-bloqueante se

$$\overline{\mathcal{L}_m(G)} = \mathcal{L}(G).$$

3.3.2.3 Composição de autômatos

A modelagem de um SED por ADEFs pode seguir duas abordagens diferentes, a global e a local. Na abordagem global, deseja-se representar um sistema através de uma ADEF, na qual, todas as sequências possíveis de eventos do sistema sejam descritas por ele. Embora pareça interessante ter um único autômato que represente todo sistema, sua construção para sistemas de grande porte se torna uma tarefa árdua e qualquer mudança necessária pode representar a reconstrução de todo o modelo.

A modelagem de SEDs pode também ser vista como a composição de subsistemas que, quando compostos, constituem o sistema global. Assim, a abordagem local sugere a construção de subsistemas específicos e de restrições de coordenação entre eles. Com esta abordagem, a modelagem de sistemas de grande porte se torna mais fácil, assim como as possíveis mudanças no modelo.

A aplicabilidade da abordagem local para modelagem de SEDs por ADEF é garantida pela operação de composição de autômatos, conforme a definição 3.6 [18].

Definição 3.6. *Considerando dois ADEFs*

$$G_1 = (X_1, \Sigma_1, f_1, x_{0_1}, X_{m_1}) \text{ e } G_2 = (X_2, \Sigma_2, f_2, x_{0_2}, X_{m_2})$$

a composição síncrona $G_1||G_2$ *é definida como:*

$$G_1||G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1||2}, \{x_{0_1}, x_{0_2}\}, X_{m_1} \times X_{m_2})$$

em que a operação Ac toma a parte acessível e :

$$f_{1||2} : (X_1 \times X_2) \times (\Sigma_1 \cup \Sigma_2) \rightarrow (X_1 \times X_2)$$

Ou seja,

$$f_{1||2}((x_1,x_2),\sigma) = \begin{cases} (f_1(x_1,\sigma), f_2(x_2,\sigma)) & se\ \sigma \in \Sigma_1 \cap \Sigma_2\ e \\ & \sigma \in \Sigma_1(x_1) \cup \Sigma_2(x_2) \\ (f_1(x_1,\sigma), x_2) & se\ \sigma \in \Sigma_1\ e\ \sigma \notin \Sigma_2\ e \\ & \sigma \in \Sigma_1(x_1) \\ (x_1, f_2(x_2,\sigma)) & se\ \sigma \in \Sigma_2\ e\ \sigma \notin \Sigma_1\ e \\ & \sigma \in \Sigma_2(x_2) \\ indefinida & caso\ contrário \end{cases}$$

Em uma composição de autômatos, um evento comum e, isto é, $e \in \Sigma_1 \cap \Sigma_2$, apenas pode ser executado se ambos os autômatos o executarem ao mesmo tempo [11]. Caso um evento pertença à apenas um alfabeto, a composição será assíncrona. Quando os alfabetos são iguais $\Sigma_1 = \Sigma_2$ temos uma composição síncrona completa. Em contra partida, quando os alfabetos são complemente diferentes $\Sigma_1 \cap \Sigma_2 = \emptyset$ a composição é totalmente assíncrona.

Por fim, algumas propriedades da composição síncrona devem ser ressaltadas:

- $G_1||G_2 = G_2||G_1$;
- $(G_1||G_2)||G_3 = G_1||(G_2||G_3)$;
- A definição pode ser estendida para n autômatos.

3.4 Autômatos Temporizados

Nesta seção são definidas cadeias temporizadas pelo acoplamento de uma valor real de tempo com cada elemento de uma cadeia. Desta maneira, amplia-se a definição dos ADEF pela aceitação de cadeias

temporizadas, e definem-se linguagens temporizadas analogamente às linguagens apresentadas na seção 3.2.2.

3.4.1 Linguagem Temporazida

Uma cadeia temporizada é definida pelo acoplamento de um valor real do tempo a uma ação, denotada pelo par (a, t), onde $a \in \Sigma$ é uma ação tomada por um autômato $\mathcal{A}$ depois de $t \in \tau \subseteq \mathbb{R}_+$ unidades de tempo desde que $\mathcal{A}$ tenha iniciado, sendo $\tau = \tau_1\tau_2$ uma sequência infinita de valores de tempo $\tau_i \in \mathbb{R}_+$. Uma sequência de cadeias temporizadas (possivelmente infinita) é definida como $\xi = (a_1, t_1)(a_2, t_2)...$ $(a_i, t_i)...$ onde $t_i \leq t_{i+1}$ para todo $i \geq 1$.

Definição 3.7. *[3] Uma cadeia temporizada sobre um alfabeto Σ é um par (a, τ) onde $a = a_1a_2...$ é uma cadeia infinita sobre Σ e τ uma sequência temporal. Uma linguagem temporizada sobre Σ é um conjunto de cadeias temporizadas sobre Σ.*

Exemplo 3.2. *Sendo um alfabeto $\Sigma = \{a, b\}$. Define-se uma linguagem temporizada L_1 de todas as cadeias temporizadas (σ, t) tal que não existe a ação b para $t \geq 5.6$. Desta maneira, a linguagem L_1 é dada por*

$$L_1 = \{(\sigma, t) | \forall i.((t_i > 5.6) \rightarrow (\sigma_i = a))\}. \tag{3.1}$$

3.4.2 Autômato Temporizado

Um autômato temporizado (AT) é essencialmente um ADEF estendido com variáveis de valores reais. A mais comum destas variáveis é a representação do tempo, chamada de *clock*. O formalismo da utilização destas variáveis em autômatos de estados finitos é tratado em alguns estudos [3, 9, 10] destacando, principalmente, o enfoque da inserção do tempo. Nestes casos, todos os *clocks* são assumidos para

procederem em uma mesma taxa e medem a quantidade de tempo desde que tenham sido inicializados. Estes valores podem então ser comparados com valores reais e serem também reinicializados.

Seja X um conjunto finito de variáveis de *clock* e de variáveis inteiras, denota-se $B(X)$ o conjunto de restrições retangulares φ do tipo: $\varphi ::= x \sim k$ onde $k \in \mathbb{Z}$, $x \in X$ e $\sim \in \{<, \leq, =, >, \geq\}$ [9]. Esse conjunto se caracteriza como as guardas das transições sobre o conjunto X. A função de reinicialização das variáveis de *clock* e atualização das variáveis inteiras de X é definida por $F(X)$.

O AT, denominado $\mathcal{T}$, é definido como uma sêxtupla

$$\mathcal{T} = (L, l_0, \Sigma, X, I, R), \tag{3.2}$$

em que:

- L é um conjunto finito de *lugares*;
- $l_0 \in L$ o *lugar* inicial de $\mathcal{T}$;
- Σ um conjunto de ações;
- X um conjunto finito dos valores reais de *clock* e de variáveis inteiras;
- $I : L \longmapsto B(X)$ uma função de invariância;
- $R \subseteq L \times L \times \Sigma \times B(X) \times F(X)$ um conjunto de transições entre *lugares* com uma ação, uma guarda e uma função de reinicialização dos *clocks* e atualização das variáveis inteiras.

A semântica de um AT é definida por um sistema de transição em que um estado consiste num *lugar* e os valores atuais dos *clocks*. Existem dois tipos de transições entre estados. O autômato pode ou esperar

algum tempo (transição de espera), ou realizar uma ação (transição de ação) [9].

Sendo $a \in \Sigma$, $d \in \mathbb{R}_+$ e $g \in B(X)$, onde $x \in g$ representa que o valor da variável ou *clock* x satisfaz a guarda g, tem-se a definição 3.8.

Definição 3.8. *Seja $\mathcal{T} = (L, l_0, \Sigma, X, I, R)$ um AT. A semântica de $\mathcal{T}$ é definida como um sistema de transição*

$$(S, s_0, \delta) \tag{3.3}$$

onde

- $S \subseteq L \times \mathbb{R}_{\geq 0}$ *é o conjunto de estados;*
- $s_0 = (l_0, x_0)$ *é o estado inicial;*
- $\delta \subseteq S \times \{\mathbb{R}_{\geq 0} \cup \Sigma\} \times S$ *é a relação de transição definida por*
  - $(l, x) \xrightarrow{d} (l, x + d)$ *se* $\forall d' : 0 \leq d' \leq d \Rightarrow x \in I(l)$,
  - $(l, x) \xrightarrow{a} (l', x')$ *se* $l \xrightarrow{a,g} l', x \in g$ *e* $x' \in I(l')$.

Exemplo 3.3. *Neste exemplo, considera-se um robô que para atingir a posição final pode seguir a partir de uma bifurcação, por um caminho mais curto ou por um caminho intermediário. A Figura 3.2 apresenta um AT que representa os possíveis caminhos do robô, cuja descrição formal é relatada da seguinte maneira:*

- $L = \{inicio, decisao, pos01, fim\}$;
- $l_0 = \{inicio\}$;
- $\Sigma = \{a, b, c, d\}$
- $X = \{x, y\}$;
- $I : L \longmapsto B(X)$: $I(inicio) = (x <= 5), I(decisao) = (y <= 6)$;

- $R \subseteq L \times L \times \Sigma \times B(X) \times F(X) : R = \{(inicio, decisao, a, (x > 3), \emptyset), (decisao, pos01, b, (y > 3), (x := 0)), (decisao, fim, c, (y > 2), \emptyset), (pos01, fim, d, (x > 3), \emptyset)\}$.

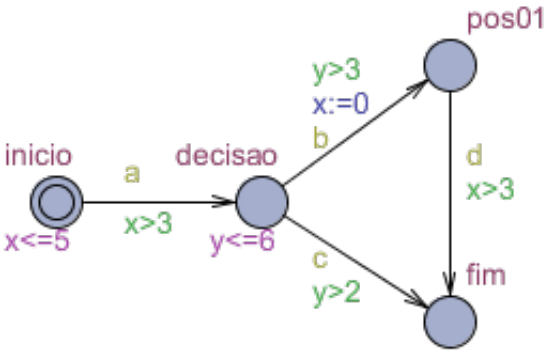


Figura 3.2: Autômato Temporizado (AT).

3.4.3 Representação por Autômatos Temporizados

3.4.3.1 Definição Geral

Com a inclusão do tempo nos SEDs, surge a necessidade de se utilizar um método que possibilite a representação de tais sistemas temporizados. De forma análoga aos ADEF, o autômato temporizado faz a conexão com as linguagens temporizadas inspecionando o diagrama de transição temporizada de estados. Porém, a utilização de estados marcados não é realizada em autômatos temporizados, por conseguinte, a linguagem marcada não se aplica para este método de representação. Considerando então todos os caminhos no diagrama de transição temporizada de estados, a linguagem gerada por um autômato temporizado é uma sequência possivelmente infinita de cadeias temporizadas, conforme definido abaixo.

Definição 3.9. *A linguagem gerada por* $\mathcal{T} = (L, l_0, \Sigma, X, I, R)$*, onde*

o sistema de transição é dado por (S, s_0, δ)*, é*

$$\mathcal{L}(\mathcal{T}) := \{a \in \Sigma^*,\ x \in X\ : \delta(s_0, (a \vee x))\ \textit{é definida}\}.$$

3.4.3.2 Sincronização de Autômatos Temporizados

Uma rede de AT pode ser sincronizada através de canais de sincronização. Estes canais, que funcionam de forma análoga ao produto síncrono de ADEF, são definidos nas transições dos modelos em que a sincronização deve ser realizada. São definidas da seguinte forma: $e!$ significa o envio do sinal e $e?$ o recebimento do sinal de sincronização. Portanto, dois AT podem ser sincronizados sobre suas transições se as guardas presentes nestas transições forem satisfeitas. Neste caso, quando existe a sincronização, as transições, nas quais são definidos os canais, disparam em paralelo, semelhante ao resultado do produto síncrono entre ADEFs. Três diferentes tipos de canais de sincronização são possíveis.

Canal Normal representa uma sincronização binária, na qual apenas duas transições podem ser sincronizadas ao mesmo tempo;

Canal Urgente é similar ao um canal normal, porém não é possível esperar em um estado precedente a uma transição que possua este canal;

Canal *Broadcast* permite a sincronização de uma transição que possua o sinal de envio $e!$ com várias outras transições de diferentes AT que possuam o sinal de recebimento $e?$. Neste caso, uma transição com o sinal de envio não precisa necessariamente de uma outra transição para ser disparada.

3.4.3.3 Rede de Autômatos Temporizados

Um sistema é modelado como uma rede de vários AT em paralelo [7]. O modelo é estendido com variáveis discretas, como as utilizadas em linguagem de programação, que fazem parte do estado. Um estado do sistema é definido pelos *lugares* de todos os autômatos, valores dos *clocks* e valores das variáveis discretas. Todo autômato pode disparar uma transição separadamente ou sincronizar com outro autômato, conforme descrito na seção anterior.

Sendo um vetor de *lugares* como $\bar{l} = [l_1, ..., l_n]$ e as funções de invariância compostas em uma função comum sobre o vetor de *lugares* $I(\bar{l}) = \wedge_i I_i(l_i)$, denota-se $\bar{l}[l'_i/l_i]$ o vetor onde o $i-$ésimo elemento l_i de $\bar{l}$ é substituído por l'_i. Assim, tem-se a definição da semântica de uma rede de AT.

Definição 3.10. *Seja* $\mathcal{T}_i = (L_i, l_0^i, \Sigma, X, I_i, R_i)$, *para* $i = \{1, ..., n\}$, *uma rede de autômatos temporizados sobre um conjunto de clocks* X *e ações* Σ. *A semântica é definida como um sistema de transição*

$$(S, s_0, \delta) \tag{3.4}$$

em que

- $S = (L_1 \times ... \times L_n) \times \mathbb{R}_{\geq 0}$ *é um conjunto de estados,*
- $s_0 = (\bar{l}_0, x_0)$ *é um estado inicial,*
- $\delta \subseteq S \times \{\mathbb{R}_{\geq 0} \cup \Sigma\} \times S$ *é a relação de transição definida por*
  - $(\bar{l}, x) \rightarrow (\bar{l}, x + d)$ *se* $\forall d' : 0 \leq d' \leq d \Rightarrow x + d \in I(\bar{l})$,
  - $(\bar{l}, x) \rightarrow (\bar{l}[l'_i/l_i], x')$, *se existe* $\exists l_i \xrightarrow{a,g} l'_i$ *tal que* $x \in g$ *e* $x' \in I(\bar{l})$,
  - $(\bar{l}, x) \rightarrow (\bar{l}[l'_j/l_j, l'_i/l_i], x')$, *se existe* $\exists l_i \xrightarrow{e?,g_i} l'i$ *e* $l_j \xrightarrow{e!,g_j} l'j$ *tal que* $x \in (g_i \wedge g_j)$ *e* $x' \in I(\bar{l})$.

Exemplo 3.4. *Continuando o Exemplo 3.3, é acrescentado um AT no sistema a fim de realizar a escolha do caminho a ser seguido pelo robô. Desta maneira, constitui-se uma rede de AT, onde os canais P, escolha do caminho intermediário, e F, escolha do caminho mais curto, representam os canais de sincronização. Neste caso os canais são considerados normais. A Figura 3.3 apresenta a rede de AT para este exemplo.*

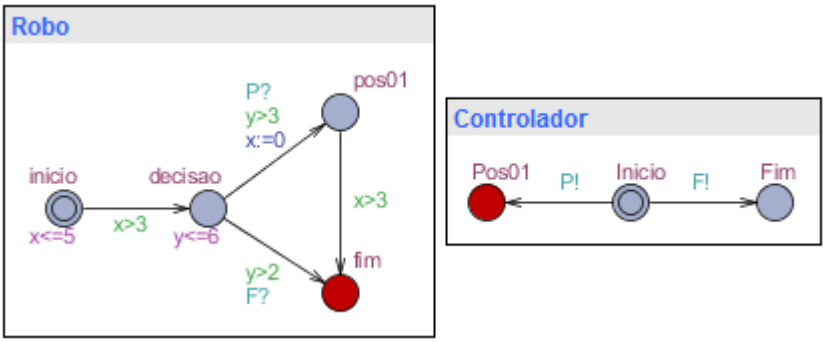


Figura 3.3: Rede de Autômatos Temporizados.

3.5 Conclusão

Neste capítulo foi realizado uma revisão a respeito dos conceitos utilizados em Sistemas a Eventos Discretos, assim como dos formalismos envolvidos para sua modelagem. Como foi visto anteriormente, diversas são as aplicações destes sistemas, abrangendo ainda mais sua aplicabilidade com a inserção do tempo em sua modelagem. É a partir destes formalismos que poderão ser sintetizados controles discretos para estes tipos de sistema. Nos capítulos seguintes, serão apresentados dois tipos de síntese, baseado na Teoria de Controle Supervisório (TCS) (Capitulo 4) e baseado em Autômato-Jogo Temporizado (AJT) (Capítulo 5).

Capítulo 4

Coordenação de SMR utilizando Teoria de Controle Supervisório

Devido a importância e aplicabilidade dos SEDs em diversas áreas de atuação faz com que seja altamente desejável encontrar soluções para problemas relacionados ao seu controle. Por este motivo, muitos pesquisadores têm desenvolvido modelos e formalismos matemáticos para a representação e controle destes sistemas. Porém, nenhuma abordagem se mostrou ser tão concisa quanto às equações diferenciais para os sistemas dinâmicos de variáveis contínuas [18]. Logo, nenhuma teoria de controle para SEDs é predominante. Dentre os modelos existentes, destaca-se o proposto por Ramadge e Wonham [43], baseado em Teoria de Linguagens e Autômatos. Em sua abordagem, utiliza-se a diferenciação entre o sistema a ser controlado, denominado *planta* e o controlador, chamado de *supervisor* [18].

Neste capítulo são apresentados os fundamentos da teoria de

controle supervisório, introduzida por [43], uma extensão desta teoria para uma abordagem modular local proposta por Queiroz e Cury [38], e por último, o desenvolvimento de um caso de uso para um sistema multi-robôs, utilizando TCS com 2 abordagens diferentes. Em cada abordagem será feito o desenvolvimento para as duas topologias, centralizada e distribuída, citadas em 2.3.1.

4.1 Controle Supervisório de SEDs

Um sistema a ser controlado pode ser composto por um ou mais subsistemas definidos por ADEFs. Eles representam funcionalidades específicas e comportamentos isolados. Atuando em conjunto, devem ser compostos a fim de obter o comportamento global. A composição do comportamento de cada subsistema pode ser identificada como a *planta* G, com comportamentos gerados $\mathcal{L}(G)$ e marcados $\mathcal{L}_m(G)$, respectivamente [18].

Como as linguagens $\mathcal{L}(G)$ e $\mathcal{L}_m(G)$ possuem cadeias indesejáveis que vão de encontro ao comportamento desejado para a planta, um conjunto de restrições deve ser definido para descrevê-lo evitando estas cadeias. Este conjunto é conhecido como a *especificação* E para a planta G.

Para que os subsistemas atuem de forma coordenada e sigam a especificação imposta, um agente controlador é introduzido no sistema. O controlador S, chamado *supervisor*, que de forma realimentada, conforme Figura 4.1, interage com a planta G, observa os eventos ocorridos em G e define, entre os eventos fisicamente possíveis, quais devem estar desabilitados, garantindo, desta maneira, que as cadeias indesejáveis, correspondentes aos maus comportamentos, sejam evitadas. De modo geral, podemos afirmar que S tem uma ação desabilitadora, assim, diz-se que S é um controle de natureza *permissiva* [18].

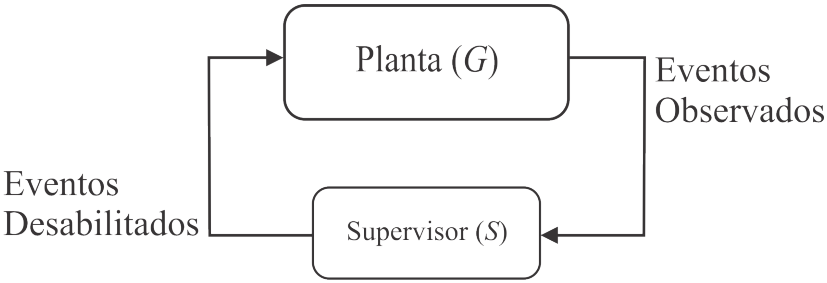


Figura 4.1: SED em malha fechada [18].

O controle supervisório foi primeiramente proposto com uma abordagem global dita monolítica [43], utilizando apenas um supervisor para o sistema global. Não obstante, devido aos vários problemas computacionais e de implementação que possui esta abordagem, extensões tais como as propostas em Wong e Wonham (1998) [46] e Queiroz (2000) [38] foram alvitradas no decorrer dos anos. Em especial, a abordagem modular proposta por Queiroz e Cury [38, 39, 40, 41] foi utilizada neste trabalho. As seções subsequentes apresentam estas abordagens e suas principais características.

4.1.1 Abordagem Monolítica

Conforme comentado anteriormente, na abordagem monolítica é sintetizado apenas um supervisor responsável pelo controle de toda a planta. Desta forma, dada uma planta G representada por um ADEF sobre o conjunto $\Sigma = \Sigma_c \cup \Sigma_u$, onde: Σ_c é o conjunto de eventos controláveis, e então, que podem ser inibidos e Σ_u o conjunto de eventos não-controláveis , que não podem ser inibidos, temos a definição da estrutura de controle Γ sobre G:

$$\Gamma = \{\gamma \in 2^{\Sigma} : \gamma \supseteq \Sigma_u\}$$

A condição de controle $\gamma \in \Gamma$ aplicada ao sistema contém o

conjunto ativo de eventos habilitado a ocorrer no sistema. Sendo que, a condição $\gamma \supseteq \Sigma_u$ indica que os eventos não-controláveis não podem ser desabilitados.

O supervisor pode ser representado por um autômato S, no qual as mudanças de estado são ativadas pelo disparo dos eventos da planta, de modo que a ação de controle sobre a planta G está implícita na estrutura de transição do supervisor S. Para tal, a ação de controle de S consiste em desabilitar na planta os eventos que não possam ocorrer no autômato S após uma cadeia de eventos observada. O sistema controlado definido por S/G pode ser descrito por um SED resultante da composição síncrona de S com G, ou seja, $S||G$. Neste caso, somente as transições permitidas em ambos autômatos são permitidas no sistema controlado [18]. Assim, podemos definir o comportamento do sistema em malha fechada como:

$$\mathcal{L}(S/G) = \mathcal{L}(S||G) \quad \text{e} \quad \mathcal{L}_m(S/G) = \mathcal{L}_m(S||G)$$

Sendo que a exigência de não bloqueio imposta ao supervisor é expressa pela condição:

$$\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}.$$

Desta maneira, em Cury (2001) [18] é definido formalmente o problema do controle supervisório, conforme o Problema 4.1 .

Problema 4.1. *[18] Dada uma planta G, com comportamento ($\mathcal{L}(G)$, $\mathcal{L}_m(G)$) e estrutura de controle Γ, definidos sobre o conjunto de eventos Σ; e especificações definidas por $A \subseteq E \subseteq \Sigma^*$; encontre um supervisor não bloqueante S para G tal que*

$$A \subseteq \mathcal{L}_m(S/G) \subseteq E, \tag{4.1}$$

onde as especificações A e E definem limites superior e inferior para o

comportamento do sistema em malha fechada.

4.1.1.1 Condições de Síntese do Controlador

A síntese de um controlador monolítico pode ser descrita usando as noções de controlabilidade e de $\mathcal{L}_m(G)$-fechamento.

Uma linguagem $K \subseteq \Sigma^*$, sendo $\overline{K}$ o prefixo-fechamento de K, é dita uma *linguagem controlável* de $\mathcal{L}(G)$, ou controlável em relação a G, ou simplesmente controlável, se

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}$$

Por definição, controlabilidade é uma propriedade de prefixo-fechado de uma linguagem. Deste modo, K é controlável se, e somente se, $\overline{K}$ também for controlável. Logo, a expressão pode ser reescrita como

$$\forall s \in \overline{K}, \forall e \in \Sigma_u, se \in \mathcal{L}(G) \Rightarrow se \in \overline{K}$$

Teorema 4.1. *[48] Seja $K \subseteq \mathcal{L}_m(G), K \neq \emptyset$. Existe um supervisor não bloqueante S para G tal que*

$$\mathcal{L}_m(S/G) = K$$

se e somente se K for controlável com relação a G.

Para uma linguagem $E \subseteq \Sigma^*$, o conjunto de todas as sublinguagens de E que são controláveis a respeito de G é descrito por $\mathcal{C}(E) = \{K \in E \quad | \quad K \text{ é controlável em relação a } G\}$ e é fechada por união [18]. Portanto, é possível provar que $\mathcal{C}(E)$ possui um elemento supremo denominado $sup\mathcal{C}(E)$.

A construção de um supervisor que restrinja o comportamento do sistema a uma linguagem especificada, as vezes, não possui solução.

O Teorema 4.1 proposto em Cury (2001)[18] apresenta a condição de satisfabilidade de um supervisor $sup\mathcal{C}(E)$.

Teorema 4.2. *Tendo as especificações $A \subseteq E \subseteq \Sigma^*$. O problema de controle supervisório possui solução se e somente se*

$$sup\mathcal{C}(E) \supseteq A.$$

Seguindo o teorema 4.2, o objetivo do controle monolítico é encontrar um supervisor S que possua o comportamento menos restritivo possível. Deste modo, o supervisor ótimo S é tal que $\mathcal{L}_m(S/G) = sup\mathcal{C}(E)$.

4.1.1.2 Metodologia para a síntese de supervisores ótimos

A metodologia para a síntese de supervisores ótimos foi inicialmente proposta por Ramadge e Wonham (RW) [43], e se baseia em três passos:

1. Obtenção de um modelo para a planta a ser controlada;
2. Obtenção de um modelo de representação das especificações a serem respeitadas;
3. Síntese de uma lógica de controle não bloqueante e ótima.

Inicialmente é realizada a definição dos subsistemas e dos dispositivos que irão fazer parte do sistema a ser controlado, e em seguida, a modelagem através de ADEFs das partes deste sistema. Então, a planta é obtida através da composição síncrona de todos os ADEFs que representam os subsistemas, conforme o que foi apresentado na seção 3.3.2.3. Por fim, de posse a planta a ser controlada, define-se a estrutura de controle Γ, indicando os conjuntos de eventos controláveis Σ_c e não-controláveis Σ_u.

A abordagem de controle monolítico é ilustrada pelo exemplo 4.1, retirado de Cury (2001) [18].

Exemplo 4.1 (**Pequena Fábrica**). *Considera-se uma Linha de Transferência (figura 4.2) que contenha duas máquinas M_1 e M_2 e um armazém intermediário de capacidade 1.*

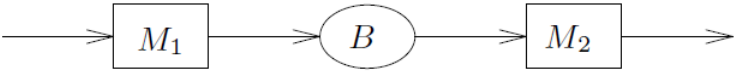
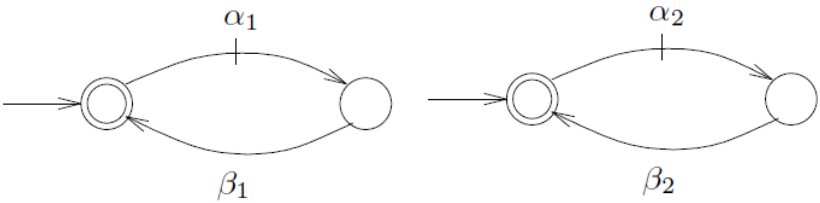


Figura 4.2: Linha de transferência.

Retirando-se a possibilidade de quebra das máquinas, quando M_i recebe o sinal α_i, a máquina carrega uma peça e opera sobre a mesma; ao término da operação, o sinal β_i é acionado, de forma não controlada, e o descarregamento da peça é efetuada. Os modelos das máquinas são apresentados pela figura 4.3. O conjunto Σ é composto por

$$\Sigma_c = \{\alpha_1, \alpha_2\}, \quad \Sigma_u = \{\beta_1, \beta_2\}.$$



(a) M_1 (b) M_2

Figura 4.3: Modelo das máquinas.

A composição dos modelos de cada máquina nos fornecerá o modelo da planta G, mostrado na figura 4.4.

De posse da planta, é necessário obter a linguagem $E \subset \mathcal{L}_m(G)$ que restrinja o comportamento do sistema a fim de que atenda aos

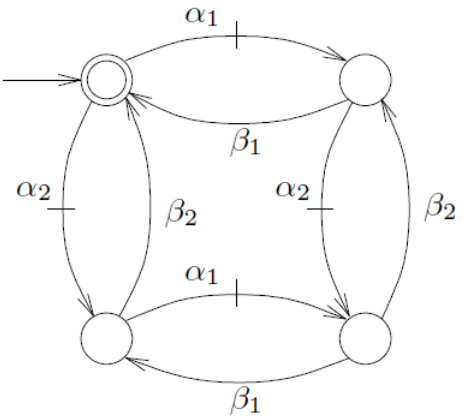


Figura 4.4: Modelo da planta.

objetivos de projeto. Como na modelagem da planta, a especificação também pode ser realizada localmente e em seguida composta utilizando composição síncrona. Portanto, para se obter a especificação global E, deve-se, primeiramente, construir os autômatos E_j para cada restrição j do sistema a ser controlado e, posteriormente, realizar a composição síncrona de cada uma das restrições com a planta G gerando o autômato H. Para se chegar a especificação global E, elimina-se de H os estados considerados proibidos e calcula-se sua componente co-acessível. Por fim, a especificação global E é dado por

$$E = L_m(H).$$

Exemplo 4.2. *Seguindo o exemplo 4.1, uma restrição de coordenação E_1 que impede overflow (M_1 tenta descarregar a peça no buffer já cheio) ou underflow (M_2 tenta carregar a peça com o buffer vazio) é modelada conforme a Figura 4.5. A cada ocorrência de β_1 (final da operação de M_1) apenas pode ocorrer α_2 (carregamento de M_2), assim alternando esses eventos. O estado marcado indica que a tarefa é somente concluída quando o buffer estiver vazio.*

O autômato H que representa a especificação E, obtida pela com-

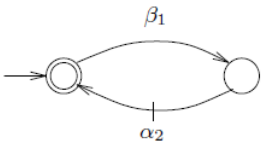


Figura 4.5: Autômato para a especificação de não *overflow* e não *underflow* do buffer B.

posição de E_1 com G, é apresentado pela figura 4.6.

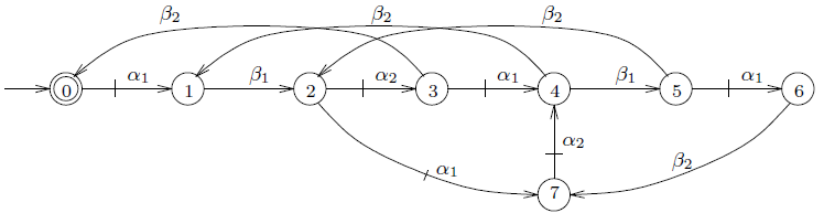


Figura 4.6: Autômato H para a especificação E.

Para terminar a resolução do problema de controle supervisório, a síntese do supervisor S que implementa a lógica não bloqueante ótima, no sentido de menos restritiva possível deve ser calculada. Segundo o teorema 4.1, existe um supervisor não bloqueante, tal que $\mathcal{L}_m(S/G)$, caso $E \subset \mathcal{L}_m(G)$. Porém, como E pode não atender a condição de controlabilidade, se faz necessário o cálculo de uma linguagem controlável que mais se aproxime de E. Esta linguagem é dada por $sup\mathcal{C}(E)$ e representa a lógica ótima de supervisão.

Para o cálculo de $sup\mathcal{C}(E)$, deve-se identificar os maus estados de H e eliminá-los. Um mau estado é identificado quando existe uma cadeia $\sigma \in \Sigma_u$, tal que σ pertença a cadeia de eventos de G e não pertença a cadeia de eventos de H. Após esta etapa, o cálculo da componente trim de H deve ser realizado. Desta maneira, como $\mathcal{L}_m(H) = E \subset \mathcal{L}_m(G)$, podemos fazer $S = H$. Seguindo a complementação do exemplo 4.1, o supervisor S é mostrado pela figura 4.7.

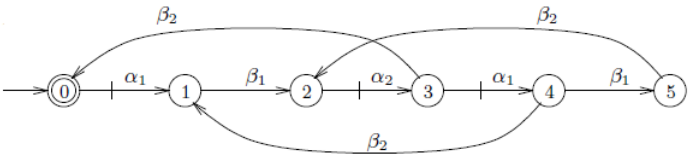


Figura 4.7: Autômato representando a máxima linguagem controlável (S).

Para ser feita a implementação do supervisor, é utilizado um programa de controle que faça a aplicação do autômato S no sistema.

4.1.2 Abordagem Modular Local

Tendo em vista a complexidade da síntese de controladores utilizando a abordagem monolítica, conforme proposto por Ramadge e Wonham, uma solução para a diminuição desta complexidade foi o uso combinado de subcontroladores para solucionar o problema original. Uma primeira solução foi proposta também por Ramadge e Wonham [47], porém esta abordagem não se mostrou eficiente devido a má representação do paralelismo entre autômatos finitos. Uma outra solução é apresentada em diversos trabalhos [38, 39, 40, 41], que permitem explorar a modularidade das especificações da planta, na qual foi intitulada como controle modular local.

A síntese modular dos controladores locais, chamados de supervisores modulares, é realizada a partir de uma planta local para cada especificação, cujos eventos estão presentes na especificação. Nesta arquitetura de controle cada supervisor atua somente sobre um subsistema (ou um conjunto de subsistemas) da planta global. A figura 4.8 expõe como se organiza esta estrutura.

Cada subsistema é representado por um autômato $G_i = (X_i, \Sigma_i, f_i, X_0 i, X_m i)$, onde $i \in I = \{1, ..., n\}$ e n o número de subsistemas. Desta maneira, tem-se $G = \|_{i=1}^{n} G_i$, com alfabeto de eventos $\Sigma =$

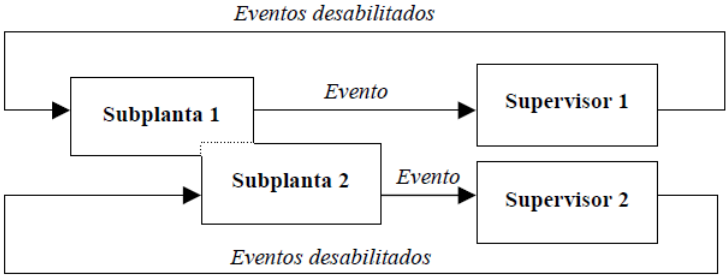


Figura 4.8: Arquitetura da abordagem de controle modular local [41].

$\bigcup_{i=1}^{n} \Sigma_i$. Desde que a modularidade local seja válida, conforme definição 4.1, a síntese da máxima linguagem controlável de múltiplas especificações pode ser executada diretamente a partir das especificações locais sem perda de performance em relação à solução monolítica (e, por conseguinte, à solução modular clássica). Uma representação dos supervisores locais é mostrada na figura 4.9.

Definição 4.1. *[39] Sejam* $E_i \subseteq \Sigma^*$, *onde* $i \in I = \{1, ..., n\}$, *e* n *o número de subsistemas. O conjunto de linguagens* $\{E_i, i \in I\}$ *é localmente modular se* $\|_{i=1}^{n} \overline{L_i} = \overline{\|_{i=1}^{n} L_i}$.

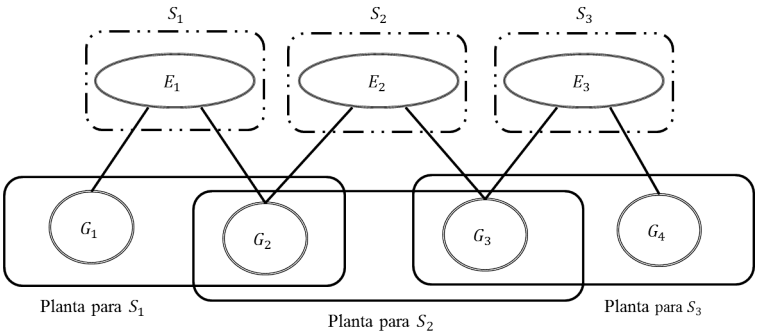


Figura 4.9: Controle supervisório modular local.

Utilizando o teorema 4.3 para fundamentar a abordagem da síntese de controle supervisor modular, podemos afirmar que para cada

especificação é possível construir um supervisor não bloqueante. Para isso, cada especificação local deve ser expressa em termos dos subsistemas locais (G_i) afetadas por ela.

Teorema 4.3. *[39] Se* $\{sup\mathcal{C}(E_{local,i}, G_{local,i}), i \in I\}$ *é localmente modular, então*

$$sup\mathcal{C}(\cap_{i=1}^{n} K_{global}, G_{global}) = \|_{i=1}^{n} sup\mathcal{C}(E_{local,i}, G_{local,i}). \quad (4.2)$$

4.1.2.1 Metodologia para a síntese de supervisores modulares locais

A metodologia para a síntese de supervisores modulares locais, proposta por Queiroz e Cury [41], segue uma sequência de passos, correspondentes ao cálculo da máxima linguagem controlável contida em cada especificação local, ao teste de conflito e à redução de supervisores.

Inicialmente, modela-se cada componente elementar do sistema e faz-se a composição dos subsistemas síncronos. De posse das partes que compõe o sistema global, as especificações E_i devem ser modeladas isoladamente. Então, a planta local é obtida de modo que cada especificação seja composta com os subsistemas com os quais tenham eventos em comum. A linguagem de cada planta local que satisfaz a especificação é calculada através do produto síncrono de cada planta local com sua respectiva especificação genérica e então, a máxima linguagem controlável é obtida.

Finalizada esta etapa, a modularidade local das linguagens resultantes deve ser verificada através do teste de conflito. Aplicando o Teorema 4.3, pode-se verificar se os supervisores são modulares locais. Caso não forem, os conflitos devem ser resolvidos através da abordagem monolítica ou com o uso de outras abordagens mais elaboradas encontradas na literatura [41]. Mas se forem modulares locais, deve-se então implementar um supervisor reduzido para cada linguagem controlável.

Um algoritmo formal para minimização de supervisores é utilizado de modo que todos os estados do supervisor original são agrupados no menor número possível de blocos, representando uma estrutura determinística de transição entre os blocos [41].

4.2 Projeto de um Coordenador de Sistemas Multi-Robôs pela abordagem TCS Monolítico

Considerando o estudo de caso apresentado na Seção 2.3.1, será apresentado o desenvolvimento do projeto para SMR utilizando a abordagem TCS monolítica para as duas topologias sugeridas: *centralizada* e *distribuída*. Porém, em um primeiro momento, para mostrar o método centralizado será apresentado o desenvolvimento do plano de tarefas para apenas um robô R_1 dentro do espaço de trabalho proposto no Capítulo 2. O plano a ser gerado deve levar o robô a atingir a posição 12 e evitar a colisão com as portas e o obstáculo fixo, garantindo assim as propriedades de alcançabilidade e segurança.

Os eventos dos movimentos dos robôs são identificados segundo qual robô está fazendo o movimento e quais são a origem e o destino deste. Por exemplo, R_1 deseja ir da posição 3 para a 6, desta maneira, o evento será $R_1.0306$. Assim, primeiramente aparece a identificação do robô, depois o lugar de origem e, por fim, o lugar de destino.

Para a síntese dos controladores foi utilizado uma ferramenta para solucionar sistemas de eventos discretos IDES[1].

[1] https://qshare.queensu.ca/Users01/rudie/www/software.html

4.2.1 Coordenador para o caso de 1 Robô

O sistema com apenas 1 robô no espaço de trabalho, possui um problema mais simples pelo motivo de não possuir problemas de colisão entre robôs. Este estudo de caso simplificado tem como objetivo apresentar o método a ser aplicado.

4.2.1.1 Planta

A planta para este sistema é composta do modelo dos possíveis movimentos do robô no espaço de trabalho, incluindo os obstáculos fixos, com os modelos de visualização das portas pelos robôs. A figura 4.10 apresenta o modelo dos possíveis movimentos de R_1, onde os estados indicam a posição do robô no espaço de trabalho e as transições são representadas conforme descrito anteriormente. Neste modelo, a célula do obstáculo fixo (posição 8) é proibida para o robô.

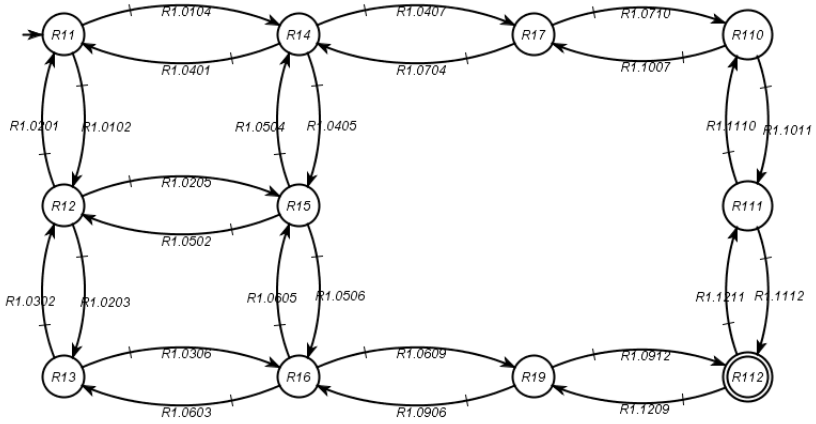


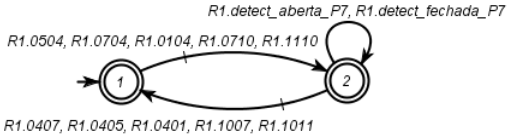
Figura 4.10: Modelo do movimento do robô R_1.

As portas P_7 e P_9, presentes nas posições 7 e 9, foram modeladas conforme são vistas pelo robô. Os modelos, representados pela Figuras 4.11(a) e 4.11(b), levam em consideração que apenas nas células

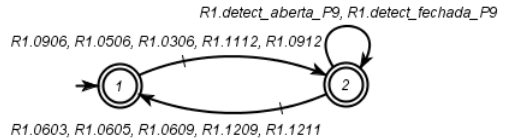
vizinhas à porta o robô poderá identificar seu estado. Por exemplo, o robô poderá apenas identificar a porta P_7 nas células 4 e 10 enquanto a porta P_9 na célula 6 e 12. Os eventos não controláveis representam a detecção do estado da porta pelo robô.

Logo, temos como eventos não controláveis das portas:

$$\begin{aligned}\Sigma_u &= \{R1.detect\_aberta\_P7, R1.detect\_fechada\_P7, \\ &\quad R1.detect\_aberta\_P9, R1.detect\_fechada\_P9\}. \end{aligned} \quad (4.3)$$



(a) Modelo de visualização da porta P_7 pelo robô R_1 ($VisP7\_R1$).



(b) Modelo de visualização da porta P_9 pelo robô R_1 ($VisP9\_R1$).

Figura 4.11: Modelos da visualização das portas pelo robô R_1.

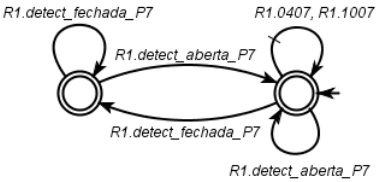
Desta maneira, o produto síncrono entre o modelos que compõem o comportamento da visualização das portas pelos robôs $VisP7\_R1$ e $VisP9\_R1$ com o modelo dos movimentos de R_1 fornece a planta para este sistema conforme a equação abaixo.

$$G = R_1||VisP7\_R1||VisP9\_R1 \quad (4.4)$$

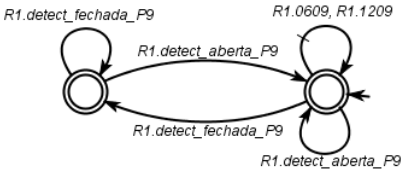
O autômato resultante desta composição apresentou 11 estados e 34 transições.

4.2.1.2 Especificação

A modelagem das restrições de coordenação entre o robô e as portas foi definida restringindo as ações dos movimentos do robô que o levam até a porta que esteja fechada. Por exemplo, quando ocorrer o evento $R1.detect\_aberta\_P7$, o robô poderá alcançar a posição 7 do espaço de trabalho. Porém, quando ocorrer $R1.detect\_fechada\_P7$, o robô não poderá atingir tal posição. Ambos os modelos $ResP7\_R1$ e $ResP9\_R1$ possuem a mesma estrutura e são apresentados pelas Figuras 4.12(a) e 4.12(b).



(a) Restrição $ResP7\_R1$.



(b) Restrição $ResP9\_R1$.

Figura 4.12: Modelos das restrições nas portas.

A especificação para o sistema, definida pela composição das restrições juntamente com a planta G, conforme a equação 4.5, levou a um autômato H para a especificação E com 40 estados e 120 transições.

$$H = G||ResP7\_R1||ResP9\_R1 \tag{4.5}$$

4.2.1.3 Síntese do Supervisor

A síntese do supervisor para este sistema de 1 robô foi realizada através do cálculo de $SupC(E)$, no qual, o supervisor sintetizado possui

40 estados e 120 transições, conforme apresentado pela Figura 4.13.

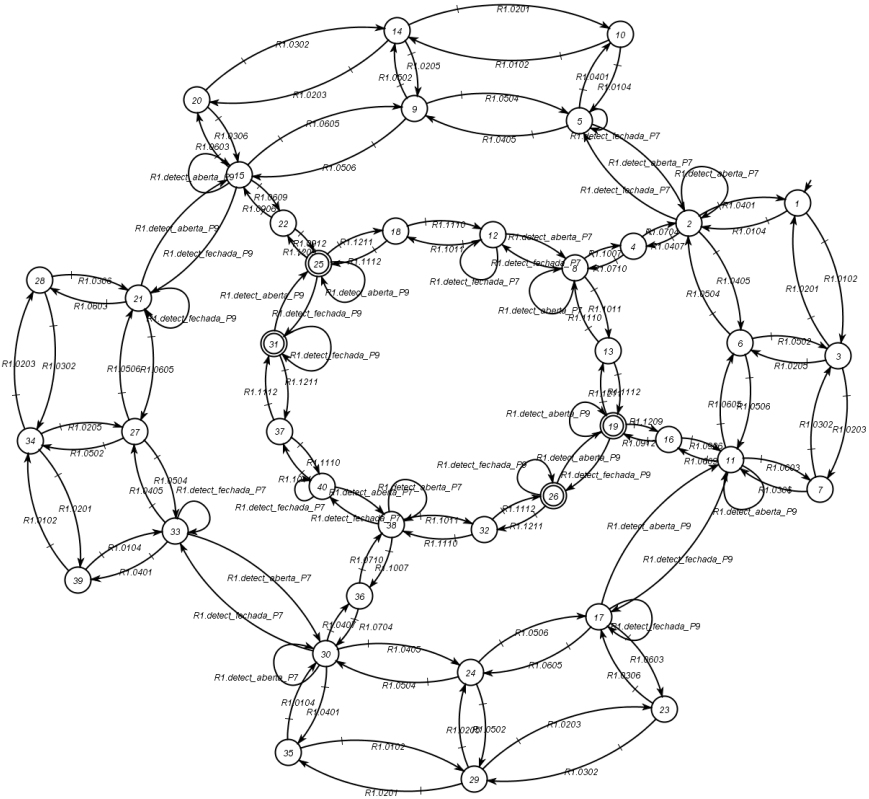
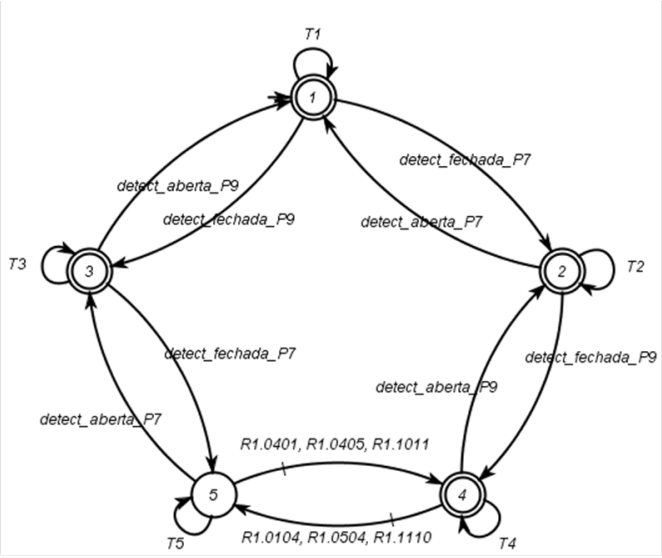


Figura 4.13: Modelo do supervisor monolítico.

Também foi realizada a redução do coordenador, gerando um supervisor com 5 estados e 104 transições, conforme a apresentado na Figura 4.14. A Tabela 4.1 apresenta os eventos desabilitados em cada estado do supervisor reduzido.



T1: R1.0102, R1.0104, R1.0201, R1.0203, R1.0205, R1.0302, R1.0306, R1.0401, R1.0405, R1.0407, R1.0502, R1.0504, R1.0506, R1.0603, R1.0605, R1.0609, R1.0704, R1.0710, R1.0906, R1.0912, R1.1007, R1.1011, R1.1110, R1.1112, R1.1209, R1.1211, R1.detect_aberta_P7, R1.detect_aberta_P9.
T2: R1.0102, R1.0104, R1.0201, R1.0203, R1.0205, R1.0302, R1.0306, R1.0401, R1.0405, R1.0502, R1.0504, R1.0506, R1.0603, R1.0605, R1.0609, R1.0906, R1.0912, R1.1011, R1.1110, R1.1112, R1.1209, R1.1211, R1.detect_fechada_P7, R1.detect_aberta_P9.
T3: R1.0102, R1.0104, R1.0201, R1.0203, R1.0205, R1.0302, R1.0306, R1.0401, R1.0405, R1.0407, R1.0502, R1.0504, R1.0506, R1.0603, R1.0605, R1.0704, R1.0710, R1.1007, R1.1011, R1.1110, R1.1112, R1.1211, R1.detect_aberta_P7, R1.detect_fechada_P9.
T4: R1.0102, R1.0201, R1.0203, R1.0205, R1.0302, R1.0306, R1.0502, R1.0506, R1.0603, R1.0605, R1.1112, R1.1211, R1.detect_fechada_P9.
T5: R1.detect_fechada_P7.

Figura 4.14: Modelo do supervisor monolítico reduzido.

Estado	Evento Desabilitado
1	$R1.0407$
1	$R1.1007$
2	$R1.0609$
2	$R1.1209$
3	$R1.0609$
3	$R1.1209$
4	$R1.0407$
4	$R1.1007$

Tabela 4.1: Eventos desabilitados no supervisor reduzido.

4.2.2 Coordenador Centralizado para o caso de 2 robôs

A partir deste momento, todas as abordagens desenvolvidas por TCS utilizarão um SMR com dois robôs. Nesta seção, será utilizado o conceito de um controlador único, centralizado, que controlará os movimentos de todos os robôs do sistema. Com objetivos independentes pré-determinados e restrições de não colisão com obstáculos fixos e dinâmicos e de não colisão entre robôs, estudou-se o caso de dois robôs R_1 e R_2 no espaço de trabalho descrito anteriormente.

4.2.2.1 Planta

No contexto de SMR, o modelo dos robôs é composto dos movimentos dos dois robôs R_1 e R_2 presentes no espaço de trabalho. O modelo de R_1 continua o mesmo apresentado pela Figura 4.10, enquanto o modelo de R_2, apesar de ser estruturalmente semelhante, possui estado inicial na posição 3, estado marcado na posição 10, alfabeto diferente e com a proibição de atingir o obstáculo fixo (posição 8). A Figura 4.15 apresenta o modelo do segundo robô.

Os modelos dos autômatos para a visualização das portas pelo robô R_1 são os mesmos do caso anterior. Porém, dois novos modelos são acrescentados, representando a visualização das portas pelo robô R_2. Estes novos modelos estão representados pelas figuras 4.16(a) e 4.16(b).

Temos como eventos não controláveis:

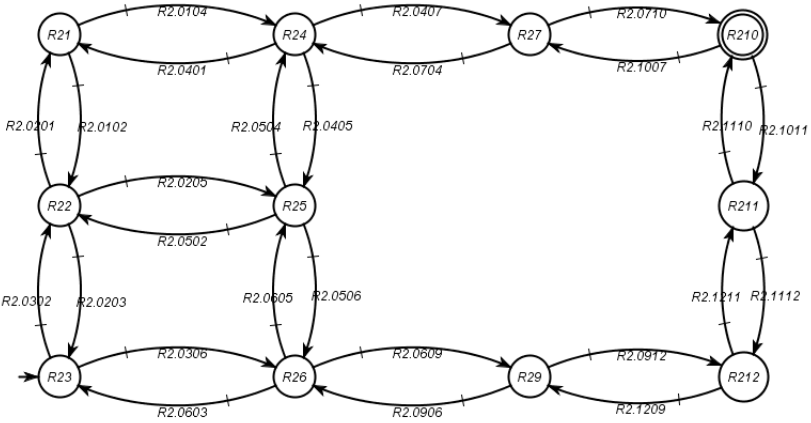
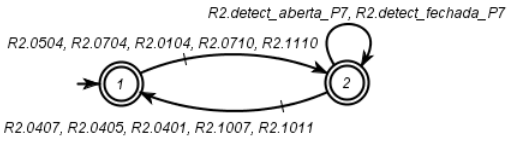


Figura 4.15: Modelo do movimento do robô R_2.

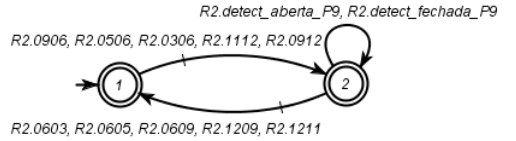
$$\begin{aligned}\Sigma_u \quad = \quad & \{R1.detect\_aberta\_P7, R1.detect\_fechada\_P7, \\ & R1.detect\_aberta\_P9, R1.detect\_fechada\_P9, \\ & R2.detect\_aberta\_P7, R2.detect\_fechada\_P7, \\ & R2.detect\_aberta\_P9, R2.detect\_fechada\_P9\}. \end{aligned} \quad (4.6)$$

O modelo da planta G é obtido da mesma forma através da composição síncrona entre os modelos da visualização das portas pelos robôs e os modelos dos movimentos dos robôs R_1 e R_2, conforme apresentado na Equação 4.7. O autômato G resultante tem 121 estados e 748 transições.

$$G = R_1||R_2||VisP7\_R1||VisP9\_R1||VisP7\_R2||VisP9\_R2. \quad (4.7)$$



(a) Modelo de visualização da porta P_7 pelo robô R_2 ($VisP7\_R2$).

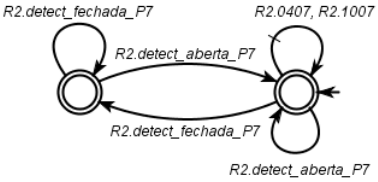


(b) Modelo de visualização da porta P_9 pelo robô R_2 ($VisP9\_R2$).

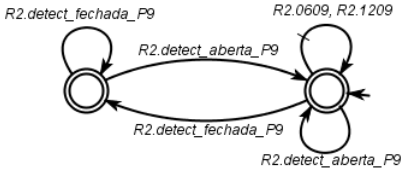
Figura 4.16: Modelos da visualização das portas pelo robô R_2.

4.2.2.2 Especificação

As restrições de coordenação do robô R_1 através das portas foram as mesmas do caso anterior. Neste caso, é realizado o acréscimo das restrições de coordenação do robô R_2 através das portas, conforme os modelos ilustrados pelas Figuras 4.17(a) e 4.17(b).



(a) Restrição $ResP7\_R2$.

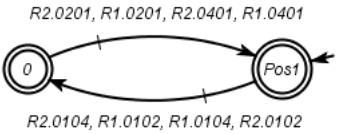


(b) Restrição $ResP9\_R2$.

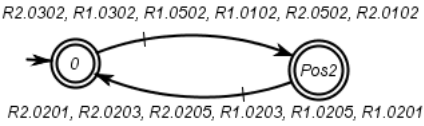
Figura 4.17: Modelos das restrições nas portas.

Para este problema uma restrição adicional para evitar a colisão entre robôs deve ser incorporada. Em outras palavras, os robôs não podem assumir a mesma posição no espaço de trabalho em um deter-

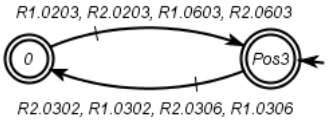
minado instante, ou seja, não colidirem. Nas figuras 4.18 estão representados os modelos destas restrições, onde o estado representado por $Posx$ indica que a posição x está ocupada por algum robô. Para efeito de simplificação serão mostrados apenas 4 modelos, pois eles totalizam 12 modelos e todos possuindo uma arquitetura semelhante.



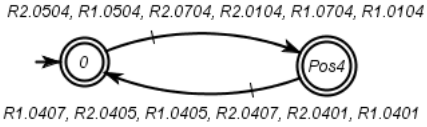
(a) Restrição Res_{Pos_1}



(b) Restrição Res_{Pos_2}



(c) Restrição Res_{Pos_3}



(d) Restrição Res_{Pos_4}

Figura 4.18: Modelos das restrições de lugar.

A especificação para o sistema, definida pela composição das restrições juntamente com a planta G, conforme a equação 4.8, nos levou a um autômato H para a especificação E com 1448 estados e 8000 transições.

$$\begin{aligned} H &= G||ResP7\_R1||ResP9\_R1||ResP7\_R2||ResP9\_R2|| \\ &\quad Res_{Pos_1}||Res_{Pos_2}||...||Res_{Pos_{12}}. \end{aligned} \tag{4.8}$$

4.2.2.3 Síntese do Supervisor

A síntese do controlador centralizado foi realizado através do cálculo de $SupC(E)$. Como resultado, obteve-se um autômato com

1448 estados e 8000 transições, que representa o supervisor centralizado para este sistema, portanto o plano a ser seguido pelos dois robôs.

4.2.3 Coordenador Distribuído para o caso de 2 Robôs

A abordagem distribuída considera que o controle é implementado em cada robô, de maneira que, apenas seus movimentos são controláveis, enquanto os outros robôs têm seus movimentos controlados também por controladores próprios. Porém, no caso aqui tratado, cada robô tem a informação da posição dos outros robôs no espaço de trabalho. O controlador apresentado a seguir é o do robô R_1, sendo que, para este, o robô R_2 é um obstáculo dinâmico presente no espaço de trabalho. Similarmente, pode-se calcular o controlador para R_2, considerando R_1 como obstáculo dinâmico.

4.2.3.1 Planta

Para o caso onde calcula-se o controlador do robô R_1, considerando R_2 como obstáculo dinâmico, existe uma diferença básica entre os modelos dos dois robôs. O primeiro, no qual será implementado o supervisor, possui todos seus eventos controláveis. A Figura 4.10 mostra o modelo de R_1. O robô R_2 não será influenciado pelo supervisor a ser calculado. Portanto, seu modelo de possíveis movimentos deve estar incluso no modelo da planta com todos os seus eventos não-controláveis. A figura 4.19 mostra a modelagem de R_2.

Em relação a visualização das portas do ambiente pelos robôs, os modelos que as representam são considerados os mesmos do caso para um robô (Seção 4.2.1). Pois, devido às características do controle distribuído, apenas interessa o funcionamento da porta em relação a passagem do robô a ser controlado, no caso, o robô R_1. Logo, os modelos da visualização das portas utilizados para este caso são os mesmos das figuras 4.11(a) e 4.11(b).

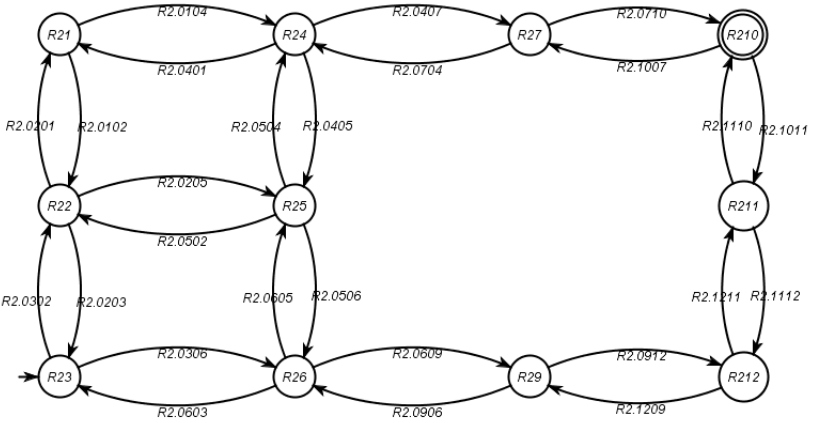


Figura 4.19: Modelo do movimento do robô R_2.

O modelo da planta G_{R_1} é obtido através da composição entre os modelos da visualização das portas pelos robôs e o modelo da composição dos movimentos dos robôs R_1 e R_2, conforme Equação 4.9. O autômato gerado para a planta G_{R_1} é formado por 121 estados e 660 transições.

$$G_{R_1} = R_1||R_2||VisP7\_R1||VisP9\_R1. \tag{4.9}$$

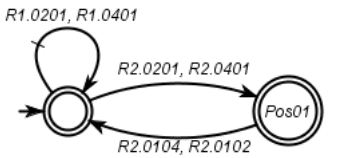
4.2.3.2 Especificação

O controlador a ser gerado tem como objetivo somente o controle dos movimentos de R_1. Em consequência, as restrições de coordenação do robô através das portas são consideradas apenas para esse robô e são as mesmas do caso de um robô (Seção 4.2.1). As Figuras 4.12(a) e 4.12(b) representam estas restrições.

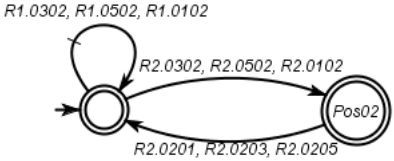
As restrições de ocupação das posições no ambiente de trabalho pelos robôs para este caso difere do controle centralizado. O impedimento da colisão entre robôs é garantido através da combinação dos

supervisores de cada robô presente no espaço de trabalho. Portanto, cada robô é apenas responsável por não colidir com os outros robôs e, desta maneira, garantindo a não colisão entre os robôs presentes no espaço de trabalho. Por exemplo, o controlador do robô R_1 evitará que este choque com o robô R_2. Por outro lado, o controlador do robô R_2 evitará que este choque com o robô R_1.

As Figuras 4.20(a), 4.20(b), 4.20(c) e 4.20(d) mostram os modelos das restrições de lugar do robô R_1, onde o estado representado por $Posx$ indica que a posição x está ocupada pelo robô R_2 e o robô R_1 não pode ocupar tal posição. Foram apresentadas apenas 4 restrições por razão de simplicidade, mas são 12 restrições de lugar ao todo.

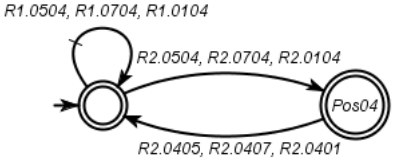


(a) Restrição Res_{Pos_1}



(b) Restrição Res_{Pos_2}

(c) Restrição Res_{Pos_3}



(d) Restrição Res_{Pos_4}

Figura 4.20: Modelos das restrições de lugar do robô R_1.

A especificação final para o robô R_1, modelada pela composição das restrições juntamente com a planta G_{R_1}, conforme a equação 4.10, nos levou a um autômato H para a especificação E com 440 estados e 2272 transições.

$$H_{R_1} = G_{R_1}||ResP7\_R1||ResP9\_R1|| \\ Res_{Pos_1}||Res_{Pos_2}||...||Res_{Pos_{12}} \quad (4.10)$$

4.2.3.3 Síntese do Controlador

O controle supervisório monolítico distribuído, para o robô R_1, foi calculado por $SupC(E)$ e gerou um autômato com 440 estados e 2272 transições. Para o controle supervisório do robô R_2, calculado de forma similar, o autômato gerado também teve obviamente o mesmo número de estados e transições, já que os modelos são simétricos.

Nota-se que a característica distribuída aqui representada se remete ao uso da abordagem modular local, de modo que podemos analisar este caso como um conjunto de supervisores modulares. Desta maneira, conforme apresentado no capítulo anterior, pode haver um conjunto de supervisores conflitantes que impedem o controle modular de ser diretamente aplicado, neste caso o distribuído. Assim, foi realizado o teste de não conflito entre os supervisores do robô R_1 e do robô R_2, que confirmou a modularidade destes controladores.

Para efeitos de comparação, foi realizado o produto síncrono dos supervisores distribuídos. O resultado deste produto originou um autômato igual ao do caso centralizado (1448 estados e 8000 transições). Este resultado já era esperado, consoante o teorema 4.3. Portanto, pode-se concluir que o produto síncrono dos supervisores distribuídos, i.e., de todos os robôs, fornece o supervisor centralizado. De forma intuitiva, pode-se afirmar que o comportamento global de todos os robôs com controle distribuído no espaço de trabalho é exatamente o mesmo ao caso de um controle centralizado destes robôs.

4.3 Projeto de um Coordenador de Sistemas Multi-Robôs pela abordagem TCS Modular Local

Considerando o mesmo estudo de caso da Seção 4.2, será apresentado o desenvolvimento do projeto para SMR utilizando a abordagem TCS modular local, conforme Seção 4.1.2 , para as duas topologias sugeridas: *centralizada* e *distribuída*.

4.3.1 Coordenador Centralizado para o caso de 2 Robôs

Nesta seção, será utilizado o conceito de um controlador único, centralizado, que controlará os movimentos de todos os robôs do sistema.

4.3.1.1 Planta

Para o caso da utilização de controle modular centralizado, será utilizada a mesma modelagem dos robôs no controle monolítico centralizado (Seção 4.2.2), sendo R_1 e R_2 representados pelas Figuras 4.10 e 4.15, respectivamente. Quanto aos modelos da visualização das portas pelos robôs também são os mesmos do controle monolítico centralizado, representados pelas Figuras 4.11(a), 4.11(b), 4.16(a) e 4.16(b) . Desta maneira, todos os modelos locais são iguais ao caso monolítico, porém sua composição é realizada de uma forma diferente, que dependerá da síntese dos controladores modulares locais.

4.3.1.2 Especificações

Os modelos das restrições de coordenação dos robôs entre as portas (Figura 4.12(a), 4.12(b), 4.17(a) e 4.17(b)) e da ocupação das posições (Figuras 4.18) serão os mesmos que no caso centralizado monolítico, apresentado na Seção 4.2.2. A utilização destas especificações é baseada no subsistema em que se deseja sintetizar um supervisor modular. Desta maneira, não se obtém uma especificação global final para este caso.

4.3.1.3 Síntese de Supervisores Modulares

A diferença desta abordagem com a monolítica, está na construção do supervisor. No caso anterior, um supervisor global coordena toda a planta e todas as restrições. Agora, diversos supervisores serão sintetizados, cada um sendo responsável por uma parte da planta e uma restrição específica.

Logo, definimos como G_i a parte da planta a ser supervisionada, R_i o autômato de restrições, H_i o autômato que leva à especificação final E_i para uma parte do sistema, S_i o supervisor modular do cálculo de $SupC(E_i)$ e Sr_i o supervisor reduzido de S_i, sendo $i \in \{P7\_R1, P9\_R1, P7\_R2, P9\_R2, Pos_1, Pos_2, ..., Pos_{12}\}$. Cada valor de i representa uma configuração modular do sistema definidos pelos 15 grupos de especificação respectivamente: restrição da porta 7 pelo robô $R_1(ResP7\_R1)$; restrição da porta 9 pelo robô R_1 $(ResP9\_R1)$; restrição da porta 7 pelo robô R_2 $(ResP7\_R2)$; restrição da porta 9 pelo robô R_2 $(ResP9\_R2)$; e Restrição de lugar (Res_{Pos_x}), onde x representa o lugar em questão.

A restrição para $i = P7\_R1$ leva em consideração o modelos do robôs R_1, o modelo de visualização da porta P_7 pelo robô R_1 $(VisP7\_R1)$ para calcular a planta G_{P7_R1}. A composição desses com o modelo da restrição da porta 7 $(ResP7\_R1)$ permite definir o modelo

H_{P7_R1}, onde $E_{P7\_R1} = L_m(H_{P7\_R1})$. As equações abaixo mostram como é obtido o supervisor para este módulo.

$$G_{P7\_R1} = R_1||VisP7\_R1 \quad (4.11)$$
$$H_{P7\_R1} = G_{P7\_R1}||ResP7\_R1 \quad (4.12)$$
$$S_{P7\_R1} = SupC(E_{P7\_R1}) \quad (4.13)$$

A restrição para $i = P9\_R1$ leva em consideração o modelos do robôs R_1, o modelo de visualização da porta P_9 pelo robô R_1 ($VisP9\_R1$) para calcular a planta G_{P9_R1}. A composição desses com o modelo da restrição da porta 9 ($ResP9\_R1$) permite definir o modelo H_{P9_R1}, onde $E_{P9\_R1} = L_m(H_{P9\_R1})$. As equações abaixo mostram como é obtido o supervisor para este módulo.

$$G_{P9\_R1} = R_1||VisP9\_R1 \quad (4.14)$$
$$H_{P9\_R1} = G_{P9\_R1}||ResP9\_R1 \quad (4.15)$$
$$S_{P9\_R1} = SupC(E_{P9\_R1}) \quad (4.16)$$

As restrições para $i = P7\_R2$ e $i = P9\_R2$ são calculadas da mesma maneira que nos dois casos anteriores, alterando apenas o modelo do robô R_1 pelo modelo do robô R_2, o modelo da visualização das portas pelo robô R_1 pelo modelo da visualização das portas pelo robô R_2 e o modelo da restrição das portas do robô R_1 pelo modelo da restrição das portas do robô R_2.

E por fim, a restrição para $i = Pos_x$ leva em consideração as restrições de lugar no espaço de trabalho. A planta G_{Pos_x} é calculada a partir dos modelos dos robôs R_1 e R_2. A especificação é então obtida com a composição do modelo da planta com o modelo de restrição de lugar Res_{Pos_x}, onde x indica uma posição no espaço de trabalho. As

equações abaixo nos indicam como é calculado o supervisor para este módulo, onde $E_{Pos_x} = L_m(H_{Pos_x})$.

$$G_{Pos_x} = R_1||R_2 \quad (4.17)$$

$$H_{Pos_x} = G_{Pos_x}||Res_{Pos_x} \quad (4.18)$$

$$S_{Pos_x} = SupC(E_{Pos_x}) \quad (4.19)$$

Os resultados obtidos dos 15 supervisores modulares locais descritos anteriormente estão apresentados na tabela 4.2 em relação ao número de estados.

i	G_i	R_i	H_i	S_i	Sr_i
$P7\_R1$	11	2	21	21	2
$P9\_R1$	11	2	21	21	2
$P7\_R2$	11	2	21	21	2
$P9\_R2$	11	2	21	21	3
Pos_1	121	2	120	120	23
Pos_2	121	2	120	120	12
Pos_3	121	2	120	120	13
Pos_4	121	2	120	120	12
Pos_5	121	2	120	120	12
Pos_6	121	2	120	120	12
Pos_7	121	2	120	120	12
Pos_9	121	2	120	120	12
Pos_{10}	121	2	120	120	12
Pos_{11}	121	2	120	120	12
Pos_{12}	121	2	120	120	12

Tabela 4.2: Número de estados dos modelos para o caso centralizado.

Após a síntese destes supervisores, foi realizado o teste de não-conflito entre eles, no qual, para este sistema, obtivemos êxito em relação ao teste. Isto permite dizer que os supervisores não são conflitantes entre si e a implementação destes pode ser realizada localmente de forma segura. Portanto, pode-se concluir que para o caso centra-

lizado, o controle modular fornece autômatos menores do que no caso monolítico, maior facilidade de desenvolvimento e menor complexidade computacional devido a explosão de estados. De acordo com o Teorema 4.3, foi realizado o produto síncrono dos supervisores modulares, e como o esperado, o resultado forneceu o supervisor centralizado monolítico.

4.3.2 Coordenador Distribuído para o caso de 2 Robôs

Conforme discutido no caso do controle monolítico distribuído, o conceito da abordagem modular local pôde ser utilizada para o uso distribuído dos controladores, evidenciando uma nova aplicação para tal. Desta forma, será utilizado neste caso a abordagem modular local da mesma forma que no caso anterior juntamente com o modo que foi aplicado no controle monolítico distribuído.

4.3.2.1 Planta

Para o cálculo do supervisor distribuído modular, será utilizada a mesma modelagem dos robôs que no caso monolítico distribuído, sendo R_1 representado pela Figura 4.10 e R_2 representado pela Figura 4.19, que difere do modelo da Figura 4.15 pelo fato de todos os eventos serem não controlados. Como no caso monolítico distribuído calcula-se inicialmente o controlador que se refere ao robô R_1, sendo seus eventos controláveis, enquanto o robô R_2 apenas se encontra no ambiente de trabalho como se fosse um obstáculo em movimento, sendo seus eventos considerados não controláveis. Quanto aos modelos da visualização das portas pelos robôs também são os mesmos do caso monolítico distribuído, conforme as Figuras 4.11(a) e 4.11(b). Entretanto, a composição destes é realizada de uma forma diferente, que dependerá da síntese dos controladores modulares locais.

4.3.2.2 Especificações

As restrições de coordenação do robô entre as portas (Figuras 4.12(a) e 4.12(b)) e da ocupação das posições (4.20) serão as mesmas que no caso monolítico distribuído. A composição desses modelos dependerá também da síntese dos controladores modulares locais.

4.3.2.3 Síntese de Supervisores Modulares

A construção dos supervisores modulares segue o mesmo procedimento que para o caso modular centralizado, onde as equações são as mesmas, diferenciando apenas nos modelos que são utilizados. Desta maneira, aplicando as equações do caso modular centralizado com os modelos do caso monolítico distribuído, chega-se aos resultados obtidos para os 13 supervisores modulares, com exceção aos supervisores para as portas vistas pelo robô R_2, conforme descrito no capítulo anterior. A tabela 4.3 apresenta o número de estados destes supervisores modulares, no caso distribuído.

i	G_i	R_i	H_i	S_i	Sr_i
$P7\_R1$	11	2	21	21	2
$P9\_R1$	11	2	21	21	2
Pos_1	121	2	121	121	2
Pos_2	121	2	121	121	2
Pos_3	121	2	121	121	3
Pos_4	121	2	121	121	2
Pos_5	121	2	121	121	2
Pos_6	121	2	121	121	2
Pos_7	121	2	121	121	2
Pos_9	121	2	121	121	2
Pos_{10}	121	2	121	121	2
Pos_{11}	121	2	121	121	2
Pos_{12}	121	2	121	121	2

Tabela 4.3: Número de estados dos modelos para o caso distribuído.

Após a síntese destes supervisores, foi realizado com êxito o teste

de não-conflito entre eles. O que permite concluir que os supervisores não são conflitantes entre si e passíveis de serem implementados localmente, de forma segura. O controle modular local nos forneceu autômatos menores, maior facilidade de desenvolvimento e menor complexidade computacional. Em comparação ao caso modular centralizado, os autômatos dos supervisores locais ficaram menores ou iguais. Isto é, sobretudo visível no caso dos supervisores para as restrições de posição.

Os supervisores modulares para o robô R_2 também foram calculados e do mesmo modo que no caso monolítico distribuído, o conceito da abordagem modular local é utilizada para os supervisores de ambos robôs. Assim, o teste de não-conflito também foi realizado entre estes supervisores, indicando que não são conflitantes e que a composição dos supervisores dos robôs R_1 e R_2 garantem os objetivos globais desejados.

Para efeitos de comparação, o produto síncrono entre os 13 supervisores modulares do robô R_1 foi realizado e resultou em um autômato igual ao supervisor gerado para o caso monolítico distribuído. Além disso, também foi realizado o produto síncrono de todos os supervisores modulares de R_1 e R_2, resultando no supervisor gerado pelo caso monolítico centralizado.

4.4 Coordenação de SMR para Multíplos Robôs

Em todos os casos abordados por TCS, obteve-se um controlador para o sistema multi-robôs. Entretanto, a fim de analisar as limitações da metodologia, foi realizada uma expansão do estudo de caso. Considerando-se agora um espaço de trabalho com 20 células, 2 portas e 2 obstáculos fixos, foram sintetizados controladores para 1, 2 e 3 robôs nas abordagens Monolítica Centralizada e Modular Dis-

tribuída. A Tabela 4.4 apresenta o número de estados e transições dos controladores sintetizados pela abordagem Monolítica Centralizada.

Nº de Robôs	Nº de Estados	Nº de Transições
1	68	208
2	4360	25280
3	$> 100K$	$> 100K$

Tabela 4.4: Supervisores pela abordagem Monolítica Centralizada.

A Tabela 4.5 apresenta o número de estados dos modelos pela abordagem Modular Distribuída para os casos de 1, 2 e 3 robôs do ponto de vista de um robô. Os modelos são representados por G_i, a parte da planta a ser supervisionada; R_i o autômato de restrições; H_i, o autômato que leva à especificação final E_i para uma parte do sistema; S_i, o supervisor modular do cálculo de $SupC(E_i)$; e Sr_i, o supervisor reduzido de S_i. Sendo i a representação de cada configuração modular do sistema.

Vale ressaltar que a verificação de não bloqueio entre os supervisores de cada robô inserido no espaço de trabalho através do teste de não conflito deve ser necessariamente feita.

4.5 Discussões

Neste capítulo foi apresentada uma visão geral sobre a utilização da teoria de controle supervisório para coordenação de SMR tanto utilizando a abordagem monolítica quanto a abordagem modular local. Foram também projetados controladores em ambas as abordagens, cada um com duas topologias diferentes: centralizada e distribuída.

Serão destacados os aspectos que diferenciam as abordagens aqui citadas. O controle supervisório modular local propõe uma solução, cujo custo computacional é consideravelmente menor e o esforço de

Nº de Robôs	i	G_i	R_i	H_i	S_i	Sr_i
1	Porta 1	18	2	35	35	3
	Porta 2	18	2	35	35	2
2	Porta 1	18	2	35	35	3
	Porta 2	18	2	35	35	2
	Posição 1 2º Robô	324	2	324	324	2
	Posição 2 2º Robô	324	2	324	324	2
	Posição 3 2º Robô	324	2	324	324	3
	Posição 4 2º Robô	324	2	324	324	2
	Posição 5 2º Robô	324	2	324	324	2
	...	...	...	...	...	...
	Posição 20 2º Robô	324	2	324	324	2
3	Porta 1	18	2	35	35	3
	Porta 2	18	2	35	35	2
	Posição 1 2º Robô	324	2	324	324	2
	Posição 2 2º Robô	324	2	324	324	2
	Posição 3 2º Robô	324	2	324	324	3
	Posição 4 2º Robô	324	2	324	324	2
	Posição 5 2º Robô	324	2	324	324	2
	...	...	...	...	...	...
	Posição 20 2º Robô	324	2	324	324	2
	Posição 1 3º Robô	324	2	324	324	2
	Posição 2 3º Robô	324	2	324	324	3
	Posição 3 3º Robô	324	2	324	324	2
	Posição 4 3º Robô	324	2	324	324	2
	Posição 5 3º Robô	324	2	324	324	2
	...	...	...	...	...	...
	Posição 20 3º Robô	324	2	324	324	2

Tabela 4.5: Número de estados dos modelos pela abordagem Modular Distribuída.

implementação mais sistematizada e menos árdua que no caso monolítico. Em contrapartida, existe a possibilidade de bloqueio devido aos possíveis conflitos entre os supervisores locais o que pode se tornar um limitante para este método, entretanto técnicas de resolução de conflito (i.e. coordenador) permitem resolver este problema [46]. Neste trabalho, nenhum caso apresentou conflitos entre os supervisores, viabilizando assim a implementação destes.

Em relação aos controles centralizado e distribuído, percebemos diferenças no desenvolvimento do sistema. Quando se aborda o problema de forma distribuída, a modelagem se torna mais simples, pois, a preocupação foca-se apenas em um robô. Assim, os outros robôs existentes no espaço de trabalho fazem parte apenas da planta, sendo seus eventos não-controláveis. Ou seja, o controlador apenas receberá a informação de onde estão os outros robôs e nunca agirá sobre eles, embora no caso modular o robô também envia a informação de localização para os outros robôs.

Aplicando o controle distribuído para todos os robôs, eles irão se comportar, de forma global, da mesma maneira caso houvesse um controlador centralizado, garantindo as propriedades desejadas para o sistema. Isto provem do fato de que o controlador distribuído é desenvolvido com o conceito da abordagem modular local, no qual permite explorar, além da modularidade das especificações, a estrutura naturalmente distribuída deste sistema. Vale ressaltar que mesmo para o caso monolítico distribuído, a ideia da abordagem modular local é utilizada. Pois, embora o modelo seja obtido pela abordagem monolítica, globalmente ele se comporta de forma semelhante a abordagem modular local. Desta maneira, com o êxito no teste de não-conflito entre os supervisores distribuídos, garante-se as propriedades desejadas.

Além disso, uma diminuição considerável no número de estados do supervisor quando se utiliza a abordagem modular local é observada. Esta característica se torna muito importante no caso distribuído, onde o controlador é embarcado no robô. As restrições quanto à capacidade

de memória podem ser limitantes para aplicação do controle do tipo monolítico distribuído. Nota-se também que a partir do produto síncrono dos supervisores modulares distribuídos de todos os robôs pode-se chegar aos mesmos supervisores que todos os outros casos desenvolvidos nesta dissertação: monolítico distribuído, modular centralizado e monolítico centralizado.

Por fim, foi obtido em todos os casos coordenadores para que os robôs atendam as propriedades de segurança e alcançabilidade, i.e. para que aos robôs não colidam com os obstáculos e entre si; e atinjam os objetivos pré-definidos. Porém, devido à característica do controle supervisório de ser o mais permissível possível, implicando que cada robô tenha mais de uma possibilidade de caminho a cada posição, uma lógica adicional na implementação deve ser introduzida a fim de permitir aos robôs de atingir seus objetivos de forma otimizada, por exemplo o caminho mais curto. Esta estratégia ajudará o robô a escolher um transição dentre as que são possíveis em cada momento e será melhor discutida no Capítulo 6.

4.6 Conclusão

Conclui-se neste capítulo que a Teoria de Controle Supervisório, juntamente com sua extensão modular local é capaz de resolver o problema de coordenação em sistemas multi-robôs. Entretanto, essa abordagem introduz um nível adicional de tomada de decisão. No próximo capítulo, será apresentado um outro método que dá um poder maior de representação do modelo.

Capítulo 5

Coordenação de SMR utilizando Autômato-Jogo Temporizado

Como já visto no capítulo anterior, a utilização de formalismos para a coordenação e controle de SMR é de suma importância devido a complexidade destes sistemas. Todavia, os métodos utilizados até então nesta dissertação não consideram o tempo em seu formalismo. A partir deste cenário, propõe-se a utilização de um método, que através de um jogo, leva em consideração o tempo em sua modelagem garantindo as propriedades de segurança e alcançabilidade e sua posterior comparação com TCS.

Neste capítulo é apresentado o método de síntese de controladores discretos para sistemas temporizados através de Autômato-Jogo Temporizado (AJT), proposto por [32]. Inicialmente, é apresentado o

método utilizado para a síntese da estratégia do controlador, o algoritmo utilizado para a síntese, e por fim, o desenvolvimento para as duas topologias diferentes: *centralizada* e *distribuída*.

5.1 Jogos Temporizados

A teoria dos jogos é o estudo da tomada de decisão sob competição. Mais especificamente, a teoria de jogos é o estudo da tomada de decisão ótima sob competição quando a decisão de um jogador afeta o resultado de uma situação para todos os outros jogadores envolvidos[24]. Nesta dissertação, o jogo será uma considerado como uma competição entre o coordenador (controlador) e o ambiente, cujos jogadores estão inseridos em um sistema composto por robôs controláveis e por portas não controláveis, ambos situados dentro de um espaço de trabalho. O objetivo geral é encontrar uma estratégia para o jogador *coordenador* que quaisquer que sejam as ações do jogador *portas* o coordenador vença. Diga-se que uma estratégia de um jogador é vencedora quando ela sempre garantir que este jogador atinja seus objetivos independentemente das ações de seus adversários. O Exemplo 5.1 elucida a definição de um jogo, retirado de Maler *et al.* (1995) [32].

Exemplo 5.1 (Jogo de Perseguição)**.** *Considera-se o jogo apresentado pela Figura 5.1. Um jogador, definido como $J1$, representado pelo círculo a esquerda da figura, inicia movimentando-se (running) a partir da posição inicial. Neste ponto, ele gasta no mínimo 3 segundos para chegar a junção (junction). Na junção, o jogador $J1$ pode ou esperar ou escolher um caminho, direita (run-right) ou esquerda (run-left). Após ter decidido o caminho, ele pode atingir a ponte em 5 segundos. Caso ela não esteja bloqueada, o jogador poderá atingir a posição final (end) em 3 segundos. O jogador $J2$, representado pelo quadrado no centro da figura, inicia o jogo localizado no centro (center) entre as pontes da direita (right-bridge) e da esquerda (left-bridge). Dentro de um intervalo*

de 8 segundos, ele deve escolher uma posição para atingir, ou a ponte da esquerda ou da direita, e assim, bloquear a ponte em questão.

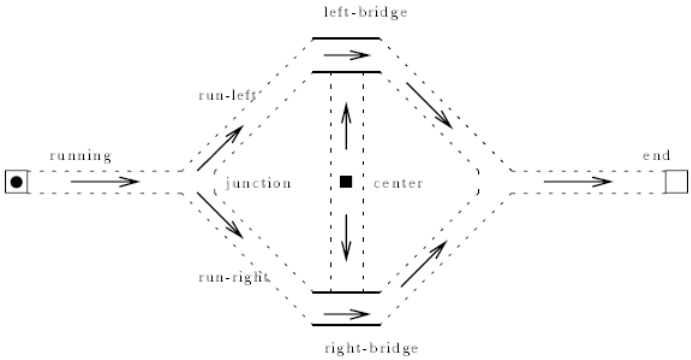


Figura 5.1: Um jogo de perseguição.

Para o jogador $J1$ conseguir a vitória, ele deverá atingir a posição final (end) sem que o jogador $J2$ bloqueie sua passagem. Desta maneira, uma estratégia para o jogador $J1$ sempre atingir a posição final (end) independentemente de $J2$ deve ser sintetizada para garantir a sua vitória.

Portanto, neste caso desta dissertação, o princípio de funcionamento, que permitirá definir esta estratégia, é baseado em um jogo, no qual o controlador dos robôs e o ambiente são adversários. O ambiente é considerado nesse trabalho como o conjunto de portas existentes no espaço de trabalho. O coordenador do robô é sintetizado através da definição de condições de aceitação que a estratégia deve garantir. Baseadas em propriedades de segurança e alcançabilidade, os obstáculos fixos e móveis e a exclusão de colisão entre robôs deverão ser levados em conta nas condições de aceitação.

O jogo evolui da seguinte forma, cada jogador (controlador e ambiente) tem sua vez de jogar, podendo escolher uma ação ou deixar o tempo passar. Desta maneira, é possível obter uma estratégia dita

vencedora para o controlador quando as condições de aceitação forem verificadas independentemente das ações do ambiente.

5.2 Definições para Jogos Temporizados

A inserção do tempo nos Sistemas a Eventos Discretos, conforme apresentado no Capítulo 3, proporciona um maior poder de representação destes sistemas e, por consequência, a possibilidade de sintetizar controladores discretos mais elaborados. Um destes métodos é o uso de Jogos Temporizados baseados em Autômato-Jogo Temporizado (AJT), proposto inicialmente por Maler *et al* (1995) [32]. Nesta seção, será apresentado a definição de AJT e posteriormente as possíveis expressões lógicas utilizadas para sintetizar estratégias utilizando este método.

5.2.1 Sistema Autômato-Jogo Temporizado

O Autômato-Jogo Temporizado (AJT), proposto por Maler *et al.* (1995) e Asarin *et al.* (1998) [32, 36], foi introduzido para resolver problemas de controle em sistemas temporizados. Adaptando o autômato temporizado (AT), já apresentado na Seção 3.4.2, para configurações de um jogo, chega-se a noção do AJT, conforme a Definição 5.1.

Definição 5.1. *Um Autômato-Jogo Temporizado (AJT) é um autômato temporizado [3], cujo conjunto de ações* $\Sigma = \Sigma_c \cup \Sigma_u$ *é particionado em ações controláveis* (Σ_c) *e ações não controláveis* (Σ_u).

Um AJT pode ser composto a partir de n AJTs $\mathcal{A}_i = (L_i, l_0^i, \Sigma, X, I_i, R_i)$ onde o conjunto de ações sobre a rede é dado por $\overline{\Sigma} = \Sigma \times \Sigma$, tal que $\overline{\Sigma} = \overline{\Sigma_c} \cup \overline{\Sigma_u}$ onde $\overline{\Sigma_c} = \Sigma_c \times \{\Sigma \cup \{a\}\}$ e $\overline{\Sigma_u} = \overline{\Sigma} \backslash \overline{\Sigma_c}$.

Em AJT, uma transição é disparada ou pelo jogador principal, através de ações controláveis (Σ_c), ou pelo jogador oponente, através

de ações não controláveis (Σ_u). Em um contexto teórico de jogo, a modelagem de um sistema pode ser realizada por um AJT. Se um controlador puder ser sintetizado para um objetivo de controle particular, então sob sua supervisão, o sistema AJT irá lidar apenas com bons caminhos do sistema.

5.2.2 Especificações em Lógica Temporal

Para a síntese de uma estratégia em AJT, utiliza-se expressões em lógica temporal para a obtenção das especificações desejadas para o sistema. No Anexo A é apresentado uma visão geral sobre lógica temporal baseada em vários trabalhos [16, 20, 37, 44]. Neste trabalho, utiliza-se uma versão simplificada de TCTL [2], uma extensão de CTL com a inclusão do tempo, para a definição de fórmulas para a alcançabilidade e segurança. Igualmente à TCTL, a linguagem utilizada consiste de fórmulas ,de caminho e fórmulas de estado [7]. As fórmulas de estado descrevem estados individuais, enquanto fórmulas de caminho quantificam sobre os caminhos do modelo. Fórmulas podem ser classificadas em alcançabilidade, segurança e vivacidade. Nesta dissertação, apenas alcançabilidade e segurança serão representadas, conforme descritas abaixo.

Alcançabilidade Dada uma fórmula de estado φ, a propriedade de alcançabilidade é garantida se for possível satisfazê-la para qualquer estado alcançável. Ou seja, existe um caminho, iniciado do estado inicial, tal que φ é eventualmente satisfeito.

Segurança Propriedades de segurança são garantidas quando nada de indesejável para o sistema ocorra. Desta maneira, dada uma fórmula de estado φ, o propriedade é garantida se $\neg\varphi$ for sempre satisfeita.

Os operadores utilizados para ambas propriedades estão apresentados na Tabela 5.1.

Em CTL, assim como os operadores $\Diamond$ (algum estado futuro), $\Box$ (todos os estados futuros (Globalmente)), X (próximo estado), U (*Until*) e W (Weak-Until) de LTL, têm-se também quantificadores A e E que expressam "todos os caminhos"e "existe um caminho", respectivamente [25]. As fórmulas de caminho suportadas pela versão utilizada nesta dissertação são apresentadas na Tabela 5.1.

Propriedade	Descrição
$E\Diamond\phi$	Possibilidade: Existe um caminho onde ϕ eventualmente é satisfeita.
$E\Box\phi$	Potencialmente Sempre: Existe um caminho onde ϕ é sempre satisfeita.
$A\Diamond\phi$	Eventualmente: Para todos os caminhos ϕ eventualmente é satisfeita.
$A\Box\phi$	Invariante (Sempre): Para todos os caminhos ϕ é sempre satisfeita
$A[\varphi U\phi]$	Invariante (Sempre): Para todos os caminhos φ é sempre satisfeita até que ϕ seja satisfeita
$A[\varphi W\phi]$	Invariante (Sempre): Para todos os caminhos φ é sempre satisfeita, não sendo necessário que ϕ seja eventualmente satisfeita

Tabela 5.1: Expressões lógicas em TCTL.

5.3 Síntese de Controladores baseada em Autômato-Jogo Temporizado

Para uma perspectiva de engenharia de controle, uma aplicação embarcada consiste de uma planta a ser controlada e de um controlador. A planta, definida como um sistema dinâmico P, descreve todas as possibilidades de comportamento. Dentre estas possibilidades estão os comportamentos desejados e indesejados para o sistema. O controlador, definido como um sistema C, interage com P observando seus estados e emitindo ações de controle que influenciam o comportamento de P,

de maneira que, apenas ocorra um comportamento desejado para o sistema.

A planta e o controlador são definidos separadamente. A planta, modelada com uma rede de AJTs, possui ações dos robôs, controlados pelo controlador (jogador), e das portas, consideradas ações não controláveis do ambiente (adversário). Já o controlador é um autômato coordenador que é sintetizado para interagir com a planta através das ações dos robôs e, desta maneira, levá-los aos seus objetivos, i.e garantir a vitória do controlador. As trajetórias deste sistema são definidas conforme a Definição 5.2.

Definição 5.2. *Sendo P uma planta, C um controlador, δ uma relação de transição, $x \in X$ um valor real de clock e $a \in \Sigma$ uma ação discreta, uma sequência infinita de estados $\alpha : s[0], s[1], ...$ tal que $s[0] = s_0$ é chamada de P-trajetória se*

$$s[i+1] \in \bigcup_{(a \vee x) \in \Sigma} \delta(s[i], (a \vee x)) \tag{5.1}$$

e C-trajetória se $s[i+1] \in \delta(s[i], C(\alpha[0...i]))$ para todo $i \geq 0$.

As trajetórias *P-trajetória* e *C-trajetória* são denotadas por $L(P)$ e $L_c(P)$, respectivamente. Cada *C-trajetória* é uma trajetória do sistema, sendo $L_c(P) \subseteq L(P)$. Para toda trajetória infinita $\alpha \in L(P)$, denota-se $Vis(\alpha)$ o conjunto de todos os estados que aparecem em α e $Inf(\alpha)$ o conjunto de estados que aparecem em α infinitamente muitas vezes.

5.3.1 Metodologia para a síntese da estratégia vencedora

O método por AJT consiste então em encontrar uma estratégia que possibilite garantir comportamentos desejados para o sistema P a

ser controlado. Esta estratégia consiste em definir regras que permitam ao controlador C, escolher uma dentre as várias possíveis ações. A definição desta estratégia é baseada em um jogo, no qual o controlador e o ambiente são adversários. O controlador é sintetizado a partir da definição de condições de aceitação que a estratégia deve garantir.

O jogo evolui da seguinte forma, cada jogador (controlador e ambiente) tem sua vez de jogar, podendo escolher uma ação ou deixar o tempo passar. Desta maneira, é possível obter uma estratégia dita vencedora para o controlador quando as condições de aceitação forem verificadas apesar das ações do ambiente. O jogo é descrito por uma rede de AJTs especificando as regras do jogo e por uma fórmula temporal (φ), que representam as condições de aceitação e especificam as condições de vitória para o controlador.

Os passos necessários até a obtenção da estratégia via AJT se resumem nas seguintes etapas:

1. Modelagem do sistema, através de uma rede de Autômato-Jogo Temporizado;

2. Modelagem das condições de aceitação para uma estratégia vencedora;

3. Síntese do controlador que permite obter a estratégia vencedora do jogo.

5.3.1.1 Modelagem

Inicialmente é realizada a definição da rede de AJTs que irão fazer parte do sistema a ser controlado. Esta rede, representando a planta P, conterá todos os possíveis comportamentos para o sistema e, desta maneira, especificando as regras do jogo.

Exemplo 5.2. *Continuando o Exemplo 5.1, segue-se os passos para a obtenção de uma estratégia para o J1. Para o primeiro passo, deve-se realizar a modelagem do sistema, composto pelos movimentos de J1, representando o jogador a ser controlado, e J2, representando o ambiente (adversário). Para a definição da planta deste sistema dois AJTs são necessários, um representando o jogador J1 e outro o jogador J2. A Figura 5.2 apresenta a rede de AJTs que definem as regras do jogo. Nota-se que os eventos de J1 são controláveis, enquanto o de J2 são não controláveis.*

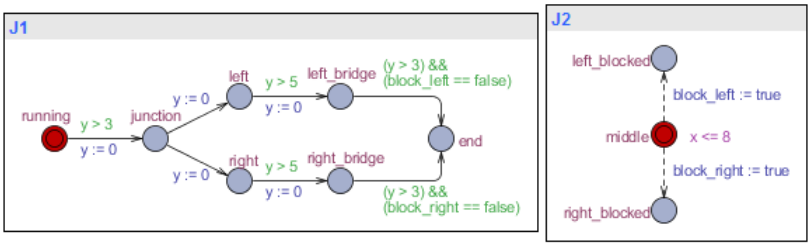


Figura 5.2: A descrição do jogo.

5.3.1.2 Condições de Aceitação

Definido um sistema $\mathcal{T}$ a ser controlado, um conjunto de condições de aceitação Ω deve ser selecionado para que se definam os estados do sistema a serem evitados e/ou atingidos. A partir deste conjunto, estabelecem-se os objetivos do controlador. Sendo $\alpha \in L(P)$ uma trajetória infinita, $Vis(\alpha)$ um conjunto de todos os estados que aparecem em α e $Inf(\alpha)$ o conjunto de estados que aparecem em α infinitamente muitas vezes, tem-se F como o conjunto de estados a serem atingidos/evitados, ou seja, os estados que representam os objetivos do controlador. Desta maneira, define-se um conjunto de condições de aceitação consoante a equação

$$\Omega \in \{(F, \Diamond), (F, \Box), (F, \Diamond\Box), (F, \Box\Diamond)\} \tag{5.2}$$

$L(\mathcal{T}, F, \Box)$	$\alpha \in L(\mathcal{T}) : Vis(\alpha) \subseteq F$	α sempre permanece em F
$L(\mathcal{T}, F, \Diamond)$	$\alpha \in L(\mathcal{T}) : Vis(\alpha) \cap F \neq \emptyset$	α eventualmente visita F
$L(\mathcal{T}, F, \Diamond\Box)$	$\alpha \in L(\mathcal{T}) : Inf(\alpha) \subseteq F$	α eventualmente permanece em F
$L(\mathcal{T}, F, \Box\Diamond)$	$\alpha \in L(\mathcal{T}) : Inf(\alpha) \cap F \neq \emptyset$	α visita F *infinitely often*

Tabela 5.2: Condições de aceitação para $\mathcal{T}$.

onde as sequências de $\mathcal{T}$ de acordo com $\omega \in \Omega$ são definidas conforme a Tabela 5.2.

Exemplo 5.3. *Continuando o Exemplo 5.1, deve-se realizar o segundo passo para a obtenção da estratégia. Desta forma, deseja-se que o jogador $J1$ atinja o fim do trajeto, não sendo bloqueado por alguma porta. Logo, o conjunto de condições de aceitação a ser utilizado neste caso é definido como*

$$\Omega = \{(F, \Diamond)\}, \tag{5.3}$$

no qual deseja-se que a propriedade de alcançabilidade seja verificada. Neste exemplo, o conjunto dos estados a serem atingidos F é composto de todos os estados que contenham o lugar end do modelo de $J1$.

Sendo $\bar{l}$ o vetor de lugares do sistema, no qual a primeira posição se refere ao lugar de $J1$ e a segunda de $J2$ e $i = \{1, .., n_{J2}\}$, onde n_{J2} é o número total de lugares de $J2$, tem-se

$$F = \{\bigcup_{i=1}^{n_{J2}} (\bar{l}(i), \mathbb{R}_{\geq 0})\} \tag{5.4}$$

onde $\bar{l}(i) = [end \quad l_{J2}(i)]$ e $l_{J2}(i)$ se refere ao i-enésimo lugar de $J2$.

5.3.1.3 Obtenção do Controlador

Para um sistema $\mathcal{T}$ e um conjunto de condição de aceitação Ω, o problema de síntese do controlador, $Synth(\mathcal{T}, \Omega)$, é encontrar um controlador C tal que $L_C(\mathcal{T}) \subseteq L(\mathcal{T}, \Omega)$, sendo $L_C(\mathcal{T})$ um conjunto de *C-trajetórias* de $\mathcal{T}$, que são geradas a partir das ações do controlador. Caso a resposta de $Synth(\mathcal{T}, \Omega)$ seja positiva no ponto de vista da possibilidade de obter um controlador, diz-se que $(\mathcal{T}, \Omega)$ é controlável [36].

Os primeiros algoritmos utilizados para a realização da síntese, propostos por Maler *et al.* (1995) e Asarin *et al.* (1998) [32, 36], são baseados no cálculo retroativo de equações de ponto-fixo para encontrar o conjunto dos estados vencedores W. Este conjunto é composto pelos estados que, de acordo com a condição de aceitação, o controlador poderá levar ao bom comportamento do sistema. Portanto, a estratégia desejada para o sistema $\mathcal{T}$ é gerada a partir do conjunto dos estados vencedores W.

Através de um operador $\pi : 2^S \mapsto 2^S$, mapeia-se um conjunto de estados $K \subseteq S$ para o conjunto de seus *predecessores controláveis*, i.e., o conjunto de estados a partir do qual o controlador pode levar a planta para K em um passo [32].

$$\pi(K) = \{s : \exists a \in \Sigma_c, \exists d \in \mathbb{R}_{\geq 0}, \delta(s, (a \vee d)) \subseteq K\}. \tag{5.5}$$

O cálculo dos *predecessores controláveis* é a essência dos algoritmos de síntese e é realizado através da iteração de equações de ponto-fixo[1], consoante equações

[1] μ operador mínimo ponto-fixo, υ operador máximo ponto-fixo

$$\Box \quad : \quad \upsilon W(F \cap \pi(W)) \tag{5.6}$$

$$\Diamond \quad : \quad \mu W(F \cup \pi(W)) \tag{5.7}$$

$$\Diamond\Box \quad : \quad \mu W \upsilon H(\pi(H) \cap (F \cup \pi(W))) \tag{5.8}$$

$$\Box\Diamond \quad : \quad \upsilon W \mu H(\pi(H) \cup (F \cap \pi(W))) \tag{5.9}$$

onde o sistema é considerado controlável se e somente se $s_0 \in W$. Cada equação indica uma propriedade específica que o controlador pode garantir para o sistema. Por exemplo, as Equações 5.6 e 5.7 representam as propriedades de segurança e alcançabilidade, respectivamente.

Para o caso de alcançabilidade, a Equação 5.7 é calculada em um processo iterativo dado por $W^0 = F$ e $W^{n+1} = W^n \cup \pi(W^n)$. Essa iteração converge após muitos passos [32], onde W é o máximo conjunto dos estados vencedores.

Apesar destes algoritmos encontrarem uma estratégia que garante as condições de aceitação, sua iteração pode ser infinita ou despender muito tempo para ser realizada dependendo do AJT a ser utilizado, possuindo assim, uma baixa eficiência computacional. Além disso, se for necessário utilizar mais de uma propriedade em uma única estratégia, por exemplo segurança e alcançabilidade, os algoritmos em questão não a sintetizam, pois apenas uma condição de aceitação é possível. Ao visto destas limitações, uma extensão desses algoritmos foi utilizada.

5.3.1.4 Algoritmos On-the-Fly para síntese de controladores

Para sistemas de estados finitos, os algoritmos de *model-checking on-the-fly* têm sido muito utilizados desde o fim dos anos 80 [13]. Dentre estes algoritmos, destaca-se por sua eficiência o algoritmo proposto

por Liu e Smolka (1998) [31]. Em contraste com os algoritmos baseados no cálculo retroativo de equações de ponto-fixo, como citado anteriormente, os algoritmos *on-the-fly* utilizados para *model-checking* de modelos de AT fazem uma exploração progressiva do espaço de estados simbólicos resultando no chamado grafo de simulação. O grafo de simulação é baseado na exploração de um grafo finito, onde os nodos são estados simbólicos. Entretanto, ele é muito abstrato para ser utilizado como base de um algoritmo *on-the-fly* para calcular estratégias vencedoras.

O algoritmo proposto por Cassez *et al.* (2005,2007) [12, 13] é uma extensão do algoritmo de Liu e Smolka que utiliza de um procedimento *on-the-fly* e progressivo, baseado em zonas, para solucionar condições de segurança e alcançabilidade em jogos temporizados. Este algoritmo pode ser visto como uma combinação de computação progressiva do grafo de simulação de um AJT juntamente com retropropagação [32, 36] das informações dos estados vencedores. Devido a essas características, este algoritmo simbólico pode ter sua parada antes mesmo de explorar todo o espaço de estados, ou seja, assim que uma estratégia vencedora for identificada.

O algoritmo é baseado sobre dois conjuntos de transições $ToExplore$ e $ToBackPropagate$, que armazenam arestas simbólicas do grafo de simulação; uma lista, $Passed$, que contém todos os estados simbólicos do grafo de simulação encontrados até o momento pelo algoritmo; e uma lista de espera, $Waiting$, de arestas do grafo de simulação a serem exploradas. Além disso, o conjunto $Win[Z] \subseteq Z$ identifica em um subconjunto do estado simbólico Z os estados que estão ganhando e o conjunto $Depend[Z]$ indica o conjunto das arestas que devem ser reavaliadas, i.e. adicionadas a lista $Waiting$, quando uma nova informação sobre $Win[Z]$ é obtida. O funcionamento do algoritmo simbólico *on-the-fly* é apresentado pelo Anexo B e tem sua corretude provados pelo Lemma 5.1 e Teorema 5.1, todos apresentados por Cassez *et al.* (2005) [13] e Cassez (2007) [12].

Lemma 5.1. *O ciclo do algoritmo proposto por Cassez et al. (2005) possui as seguintes propriedades de invariância quando executado sobre um AJT $\mathcal{T}$:*

1. *Para qualquer $Z \in Passed$ se $Z \xrightarrow{a} Z'$ então ou $(Z, a, Z') \in Waiting$ ou $Z' \in Passed$ e $(Z, a, Z') \in Depend[Z']$;*
2. *Se para algum $Z \in Passed$, $s \in Win[Z]$ então $s \in W$, onde W é o conjunto dos estados vencedores;*
3. *Se $s \in Z \backslash Win[Z]$ para algum $Z \in Passed$ então ou*
   - *$e \in Waiting$ para algum $e = (Z, a, Z')$ com $Z' \in Passed$, ou*
   - *$s \notin \pi(Win[Z])$,*

onde $\pi(Win[Z])$ depende da propriedade desejada. Neste algoritmo apenas as propriedades de segurança e alcançabilidade são tratadas.

Teorema 5.1. *Após o término da execução do algoritmo sobre um determinado AJT $\mathcal{T}$, tem-se:*

1. *Se $s \in Win[Z]$ para algum $Z \in Passed$ então $s \in W$;*
2. *Se $(ToExplore \cup ToBackPropagate) = \emptyset$,$s \in Z \backslash Win[Z]$ para algum $Z \in Passed$ então $s \notin W$.*

Por fim, para se obter a estratégia do controlador segundo as condições de aceitação desejadas, utiliza-se expressões lógicas em TCTL para realizar a síntese de acordo com os algoritmos simbólicos *on-the-fly* para jogos temporizados. Existem 4 condições de vitória que podem ser utilizadas:

- $A\Diamond\phi$ (Deve eventualmente atingir ϕ),

- $A[not(\varphi)U\phi]$ (Deve evitar φ e deve atingir ϕ),
- $A[not(\varphi)W\phi]$ (Deve evitar φ e poderá atingir ϕ),
- $A\Box not(\varphi)$ (Deve sempre evitar φ).

Portanto, tendo definido a planta $\mathcal{T}$ a ser controlada e o conjunto de condições de aceitação Ω juntamente com os conjuntos dos estados a serem evitados/atingidos, define-se a expressão lógica adequada para realizar a síntese do controlador.

Exemplo 5.4. *Seguindo o Exemplo 5.1, têm-se já definidos a planta $\mathcal{T}$, representada pela Figura 5.2 e o conjunto de aceitação Ω, representado pela Equação 5.3, onde o conjunto de estados a serem atingidos F é apresentado pela Equação 5.4. Portanto, o problema de síntese $Synth(\mathcal{T}, \Omega)$ foi definido a partir da expressão lógica $A\Diamond F$.*

O resultado foi obtido com êxito, garantindo que, para este exemplo, o problema $Synth(\mathcal{T}, \Omega)$ é controlável. A Figura 5.3 ilustra um AT representando o controlador gerado. A nomeação dos estados é feita da seguinte maneira: X_Y, onde X denota o lugar de $J1$ e Y denota o lugar de $J2$. Ambos, X e Y são apresentados com a primeira letra do nome do lugar dos autômatos da Figura 5.2. Por exemplo, R para running e LB para left-blocked.

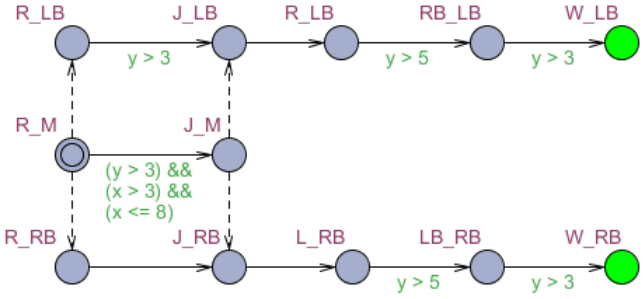


Figura 5.3: Autômato para uma estratégia vencedora.

Conforme já citado anteriormente, a síntese de uma estratégia vencedora pode não ser realizada com êxito, pois a vitória pode depender de ações não controláveis. Caso haja esta dependência, é impossível garantir a vitória ao controlador, pois o ambiente, responsável pelas ações não controláveis, pode nunca realizar as ações necessárias para o controlador garantir a vitória.

Para corrigir este problema, uma estratégia cooperativa é proposta por David *et al.* [19]. O principio intuitivo desta solução consiste em considerar algumas ações do ambiente como sendo necessárias para a garantia de uma estratégia vencedora. O jogo com a estratégia cooperativa é realizado com a divisão do espaço de estados em três áreas: (1) A zona de vitória, onde sempre o objetivo é atingido; (2) zona de possível vitória, onde o sistema ainda pode atingir o objetivo; e (3) zona de derrota, onde é impossível atingir o objetivo. Sendo assim, se o oponente estiver disposto a cooperar, então pode-se atingir um estado a partir do qual o objetivo é sempre eventualmente atingível (zona de possível vitória). A utilização deste método se faz acrescentando na fórmula em que se deseja obter a estratégia vencedora o termo $E\Diamond$, por exemplo $E\Diamond A\Diamond\phi$.

A estratégia gerada por este método é então dividida em duas partes, estratégia cooperativa e estratégia vencedora. A primeira depende de ações do ambiente necessárias para evoluir no jogo, com o objetivo de levar o sistema à zona de vitória. Caso estas ações nunca ocorram, jamais o sistema atingirá o objetivo. Porém, se as ações não controláveis desejadas ocorrerem, o sistema entra na zona de vitória, onde a estratégia do controle será outra, a estratégia vencedora. Esta estratégia é a mesma gerada pelo algoritmo de Cassez *et al.* (2005,2007) citado anteriormente, cujo objetivo é atingir a vitória do controlador, i.e., garantir as condições de aceitação.

5.4 Projeto de Coordenador de Sistemas Multi-Robôs pela abordagem AJT

Considerando o estudo de caso apresentado na Seção 2.3.1, será apresentado o projeto de um coordenador para SMR utilizando a abordagem AJT para as duas topologias sugeridas: *centralizada* e *distribuída*. Como no projeto por TCS, em um primeiro momento, será apresentado o desenvolvimento do plano de tarefas para apenas 1 robô R_1 dentro do espaço de trabalho. O plano a ser gerado deve levar o robô a atingir a posição 12 e evitar a colisão com as portas e o obstáculo fixo, garantindo assim as propriedades de alcançabilidade e segurança.

Para a síntese dos controladores foi utilizado uma ferramenta para solucionar jogos temporizados chamada UPPAAL-TIGA[2] [8], que é uma extensão da ferramenta UPPAAL[3] [7].

5.4.1 UPPAAL-TIGA

UPPAAL-TIGA é uma ferramenta para solucionar jogos baseado em Autômatos-Jogo Temporizado [8]. TIGA implementa os algoritmos *on-the-fly* [12, 13] para sintetizar estratégias vencedoras, assim como algumas extensões, como a estratégia cooperativa [19]. A linguagem de entrada é dada por uma rede de AJT (Seção 5.2.1), onde as transições podem ser definidas como controláveis e não controláveis. As condições de vitória do jogo são definidas por fórmulas em TCTL 5.1, cuja as possibilidades de propriedades são as de segurança e alcançabilidade.

5.4.2 Coordenador para o caso de 1 Robô

O sistema para este caso possui um problema mais simples pelo motivo de não possuir problemas de colisão entre robôs. Este estudo de

[2] http://www.cs.aau.dk/ adavid/tiga/
[3] http://www.uppaal.org/

caso simplificado tem como objetivo apresentar o método a ser aplicado.

5.4.2.1 Modelagem do Sistema

O sistema a ser controlado $\mathcal{A}$ é composto de uma rede de AJTs contendo o modelo do robô, cujas ações são controláveis e os modelos da visualização das portas pelos robôs,cujas ações são não controláveis. Podemos notar que o alfabeto de entrada Σ é formado pela composição das ações controláveis dos robôs Σ^R com as ações não-controláveis das portas Σ^B. O controlador a ser sintetizado (jogador) buscará a vitória levando o robô à posição desejada, enquanto o ambiente (adversário), representado pelas portas, poderá se movimentar a qualquer momento.

Para a modelagem do sistema, uma rede de AJTs $\mathcal{A}$ composta por dois AJT é necessária. Os modelos que compõem o sistema são o modelo do robô R_1, denominados $Robot(i)$; e os modelos da visualização da porta pelos robôs, onde se encontram as ações do ambiente, denominados $Barrier(id)$, onde $i \in \{1, ..., N_r\}$ representa o índice de identificação dos robôs sendo N_r o número total de robôs e $id \in \{1, ..., N_b\}$ o índice de identificação das portas, com N_b sendo o número total de portas. Igualmente ao projeto por TCS, as portas P_7 e P_9, presentes nas posições 7 e 9, foram modeladas conforme são vistas pelo robô. Vale ressaltar que, a exemplo do Capítulo anterior, nunca haverá inconsistência entre o robô e a porta quando o robô estiver na vizinhança da porta, i.e. enquanto o robô estiver em sua vizinhança, as informações que o robô tem da porta correspondem de fato ao estado da porta naquele momento. Como o caso a ser tratado possui um robô e duas portas, têm-se $N_r = 1$ e $N_b = 2$. Portanto, as portas são identificadas como $Barrier(1)$ para a da posição 7 e $Barrier(2)$ para a da posição 9 enquanto o robô R_1 é identificado como $Robot(1)$ e tem sua posição inicial na célula 1.

Para se definir um *lugar* onde se encontra um robô, utiliza-se o nome do modelo do robô seguido de um ponto e do nome do *lugar*, por

exemplo $Robot(1).Pos04$, significa que o robô R_1 está na posição 4 do espaço de trabalho.

A modelagem de um robô é realizada através de um AJT. O modelo desenvolvido considera a movimentação livre do robô pelo espaço de trabalho, onde os *lugares* representam as posições, por exemplo $Pos01$ (posição 1). O modelo possui um *lugar* inicial, correspondente a sua posição de partida e as ações de mudança de posição, cujas ações são controláveis $\Sigma^R = \Sigma_c$. A Figura 5.4 apresenta o modelo do robô, onde estão representadas as possíveis células de localização do robô, nas quais se encontram o obstáculo fixo, as portas e o restante das células.

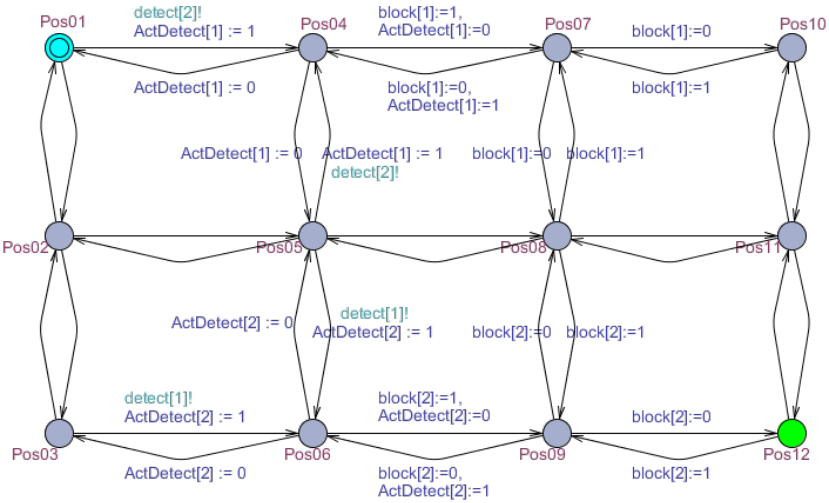


Figura 5.4: Modelo do robô no espaço de trabalho ($Robot(i)$).

O modelo da Figura 5.5 apresenta a visualização da porta pelos robôs. Ele possui apenas 2 *lugares*, um representando a porta aberta (PA) ou não visível pelos robôs, também sendo o *lugar* inicial, e outro a porta fechada (PF). Todas as ações deste modelo são não controláveis, isto é, para este modelo $\Sigma^B = \Sigma_u$.

Neste modelo, para representar as restrições dos modelos, 3 variáveis são utilizadas: (1) $block[id]$, variável que determina que a porta

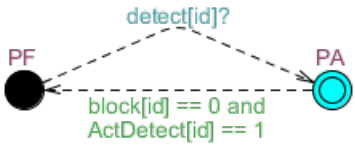


Figura 5.5: Modelo da visualização da porta pelos robôs ($Barrier(i)$).

não pode mudar de *lugar* caso algum robô esteja na célula onde está a porta; (2) $ActDetect[id]$, informa quando algum robô poderá observar o estado da porta, sendo que se $ActDetect[id] == 0$ nenhum robô pode observá-la, caso contrário pode; e (3) $detect[id]$ é a variável de sincronização entre o modelo do robô com o modelo da porta quando o robô sai da vizinhança da porta. As variáveis *block* e *ActDetect* servem de guardas no modelo da visualização da porta e são atualizadas no modelo do robô.

No modelo da visualização da porta *id* pelos robôs, esta pode ser alterada para fechada ($Barrier(id).PF$) quando nenhum robô estiver na posição da porta ($block[id] == 0$) e algum robô i estiver em uma posição vizinha a esta porta ($ActDetect[id] == 1$). Para que a visualização pelos robôs volte ao *lugar* de não visível ($Barrier(id).PA$), o canal de sincronização $detect[id]$ é ativado. Isto ocorre quando algum robô i se afasta das células vizinhas a esta porta.

5.4.2.2 Definição das Condições de Aceitação

Conforme a definição do problema, deseja-se evitar a colisão do robô com as portas e com o obstáculo fixo, além de garantir que atinja a posição 12 do espaço de trabalho. Desta maneira, duas condições de aceitação são necessárias, segurança e alcançabilidade. Em termos de segurança, define-se o subconjunto dos estados a serem evitados $\mathcal{M}$, consoante a expressão 5.10.

Sendo $\bar{l}$ o vetor de *lugares* do sistema, no qual a primeira posição se refere ao robô R_1, a segunda à porta $Barrier(1)$ e por último à

porta $Barrier(2)$ e $n_{Barrier}$ é o número total de *lugares* de $Barrier(1)$ e $Barrier(2)$, têm-se

$$\mathcal{M} = \left\{ \bigcup_{z=1}^{3} \left(\bigcup_{j=1}^{n_{Barrier}} \left(\bigcup_{i=1}^{n_{Barrier}} (\bar{l}_z(i,j), \mathbb{R}_{\geq 0}) \right) \right) \right\} \tag{5.10}$$

onde

$$\begin{aligned} \bar{l}_1(i,j) &= [Robot(1).Pos08 \quad l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)] \\ \bar{l}_2(i,j) &= [Robot(1).Pos07 \quad Barrier(1).PF \quad l_{Barrier(2)}(j)] \\ \bar{l}_3(i,j) &= [Robot(1).Pos09 \quad l_{Barrier(1)}(i) \quad Barrier(2).PF] \end{aligned}$$

e $l_{Barrier(1)}(i)$ e $l_{Barrier(2)}(j)$ se referem ao *i-enésimo lugar* de $Barrier(1)$ e ao *j-enésimo lugar* de $Barrier(2)$, respectivamente.

Em termos de alcançabilidade, o subconjunto dos estados a serem atingidos, nomeado como $\mathcal{B}$, é apresentado pela expressão 5.11. Nele está contido o estado que representa o robô R_1 na posição 12.

$$\mathcal{B} = \left\{ \bigcup_{j=1}^{n_{Barrier}} \left(\bigcup_{i=1}^{n_{Barrier}} (\bar{l}(i,j), \mathbb{R}_{\geq 0}) \right) \right\} \tag{5.11}$$

onde $\bar{l}(i,j) = [Robot(1).Pos12 \quad l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)]$ e $l_{Barrier(1)}(i)$ e $l_{Barrier(2)}(j)$ se referem ao *i-enésimo lugar* de $Barrier(1)$ e ao *j-enésimo lugar* de $Barrier(2)$, respectivamente.

Como o objetivo do controlador é evitar $\mathcal{M}$ e atingir $\mathcal{B}$, tem-se o conjunto das condições de aceitação a serem utilizadas, representadas por $\Omega = \{(\mathcal{B}, \Diamond), (\overline{\mathcal{M}}, \Box)\}$.

5.4.2.3 Obtenção do Controlador

Tendo a planta $\mathcal{A}$ e as condições de aceitação $(\mathcal{B}, \Diamond)$ e $(\overline{\mathcal{M}}, \Box)$, define-se a expressão lógica 5.12, através da ferramenta UPPAAL-TIGA, para a síntese da estratégia vencedora.

$$A[not(\mathcal{M})\ U\ \mathcal{B}], \tag{5.12}$$

onde o controlador deve evitar $\mathcal{M}$ e atingir $\mathcal{B}$ para todos os caminhos. A inserção do termo *not* para o conjunto M implica na negação destes estados, i.e. nunca serão atingidos estes estados. A resposta da síntese mostrou que não é possível obter uma estratégia para o controlador a partir destas condições. Isto ocorre, pois, como as ações das portas são não controláveis e não existem passagens alternativas além das portas, não é possível garantir que o robô vai atingir seu objetivo. Logo, podemos concluir que o controlador não garante que o sistema entre em uma região onde a vitória é garantida.

Portanto, foi necessário utilizar o princípio da estratégia cooperativa acrescentando um novo termo a expressão lógica, conforme Equação 5.13.

$$E\Diamond A[not(\mathcal{M})\ U\ \mathcal{B}]. \tag{5.13}$$

Para a síntese da estratégia cooperativa do problema para 1 robô, o resultado obtido foi atingido com êxito. Neste caso, as ações do ambiente necessárias são as que no futuro, garantem que as portas sempre abrirão enquanto o robô não as tenha atravessado.

A solução é dividida em 2 zonas. Na primeira zona, que corresponde a parte do espaço de trabalho onde o robô ainda não passou pelas portas, a ajuda do ambiente é necessária enquanto na segunda zona, que corresponde a parte do espaço de trabalho onde o robô já passou

pelas portas, o controlador sempre atingirá seu objetivo independentemente do ambiente. Na Figura 5.6, têm-se representadas os autômatos que indicam as zonas que compõem esta estratégia. Os nomes dos *lugares* representados pela figura indicam primeiramente qual a posição do robô, depois o *lugar* da porta $Barrier(1)$ e por fim o *lugar* da porta $Barrier(2)$. Por exemplo, o *lugar* que representa o robô R_1 na posição 4, a porta $Barrier(1)$ fechada e a porta $Barrier(2)$ aberta é denotado como Pos04_PF_PA. Nesta estratégia, nota-se que o robô fica procurando uma porta aberta até conseguir a passagem e consequentemente a chegada na posição desejada.

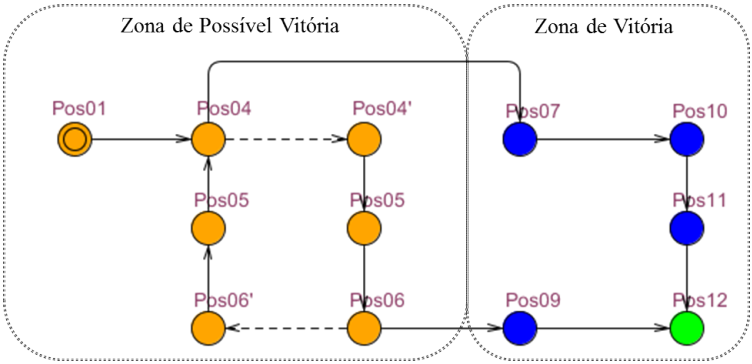


Figura 5.6: Autômato para uma estratégia vencedora.

Em relação ao controlador gerado para o caso de um robô utilizando TCS, podemos observar através da Figura 5.7 que a solução por AJT (Figura 5.6), em colorido, é sobreposta a solução por TCS (Figura 4.13), devido suas características de construção.

5.4.3 Coordenador Centralizado para o caso de 2 Robôs

A partir deste momento, todas as abordagens desenvolvidas por AJT utilizarão um SMR com 2 robôs, conforme apresentado na Figura

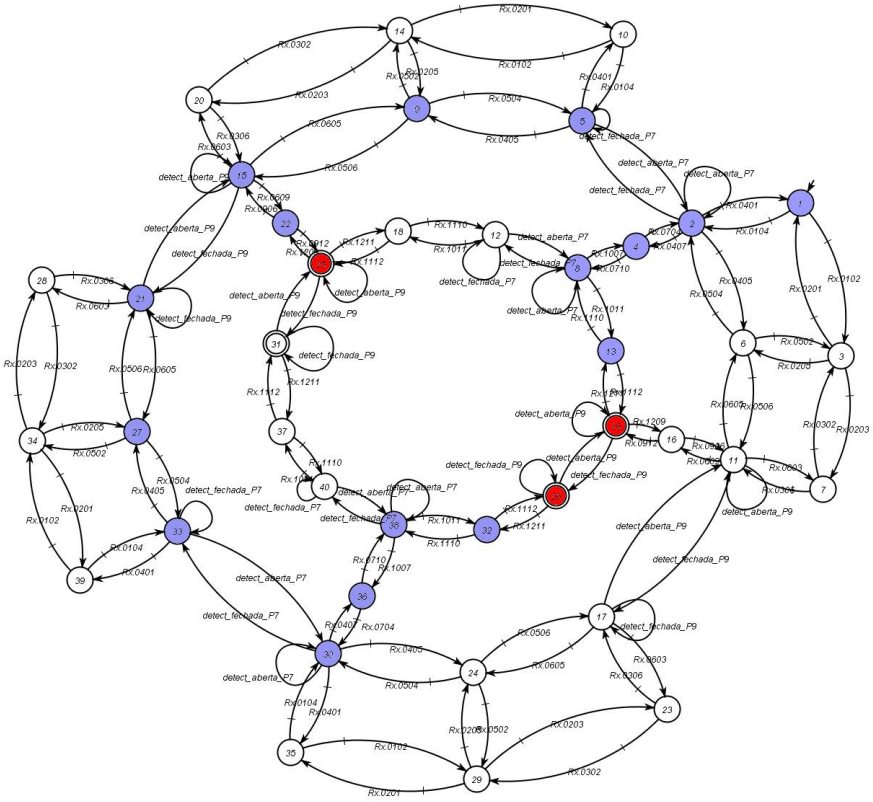


Figura 5.7: Autômato de coordenação AJT versus Autômato supervisor do TCS.

2.2. Os robôs R_1 e R_2, cujas posições iniciais são as células 1 e 3, possuem como objetivos atingir as posições 12 e 10, respectivamente. Nesta seção, será utilizado o conceito de um controlador único, centralizado, que controlará os movimentos de todos os robôs do sistema.

5.4.3.1 Modelagem do Sistema

Estende-se o problema com a inclusão de um novo robô. Neste caso, os valores $N_r = 2$, número de robôs e $N_b = 2$, número de portas, representam a nova configuração do problema. A rede de AJTs $\mathcal{A}$ que representa a planta a ser controlada é então composta pelos modelos dos robôs R_1 e R_2 dados por duas instâncias do modelo da Figura 5.4, denominados *Robot*(1) e *Robot*(2), e pelos modelos das portas dados por duas instâncias do modelo da Figura 5.5, denominados $Barrier(1)$ e $Barrier(2)$. As relações entre estes modelos são as mesmas do caso anterior, alterando apenas o número de robôs no espaço de trabalho.

5.4.3.2 Definição das Condições de Aceitação

Deseja-se para este caso gerar um plano que tenha como objetivos: para o R_1, atingir a posição 12 e para o R_2, atingir a posição 10, além de garantir a segurança de não haver colisão entre robôs, com as portas e com obstáculos fixos. Portanto, em termos de segurança, o subconjunto $\mathcal{M}$ dos estados a serem evitados é expresso pela Equação 5.14. Sendo n_{Pos} o número total de posições no espaço de trabalho, têm-se

$$\mathcal{M} = \left\{ \bigcup_{z=1}^{7} \left(\bigcup_{j=1}^{n_{Barrier}} \left(\bigcup_{i=1}^{n_{Barrier}} (\bar{l}_z(i,j), \mathbb{R}_{\geq 0}) \right) \right) \right\} \tag{5.14}$$

onde

$$
\begin{aligned}
\bar{l}_1(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos(k) \quad Robot(2).Pos(k) \\
&\quad l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)] \\
\bar{l}_2(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos08 \quad Robot(2).Pos(k) \\
&\quad l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)] \\
\bar{l}_3(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos(k) \quad Robot(2).Pos08 \\
&\quad l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)] \\
\bar{l}_4(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos07 \quad Robot(2).Pos(k) \\
&\quad Barrier(1).PF \quad l_{Barrier(2)}(j)] \\
\bar{l}_5(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos(k) \quad Robot(2).Pos07 \\
&\quad Barrier(1).PF \quad l_{Barrier(2)}(j)] \\
\bar{l}_6(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos09 \quad Robot(2).Pos(k) \\
&\quad l_{Barrier(1)}(i) \quad Barrier(2).PF] \\
\bar{l}_7(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos(k) \quad Robot(2).Pos09 \\
&\quad l_{Barrier(1)}(i) \quad Barrier(2).PF]
\end{aligned}
$$

e $l_{Barrier(1)}(i)$ e $l_{Barrier(2)}(j)$ se referem ao *i-enésimo lugar* de $Barrier(1)$ e ao *j-enésimo lugar* de $Barrier(2)$, respectivamente.

Em termos de alcançabilidade, o subconjunto $\mathcal{B}$ dos estados a serem atingidos é apresentado pela Expressão 5.15.

$$
\mathcal{B} = \left\{ \bigcup_{j=1}^{n_{Barrier}} \left(\bigcup_{i=1}^{n_{Barrier}} (\bar{l}(i,j), \mathbb{R}_{\geq 0}) \right) \right\} \tag{5.15}
$$

onde $\overline{l}(i,j) = [Robot(1).Pos12 \quad Robot(2).Pos10$
$l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)]$ e $l_{Barrier(1)}(i)$ e $l_{Barrier(2)}(j)$ se referem ao *i-enésimo lugar* de $Barrier(1)$ e ao *j-enésimo lugar* de $Barrier(2)$, respectivamente.

Como o objetivo do controlador é evitar $\mathcal{M}$ e atingir $\mathcal{B}$, tem-se o conjunto das condições de aceitação a serem utilizadas, $\Omega = \{(\mathcal{B}, \Diamond), (\overline{\mathcal{M}}, \Box)\}$.

5.4.3.3 Obtenção do Controlador

Como no caso para robô, a síntese da estratégia vencedora não é possível de ser obtida devido às mesmas condições citadas. Pois, as ações das portas são não controláveis e não existem passagens alternativas além das portas, logo, não é possível garantir que os robôs vão atingir seus objetivos. Portanto a utilização do principio da estratégia cooperativa também se faz necessário para este caso.

A síntese do controlador, a partir da expressão lógica abaixo, foi obtida com sucesso.

$$E\Diamond A[not(\mathcal{M}) \; U \; \mathcal{B}]. \tag{5.16}$$

Um autômato com 279 estados representando a estratégia a ser implementada no controlador foi gerado garantindo a vitória, desde que os obstáculos liberem a passagem alguma vez no futuro.

5.4.4 Coordenador Distribuído para o caso de 2 Robôs

A abordagem distribuída considera que o controle é implementado em cada robô, considerando-se que apenas seus movimentos são controláveis, enquanto os outros robôs que terão seus movimentos controlados também por controladores próprios são vistos por ele como

não controláveis. Porém, cada robô pode observar o movimento dos outros.

O controlador apresentado a seguir é o do robô R_1, sendo que, para este, o robô R_2 é um obstáculo dinâmico presente no espaço de trabalho. Similarmente, pode-se calcular o controlador para R_2, considerando R_1 como obstáculo dinâmico.

5.4.4.1 Modelagem do Sistema

Os valores $N_r = 1$, número de robôs e $N_b = 2$, número de portas, são diferentes em relação ao caso centralizado, pois a síntese do controlador levará em conta apenas um robô. Além disso, é acrescentado um novo valor $N_o = 1$, que é o número de robôs adversários. Logo, o sistema a ser controlado $\mathcal{A}$ é composto de uma rede de AJTs contendo o modelo do robô R_1 (jogador) dado por uma instância do modelo da Figura 5.4, $Robot(1)$, cujas ações são controláveis; os modelos da visualização das portas pelos robôs (adversário) dados por duas instâncias do modelo da Figura 5.5, $Barrier(1)$ e $Barrier(2)$, cujas ações são não controláveis; e pelo modelo do robô R_2 (adversário) dado por uma instância do modelo da Figura 5.8, nomeado por $Opponent(1)$, cujas ações também são não controláveis. Este último representa um robô que está no espaço de trabalho, no qual o controlador não pode agir sobre suas ações, i.e., todas as suas ações são então não controláveis do ponto de vista do robô R_1. As relações entre estes modelos são as mesmas do caso centralizado, exceto para o modelo do robô R_2 que não influencia nas relações, alterando apenas o número de robôs controláveis no espaço de trabalho.

Podemos notar que o alfabeto de entrada Σ é formado pela composição das ações controláveis do robô Σ^{R_1} com as ações não-controláveis das portas Σ^{B} e do robô Σ^{R_2}.

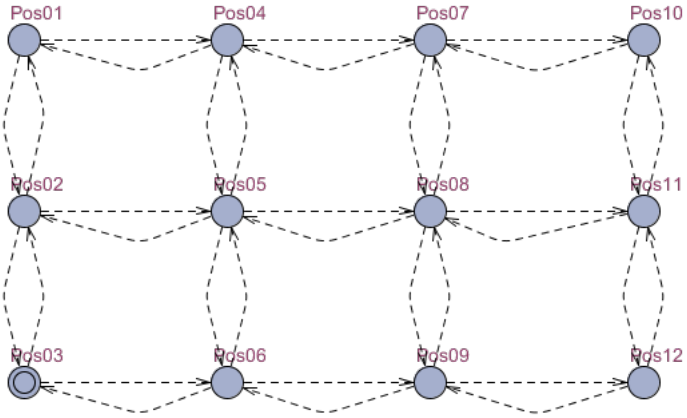


Figura 5.8: Modelo $Opponent(j)$ para um robô adversário ($j = 1...N_o$).

5.4.4.2 Definição das Condições de Aceitação

Deseja-se para este caso gerar um plano que tenha como objetivo para o R_1 atingir a posição 12, além de garantir a segurança de não haver colisão com as portas e com obstáculos fixos. Em relação à colisão entre robôs, o funcionamento é o mesmo do caso distribuído por TCS, onde cada robô é responsável por si, ou seja, a prevenção de colisão entre os robôs será apenas cogitada para o robô em que se está desenvolvendo o supervisor, neste caso o R_1. Desta maneira, o impedimento da colisão entre robôs é garantido através da combinação dos supervisores de cada robô presente no espaço de trabalho. Portanto, em termos de segurança, o subconjunto $\mathcal{M}$ dos estados a serem evitados é expresso pela Equação 5.17.

$$\mathcal{M} = \left\{ \bigcup_{z=1}^{4} \left(\bigcup_{j=1}^{n_{Barrier}} \left(\bigcup_{i=1}^{n_{Barrier}} (\bar{l}_z(i,j), \mathbb{R}_{\geq 0}) \right) \right) \right\} \tag{5.17}$$

onde

$$
\begin{aligned}
\bar{l}_1(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos(k) \quad Opponent(1).Pos(k) \\
&\qquad l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)] \\
\bar{l}_2(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos08 \quad Opponent(1).Pos(k) \\
&\qquad l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)] \\
\bar{l}_3(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos07 \quad Opponent(1).Pos(k) \\
&\qquad Barrier(1).PF \quad l_{Barrier(2)}(j)] \\
\bar{l}_4(i,j) &= \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos09 \quad Opponent(1).Pos(k) \\
&\qquad l_{Barrier(1)}(i) \quad Barrier(2).PF]
\end{aligned}
$$

e $l_{Barrier(1)}(i)$ e $l_{Barrier(2)}(j)$ se referem ao *i-enésimo lugar* de $Barrier(1)$ e ao *j-enésimo lugar* de $Barrier(2)$, respectivamente.

Em termos de alcançabilidade, o subconjunto $\mathcal{B}$ dos estados a serem atingidos é apresentado pela Expressão 5.18.

$$
\mathcal{B} = \left\{ \bigcup_{j=1}^{n_{Barrier}} \left(\bigcup_{i=1}^{n_{Barrier}} (\bar{l}(i,j), \mathbb{R}_{\geq 0}) \right) \right\} \tag{5.18}
$$

onde $\bar{l}(i) = \bigcup_{k=1}^{n_{Pos}} [Robot(1).Pos12 \quad Opponent(1).Pos(k)$ $l_{Barrier(1)}(i) \quad l_{Barrier(2)}(j)]$ e $l_{Barrier(1)}(i)$ e $l_{Barrier(2)}(j)$ se referem ao *i-enésimo lugar* de $Barrier(1)$ e ao *j-enésimo lugar* de $Barrier(2)$, respectivamente.

Como o objetivo do controlador é evitar $\mathcal{M}$ e atingir $\mathcal{B}$, tem-se o conjunto das condições de aceitação a serem utilizadas, $\Omega = \{(\mathcal{B}, \Diamond), (\overline{\mathcal{M}}, \Box)\}$.

5.4.4.3 Obtenção do Controlador

Como nos casos anteriores, a síntese da estratégia vencedora não é possível de ser obtida devido às mesmas razões já citadas além de não poder garantir que o robô R_2 não vá colidir com o robô R_1. Pois, as ações das portas e do robô R_2 são não controláveis. Desta maneira, o uso da estratégia cooperativa se faz muito necessária neste caso, pois, ela incorpora na síntese da estratégia as possibilidades do ambiente cooperar com o controlador. Sabe-se que as portas abrirão em algum momento no futuro e, principalmente, que o mesmo tipo de controlador será implementado no robô R_2, logo, combinando os controladores distribuídos que garantem as propriedades desejadas.

A síntese do controlador, a partir da expressão lógica abaixo, foi obtida com sucesso.

$$E\Diamond A[not(\mathcal{M})\ U\ \mathcal{B}]. \tag{5.19}$$

Um autômato com 220 estados representando a estratégia a ser implementada no controlador do robô R_1 foi gerado garantindo a vitória, desde que os obstáculos liberem a passagem alguma vez no futuro. Para o robô R_2, o autômato para o controlador foi também obtido com êxito tendo o mesmo número de estados para o caso do robô R_1.

5.5 Coordenação de SMR para Multíplos Robôs

Em todos os casos abordados por AJT, obteve-se uma estratégia para o controle de um sistema multi-robôs. Entretanto, a fim de analisar as limitações da metodologia, foi realizada uma expansão do estudo de caso. Considerando-se agora um espaço de trabalho com 20 células,

2 portas e 2 obstáculos fixos, foram sintetizados controladores para 1, 2 e 3 robôs nas abordagens centralizada e distribuída. A Tabela 5.3 apresenta o número de estados dos autômatos coordenadores representando a estratégia pela abordagem centralizada, enquanto a Tabela 5.4 apresenta para a abordagem distribuída.

Nº de Robôs	Nº de Estados	Tempo de Execução
1	31	$0,04s$
2	725	$0,08s$
3	13735	$1,58s$

Tabela 5.3: Coordenadores pela abordagem centralizada.

Nº de Robôs	Nº de Estados	Tempo de Execução
1	31	$0,04s$
2	589	$0,08s$
3	11191	$0,88s$

Tabela 5.4: Coordenadores pela abordagem distribuída.

5.6 Discussões

Neste capítulo foi proposta uma abordagem para o planejamento de tarefas para SMR utilizando AJT. O método do projeto do controlador utilizando AJT com os algoritmos *On-the-fly*, permite tratar simultaneamente múltiplas condições de alcançabilidade e segurança. Entretanto, eles apresentam algumas limitações como não permitir o tratamento de problemas de coordenação de robôs que requerem o cumprimento de tarefas intermediárias.

A dinâmica da trajetória do robô ou a exigência de um tempo mínimo para realizar determinada tarefa introduzem a necessidade de uma representação temporal explícita nos modelos a serem utilizados na síntese. A representação na forma de um AJT permite trabalhar sobre espaços de estados muito reduzidos em relação à representação

temporal normalmente utilizada em TCS [10]. Tendo em vista a complexidade inerente ao problema de coordenação de SMR, isto caracteriza uma grande vantagem da abordagem AJT sobre TCS, nestes casos. Por outro lado, o uso de expressões de lógica temporal provê uma maneira simples e eficiente de especificar as condições de vitória e dá maior poder de expressão para propriedades de alcançabilidade, quando comparada à TCS. Além disso o método explicita sempre um caminho para atingir a vitória, enquanto TCS considera todas as trajetórias possíveis, e dessa forma necessita de um nível de decisão adicional a ser computado para garantir o atingimento dos objetivos, em aplicações como as aqui tratadas.

Quanto às diferentes dimensões de organização desenvolvidas nesta abordagem, o desenvolvimento seguiu a mesma linha dos casos por TCS. No caso distribuído, o foco continua em apenas um robô. Assim, os outros robôs existentes no espaço de trabalho são vistos como obstáculos dinâmicos, sendo seus eventos não-controláveis. Ou seja, o controlador apenas receberá a informação de onde estão os outros robôs e nunca agirá sobre eles, embora no caso modular o robô também envia a informação de localização para os outros robôs.

Por fim, em todos os casos, foi obtido um plano no qual os robôs atendem as propriedades de segurança e alcançabilidade. Todavia, pela característica da estratégia cooperativa, uma parte da estratégia do controlador possui uma permissividade que considera mais de uma possibilidade de transição a cada momento, assemelhando-se assim ao TCS. Portanto, a propriedade de alcançabilidade, devido à estratégia cooperativa, é atingida com menor eficácia em comparação de uma estratégia sem cooperação. Mesmo assim, a eficiência em relação aos caminhos seguidos pelos robôs até suas posições alvos é considerada satisfatória.

Capítulo 6

Implementação do Coordenador de um SMR

Nos capítulos anteriores foram apresentadas diferentes abordagens para a síntese de controladores para o planejamento e coordenação de tarefas em sistemas multi-robôs. Para validar estas abordagens, foram implementados estes controladores em SMR. A abrangência desta implementação engloba diversos fatores a serem considerados. Dentre eles, têm-se a topologia a ser utilizada, a comunicação entre robôs, a sistema de detecção de obstáculos, entre outros.

Neste capítulo é apresentada uma arquitetura genérica para a implementação de sistemas multi-robôs que fornece desde o planejamento de tarefas ao controle operacional dos robôs presentes no sistema. Esta arquitetura é estendida para ambas as topologias consideradas nesta dissertação, *centralizada* e *distribuída*. Além disso, serão mostrados como foi realizado o desenvolvimento de cada parte da arquitetura e

quais foram as tecnologias utilizadas para a realização dos testes em um ambiente real.

6.1 Desenvolvimento de Arquitetura para Sistemas Multi-Robôs

A definição de uma arquitetura genérica para sistemas robóticos visa a generalização de sua implementação e a padronização dos vários níveis existentes. Neste trabalho foi desenvolvida uma arquitetura, baseada em algumas propostas existentes [22, 26, 28], e para a qual duas extensões em relação à topologia do sistema são consideradas.

6.1.1 Arquitetura Centralizada

Esta extensão, visa a implementação de um controle centralizado responsável pela locomoção dos vários robôs e cujo o agente controlador poderá ser um computador central que se comunica com todos os robôs do sistema. A figura 6.1 apresenta a arquitetura do tipo centralizada com todos os módulos pertencentes à esta arquitetura.

Os módulos são divididos entre os que são implementados em um computador e utilizados no modo *offline*, os que são implementados em um computador e utilizados no modo *online* e os que são implementados em cada robô e utilizados no modo *online*.

6.1.1.1 Computador - *Offline*

Existem dois módulos que são implementados em um computador de forma *offline*:

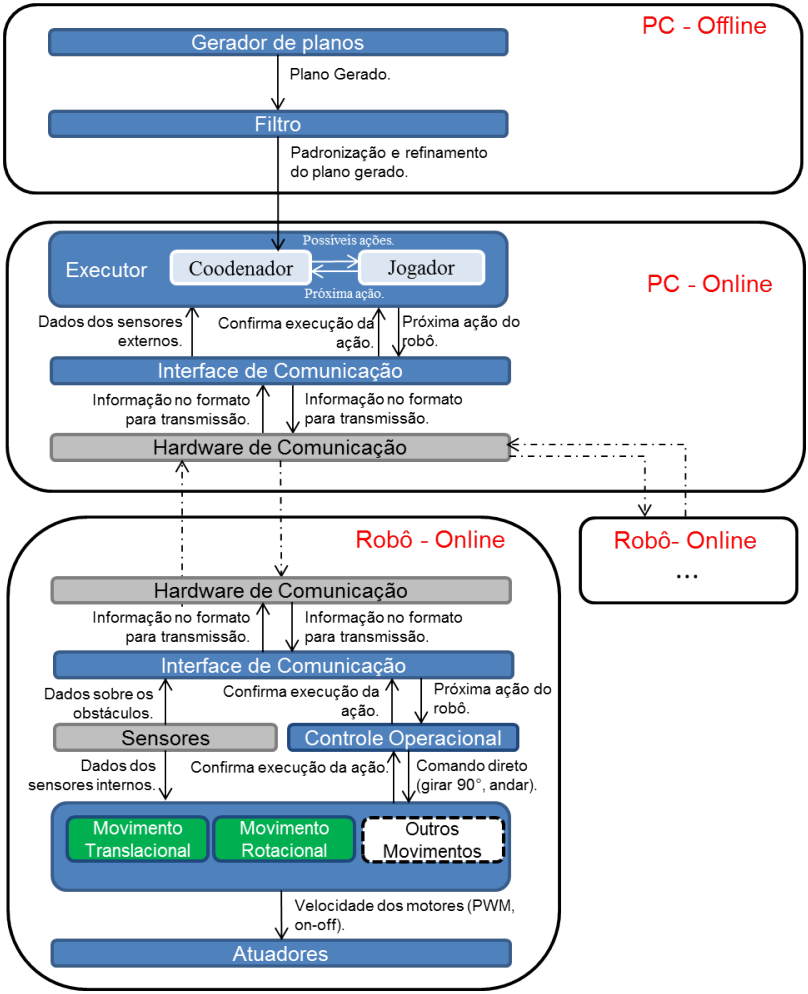


Figura 6.1: Extensão centralizada da arquitetura para SMR.

***Gerador de Planos*,** onde será desenvolvido o plano e sintetizado o coordenador a fim de fornecer aos robôs suas tarefas a serem executadas e como executá-las. Diferentes tipos de metodologias podem ser utilizadas para tal objetivo, sendo a TCS e o AJT as apresentadas neste trabalho;

***Filtro*,** no qual o principal objetivo é eliminar do plano gerado qualquer informação desnecessária à coordenação dos robôs que porventura a ferramenta de síntese gerar, e realizar um refinamento no coordenador a fim de padronizar o autômato independentemente do método e ferramenta utilizada para a síntese.

6.1.1.2 Computador - *Online*

Os módulos implementados em um computador e utilizados de modo *online* executam a coordenação do sistema. Existem três módulos com estas características:

No *Executor* é utilizado um autômato refinado, no qual a partir deste modelo o controlador enviará as ações de comando para o(s) robô(s);

A *Interface de Comunicação* é responsável por gerenciar a comunicação entre o computador e os robôs, colocando os dados no formato em que devem ser enviados e identificando as informações recebidas dos robôs. Um módulo similar se encontra na nível do robô a fim de realizar a PC-robô.

6.1.1.3 Robô - *Online*

Os módulos nesta etapa representam a parte que é embarcada no(s) robô(s). Existem três módulos que são embarcados:

A *Interface de Comunicação* é semelhante ao do caso anterior;

O *Controle Operacional* utiliza-se do comando passado pelo *Executor* através da interface para definir as próximas ações do robô (andar, girar, etc). Uma vez que o comando for executado, deve-se informar da ocorrência ao *Executor* do computador *on-line*;

O *Controle Contínuo* é a parte do sistema responsável por executar os comandos do *Controle Operacional*. Para isto, faz-se uso dos sensores e atuadores no robô a fim de executar a parte do controle de trajetória do controle híbrido final. Uma vez que os comandos foram executados, esse bloco deve informar ao *Controle Operacional* de sua conclusão.

6.1.2 Arquitetura Distribuída

A extensão distribuída da arquitetura proposta para os sistema multi-robôs possui os mesmos módulos presentes na extensão centralizada, conforme apresenta a Figura 6.2. Porém, neste caso, o *Executor* é embarcado dentro de cada robô, i.e, o controle discreto do sistema é feito de forma distribuída, onde cada robô é autônomo em relação as suas ações. Desta forma, os módulos implementados em um computador e executados de forma *online* na extensão centralizada agora serão embarcados em cada robô, na forma pela abordagem distribuída. Diferentemente do caso anterior, os robôs terão que se comunicar entre si a fim de trocarem informações a respeito de suas posições, o que implica na existência de um modelo de interface de comunicação com outros robôs em cada um deles.

Os módulos são divididos entre os que são implementados em um computador e utilizados no modo *offline*, e os que são implementados em um robô e utilizados no modo *online*. Além disso, consideram-se os outros robôs presentes no espaço de trabalho como fazendo parte do ambiente e cuja informação sobre suas posições serão comunicadas aos outros robôs. Portanto, as diferenças entre ambas extensões estão

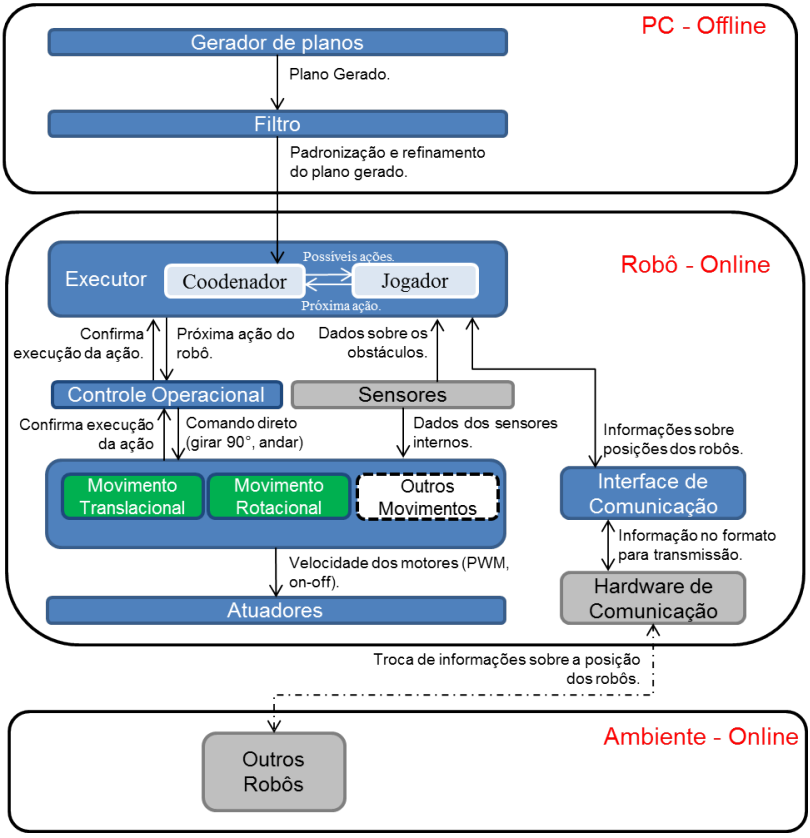


Figura 6.2: Extensão distribuída da arquitetura para SMR.

no nível em que os comandos de ação dos robôs são efetuados e principalmente no tipo de comunicação realizada, alterando de Computador-Robô no caso centralizado para Robô-Robô.

6.2 Descrição dos Módulos da Arquitetura para Sistemas Multi-Robôs

Nesta seção será descrito cada módulo presente na arquitetura.

6.2.1 Gerador de Planos

O módulo *Gerador de Planos* é responsável pela síntese do plano de tarefas a ser aplicado no sistema a ser controlado. Diversas podem ser as metodologias utilizadas para a obtenção do plano. Nos capítulos 4 e 5 foram apresentados dois métodos para sintetizar o controlador discreto a ser utilizado. No primeiro, através da ferramenta IDES (*Integrated Discrete-Event Systems*)[1] foi obtido o plano de tarefas para as duas topologias aqui tratadas. O plano é gerado na forma de um autômato em um arquivo de texto que é repassado ao *Filtro* a fim de ser enviado ao *Executor*. No método por AJT, o plano é obtido através da ferramenta UPPAAL-TIGA[2], que também é gerado como resultado um arquivo de texto com o plano a ser implementado. Do mesmo modo que no método por TCS, este arquivo é enviado ao *Filtro* para ser padronizado antes de ser repassado ao *Executor*. Este módulo implementa as diversas abordagens de desenvolvimento de um plano de tarefas conforme apresentado nos capítulos anteriores.

[1] https://qshare.queensu.ca/Users01/rudie/www/software.html
[2] www.cs.aau.dk/~adavid/tiga

6.2.2 Filtro

O principal objetivo deste módulo é padronizar o plano gerado no formato pré-determinado de um autômato para que, qualquer que seja a metodologia utilizada para a geração do plano, o *Executor* funcione da mesma maneira. Além disso, nos arquivos de saída das ferramentas utilizadas para a síntese dos controladores discretos, existem informações adicionais ou incompletas para a padronização do autômato coordenador. Deste modo, o filtro também é responsável em adequar o autômato de coordenação conforme exigido no módulo seguinte. A figura 6.3 apresenta o diagrama de classes onde se encontra o filtro e a estrutura do autômato padrão para o plano de tarefas.

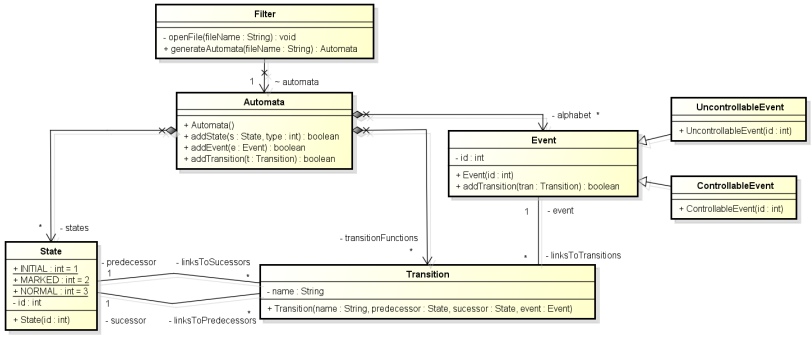


Figura 6.3: Diagrama de classes do módulo *Filtro*.

6.2.3 Executor

O módulo *Executor* é o responsável por coordenar o sistema a ser controlado. Como já discutido anteriormente, ele pode ser implementado tanto em um computador central quanto embarcado em um robô móvel. A partir do autômato coordenador gerado pelo *Filtro*, o *Executor* define as ações que o(s) robô(s) deve(m) tomar levando em consideração as informações dos eventos não controláveis do sistema.

A execução deste módulo funciona como um jogo, onde o *Executor* a cada instante envia uma ação para o robô executar. Por sua vez, as ações do ambiente são transmitidas para o *Executor*, que as considera no modelo do autômato e depois seleciona novas ações para o(s) robô(s). Seguindo estes passos, o *Executor* finaliza sua execução assim que atingir seus objetivos, i.e, os estados marcados do autômato que representa o plano de tarefas. Internamente, este módulo é composto pelo autômato coordenador, que fornece as opções das ações que poderão ser efetuadas; e pelo jogador, que seleciona uma ação dentre as possíveis para ser enviada ao *Controle Operacional*.

Este módulo é composto basicamente por três componentes: *Engine*, *RuleBase* e *Workspace*, conforme o diagrama de classes apresentado na Figura 6.4.

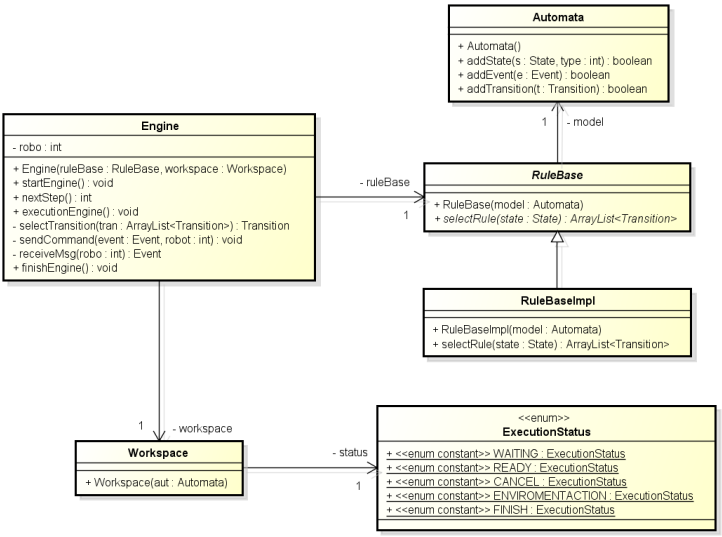


Figura 6.4: Diagrama de classes do módulo *Executor*.

***Engine*:** É a principal classe deste módulo, cuja responsabilidade é gerenciar a execução do sistema. É a partir dela que são realizadas a comunicação com os robôs, o envio de ações para os robôs, o

tratamento das ações do ambiente e a decisão de término das tarefas. Para realizar seu gerenciamento, outras duas classes são necessárias, *RuleBase* e *Workspace*.

***RuleBase*:** É um classe abstrata que utiliza o autômato que representa o plano de tarefas. A derivação desta classe implementa o método que representa o submódulo *jogador* da arquitetura (RuleBaseImpl). Desta maneira, pode-se utilizar diferentes abordagens para a decisão das ações a serem enviadas ao robô. Portanto, a classe *Engine* utiliza esta classe para acessar o plano de tarefas e, desta forma, poder selecionar uma ação dentre algumas possíveis em um determinado estado do sistema.

***Workspace*:** É nesta classe que é definido o status da execução do plano de tarefas. Existem 5 estados para a execução do plano, (1) *WAITING*, quando o *Executor* espera uma resposta de confirmação da ação enviada; (2) *READY*, quando o *Executor* está pronto para enviar uma ação; (3) *CANCEL*, quando for cancelada a execução do plano de tarefas; (4) *ENVIROMENTACTION*, quando o *Executor* recebeu uma ação do ambiente; e (5) *FINISH*, quando o *Executor* atingiu os objetivos do plano de tarefas. Além disso, é também definido nesta classe o estado atual do sistema, representando as posições dos robôs e os estados dos obstáculos atuais.

6.2.4 Controle Operacional

O módulo *Controle Operacional*, independentemente da topologia do sistema utilizada, é sempre embarcado no robô. Ele é responsável por interpretar as ações enviadas pelo *Executor* em comandos para o *Controle Contínuo* do robô. Por exemplo, a ação do *Executor* para o robô R_1, onde ele deve andar da célula 2 para a célula 3 é traduzida pelo *Controle Operacional* como um conjunto de ações onde o robô R_1

deve girar para a posição correta e andar uma célula. Estes comandos acionam movimentos específicos no módulo do *Controle Contínuo*, sendo os implementados atualmente, *rotacionar à direita*, *rotacionar à esquerda* e *andar à frente*. Quando forem concluídos os comandos deste módulo, uma mensagem de confirmação é enviada ao módulo *Executor* informando que a ação enviada por este foi efetuada com sucesso.

6.2.5 Controle Contínuo

O módulo responsável por executar os comandos do *Controle Operacional* baseia-se na construção de controladores contínuos simples que realizam movimentos já pré-definidos, onde, cada um efetua um comando específico do *Controle Operacional* para acionar os atuadores. Para isto, faz-se uso dos sensores internos para medir as velocidades nos atuadores e identificar a conclusão de um comando. Quando for concluído um comando, este módulo envia uma mensagem de confirmação de conclusão do comando ao *Controle Operacional*.

Nesta dissertação o foco está voltado ao planejamento de tarefas e, portanto, ao controle discreto. Por este motivo, controladores contínuos simples foram utilizados apenas para viabilizar o desenvolvimento do sistema como um todo. Desta maneira, apenas 2 movimentos foram desenvolvidos até este momento, Movimento Translacional e Movimento Rotacional. O primeiro é responsável por realizar a locomoção do robô de uma célula à outra, desde que o robô já esteja na direção correta, ou seja, o robô só pode se mover translacionalmente à frente. Estas trajetórias seguidas pelos robôs seguem um movimento linear de velocidade constante. O outro movimento é responsável por rotacionar o robô em 90°à esquerda ou à direita. Este movimento direciona o robô à célula aonde se deseja atingir. Em futuros trabalhos, objetiva-se o uso de diferentes movimentos a fim de otimizar o caminho seguido pelos robôs.

6.2.6 Comunicação

O módulo de *Comunicação* é necessário em ambas as arquiteturas. Na centralizada, a comunicação é feita entre o computador central e os robôs a serem controlados, enquanto na distribuída, a comunicação é feita entre robôs. No primeiro caso, uma interface de comunicação é implementada tanto no computador central quanto no robô a fim de realizar a troca de informações necessárias. O computador envia os comandos do *Executor* pela interface de comunicação. Por outro lado, cada robô envia as confirmações dos comandos do *Executor* e as ações do ambiente também pela interface de comunicação. No outro caso, a interface de comunicação é implementada em cada robô, onde apenas são trocadas as informações sobre a posição destes robôs.

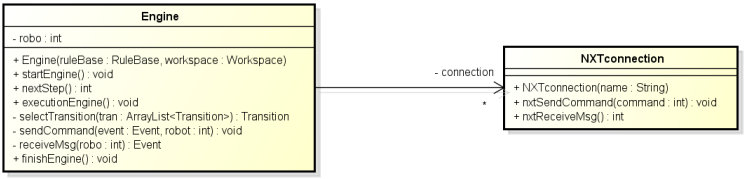


Figura 6.5: Diagrama de classes da interface de comunicação.

A figura 6.5 apresenta o diagrama de classes que se aplica em ambos os casos. A classe *Engine* do módulo *Executor* é associada a classe *NXTconnection*, que representa a interface de comunicação ligada ao *Executor*.

6.3 Tecnologias Utilizadas nos Experimentos

Para a implementação da arquitetura em um ambiente real a fim de realizar testes do desenvolvido realizado, certas tecnologias foram utilizadas em cada parte do sistema.

6.3.1 Geração de Planos

Para a geração de planos foram utilizadas duas ferramentas, IDES e UPPAAL-TIGA. A primeira é um software projetado para ajudar na solução de problemas de sistemas a evento discretos, utilizando-se de autômatos ADEF. A segunda ferramenta também é um software projetado para ajudar na solução de problemas de sistemas a evento discretos, porém focado para a solução de jogos baseados em autômato-jogo temporizado.

6.3.2 Arquitetura

Com exceção do módulo de *Geração de Planos*, o desenvolvimento da arquitetura foi realizada com a linguagem de programação orientada a objeto JAVA[3] através do ambiente de desenvolvimento Eclipse[4]. O *firmware* utilizado nos robôs foi implementado também em linguagem JAVA utilizando-se de uma Máquina Virtual Java chamada LeJOS[5], que proporciona a fácil implementação das funcionalidades desejadas para o robô.

6.3.3 Comunicação

Por se tratar de um sistema multi-robôs móveis, a comunicação deve ser sem fio (*wireless*) a fim de proporcionar a mobilidade dos robôs e troca de informações necessárias para o sistema. Dentre os diversos padrões de comunicação *wireless*, foi escolhido o padrão *Bluetooth* (IEEE 802.15.1) por ser uma tecnologia para a comunicação sem fio entre dispositivos eletrônicos a curtas distâncias, ser de fácil utilização e manter a potência de transmissão extremamente baixa poupando energia da bateria. Sendo a aplicação desta dissertação em um ambiente

[3] http://www.java.com
[4] http://www.eclipse.org/
[5] http://lejos.sourceforge.net

pequeno e *indoor*, julga-se suficiente o uso da tecnologia de comunicação *Bluetooth* e também por ser a única tecnologia *wireless* disponível nos robôs utilizados.

6.3.4 Robôs

O robôs utilizados nesta dissertação foram construídos a partir de uma linha de robôs programáveis fabricados pelo Grupo Lego, LEGO MINDSTORMS®[6]. Esta linha permite o uso de diversos componentes para a construção de robôs, entre eles, sensores, atuadores e transmissores sem fio. Portanto, com o uso desta tecnologia, foi possível a construção de robôs que atendessem as necessidades específicas deste trabalho, incluindo a utilização da tecnologia *Bluetooth*.

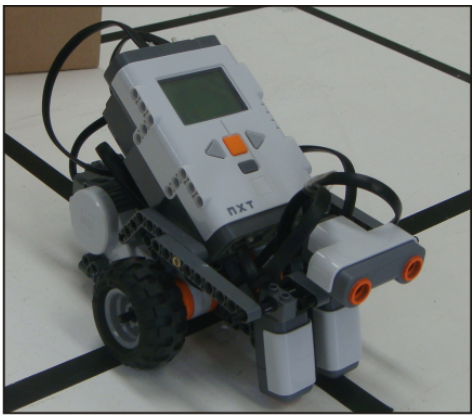


Figura 6.6: Robô móvel.

A figura 6.6 apresenta um exemplar do robô construído para a realização dos testes. Ele é equipado com 2 sensores de luz, responsáveis pela localização no espaço de trabalho, onde o robôs seguem linhas pretas colocadas no espaço de trabalho; 1 sensor sônico, responsável pela detecção dos obstáculos; e por 2 atuadores (2 motores em cada eixo do robô), responsáveis pela locomoção do robô.

[6] http://mindstorms.lego.com

6.4 Experimentos Realizados

Os experimentos em um ambiente real para o caso de uso apresentado na Seção 2.3.1 foram realizados utilizando dois robôs LEGO MINDSTORMS®, um plano onde as células foram delimitadas por linhas escuras, nos quais os robôs seguirão as trajetórias e por obstáculos que podem ser removidos manualmente, representando a dinamicidade destes. A Figura 6.7 apresenta o ambiente de testes.

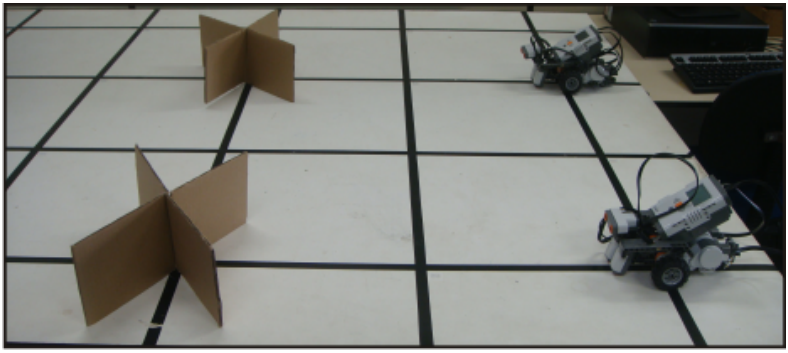


Figura 6.7: Multi-Robôs no espaço de trabalho.

Os testes foram feitos utilizando a extensão centralizada da arquitetura. Diferentes configurações foram analisadas, a fim de corroborar com o método utilizado e com a arquitetura proposta. A tabela 6.1 apresenta as configurações utilizadas, onde x_0 e X_m representam os estados iniciais e marcados de R_1 e R_2, respectivamente; P e F representam as posições das portas e dos obstáculos fixos no espaço de trabalho; e T representa o tempo em que os robôs atingiram seus objetivos considerando o movimento das portas no mesmo instante para todos os casos.

Em todos os casos analisados, os robôs atingiram seus objetivos com êxito. Como já era esperado, pelo método utilizado já se garante de antemão que o coordenador sempre atinge seu objetivo, exceto quando há erro na modelagem. Nesta caso, pode-se concluir que os modelos

Experimento	x_0 $R_1/R2$	X_m $R_1/R2$	P	F	T
1	1/3	12/10	7 e 9	8	$51,40s$
2	1/3	12/10	7 e 8	9	$57,48s$
3	1/3	12/10	8 e 9	7	$67,23s$
4	1/3	12/10	8	7 e 9	$67,54s$
5	1/3	12/10	5	7 e 9	$63,14s$
6	1/3	12/10	11	7 e 9	$55,23s$
7	1/12	12/1	7 e 9	8	$57,13s$
8	1/10	12/3	7 e 9	8	$75,02s$
9	3/10	10/3	7 e 9	8	$79,95s$
10	3/12	10/1	7 e 9	8	$57,95s$

Tabela 6.1: Experimentos realizados.

representam a realidade do problema, verificado por diversos experimentos com diferentes configurações. Apenas na questão temporal, que, devido a necessidade da inserção de um nível adicional de tomada de decisão, não são sistemáticos. Isto depende fortemente da estratégia implementada.

6.5 Discussão

Neste capítulo foi proposta uma arquitetura genérica de implementação para sistema multi-robôs. Duas extensões para a arquitetura foram desenvolvidas, cada uma levando em consideração a topologia do sistema utilizado, centralizada e distribuída.

Em relação às metodologias a serem utilizadas, a arquitetura é capaz de suportar diferentes métodos para o módulo *Gerador de Planos*. Isto é possível devido ao módulo *Filtro*, que para cada metodologia utilizada uma transformação para o formato do autômato padrão é realizada. Desta maneira, viabilizando a possibilidade de se utilizar novos métodos para a síntese de controladores discretos, provados com o uso de TCS e AJT.

Capítulo 7

Conclusões e Perspectivas

O projeto de controladores para SMR que garante condições de segurança e alcançabilidade foi abordado nesta dissertação, utilizando-se de métodos formais para a garantia das propriedades exigidas no projeto. Mais especificamente, duas técnicas foram abordadas em duas topologias diferentes do SMR a fim de poder explorar e comparar suas potencialidades e limitações.

A primeira parte do trabalho focou no desenvolvimento do projeto de controladores através da Teoria de Controle Supervisório (TCS) utilizando tanto a abordagem monolítica quanto a modular local. Embora o controle supervisório modular local proponha uma solução, cujo custo computacional é consideravelmente menor e o esforço de implementação mais sistematizada e menos árdua, o processo de verificação da modularidade local ainda requer o cálculo do comportamento mútuo de todos os supervisores, o que pode gerar um cálculo de complexidade exponencial. De modo geral, em ambas abordagens, devido à característica do controle supervisório de ser o mais permissível possível,

implicando que cada robô tem mais de uma possibilidade de caminho a cada posição, uma lógica adicional na implementação deve ser introduzida a fim de permitir aos robôs de atingir seus objetivos.

Na segunda parte do trabalho o projeto do controlador se baseou no Autômato-Jogo Temporizado (AJT) utilizando a formulação na forma de um jogo entre o controlador e o ambiente não-controlável, representados a partir de modelos de autômatos temporizados. A síntese do controlador consiste em buscar a vitória deste no jogo contra o ambiente, atendendo as propriedades de alcançabilidade e segurança desejadas.

O método do projeto do controlador utilizando AJT com algoritmo *On-the-fly*, adotado nesta dissertação, permite tratar simultaneamente múltiplas condições de alcançabilidade e segurança. Entretanto, esta abordagem apresenta algumas limitações como não permitir o tratamento de problemas de coordenação de robôs que requerem o cumprimento de tarefas intermediárias.

A dinâmica da trajetória do robô ou a exigência de um tempo mínimo para realizar determinada tarefa introduzem a necessidade de uma representação temporal explícita nos modelos a serem utilizados na síntese. A representação na forma de um AJT permite trabalhar sobre espaços de estados muito reduzidos em relação à representação temporal normalmente utilizada em TCS [10]. Tendo em vista a complexidade inerente ao problema de coordenação de sistemas multi-robôs, isto caracteriza uma grande vantagem da abordagem AJT sobre TCS, nestes casos. Por outro lado, o uso de expressões de lógica temporal provê uma maneira simples e eficiente de especificar as condições de vitória e dá maior poder de expressão para propriedades de alcançabilidade, quando comparada à TCS. Além disso o método explicita sempre um caminho para atingir a vitória, enquanto TCS considera todas as trajetórias possíveis, e dessa forma necessita de um nível de decisão adicional a ser computado para garantir o atingimento dos objetivos, em aplicações como as aqui tratadas.

As perspectivas deste trabalho incluem o estudo das potencialidades e limitações de outros métodos [30, 35] para o projeto de controladores para SMR e sua posterior inserção no *framework* aqui desenvolvido. Além de expandir as possíveis técnicas a serem utilizadas, novas funcionalidades no *framework* podem ser inseridas, tais como novos tipos de movimentos dos robôs (dinâmica contínua) e novas técnicas para a utilização da topologia distribuída, por exemplo outras tecnologias de sensoriamento.

Anexo A

Lógica Temporal

Lógica Temporal (LT) é um ramo especial da lógica simbólica especificamente feita para descrever qualquer sistema de regras, raciocínios e simbolismos para representar preposições qualificadas em termos do tempo, sendo muito utilizada na vericação formal de sistemas computacionais. Na vericação, a lógica temporal serve para especificar formalmente propriedades relacionadas com a execução de um sistema. A Lógica temporal usa proposições atômicas para fazer afirmações sobre os estados. Estas proposições são relações elementares, as quais, num dado estado, possuem um valor verdadeiro ou falso bem definido. Em lógica temporal, existem dois tipos de operadores, lógicos e temporais. Também encontra-se na lógica temporal quantificadores de caminho que permitem quantificar sobre o conjunto de execuções. Sendo ϕ e φ fórmulas de estado que representam propriedades do sistema, têm-se definidos os seguintes operadores lógicos e temporais.

A.1 Operadores Lógicos

Uma fórmula proposicional, definida em lógica proposicional, é uma combinação de proposições e operadores lógicos. Este tipo de fórmula também é usada em lógica temporal. Os possíveis operadores lógicos são:

Constantes: $true$ e $false$;

Negação: $\neg$;

Conjunção: $\vee$;

Disjunção: $\wedge$;

Implicação: $\Rightarrow$;

Dupla Implicação: $\Longleftrightarrow$.

A.2 Operadores Temporais

Os operadores temporais permitem construir expressões relacionadas ao sequenciamento dos estados ao longo de uma execução e não apenas aos estados individualmente. Os possíveis operadores temporais são:

X **ou** $\bigcirc$ **(*Next*):** A fórmula $X\phi$ é verdadeira quando o próximo estado satisfaz a propriedade ϕ;

F **ou** $\Diamond$ **(*Future*):** A fórmula $F\phi$ é verdadeira quando pelo menos um estado no futuro satisfaz ϕ sem, entretanto, especificar qual estado;

G **ou** $\Box$ **(*Globally*):** A fórmula $G\phi$ é verdadeira quando todos os estados futuros satisfazem ϕ;

U **(*Until*):** A fórmula $\phi U \varphi$ é verdadeira quando ϕ for verdadeira até que (*Until*) φ seja verdadeira;

W **(*Weak Until*):** A fórmula $\phi W \varphi$ é verdadeira quando ϕ for verdadeira sem a necessidade que φ seja verdadeira;

R **(*Release*):** A fórmula $\phi R \varphi$ é verdadeira quando φ for verdadeira até o momento que ϕ também for verdadeira, assim liberando (*Release*) φ.

A.3 Quantificadores de Caminho

E **(*Exist*):** A fórmula $E\phi$ indica que existe (*Exist*) (pelo menos) uma execução, a partir do estado atual, que satisfaz ϕ;

A **(*All*):** A fórmula $A\phi$ indica que todas (*All*) as execuções a partir do estado atual satisfazem a propriedade ϕ.

A.4 As Lógicas Temporais LTL, CTL e CTL*

Pnueli (1977) propôs a lógica *Linear Temporal Logic* (LTL). Esta linguagem considera que os comportamentos modelados seguem uma sequência linear de estados através do tempo. A cada instante, há apenas uma linha de tempo real futura que irá ocorrer. Tradicionalmente, esse cronograma é definido como começar "agora", no passo de tempo atual, e progredindo infinitamente para o futuro. As fórmulas LTL são compostas dos operadores lógicos e temporais citados acima.

Computational Tree Logic (CTL), proposta inicialmente por Clarke e Emerson (1981), é um lógica temporal ramificada que considera que os comportamentos estruturados em árvore podem seguir vários caminhos através do tempo. Uma linha do tempo ramificada inicia no passo de tempo atual, e pode progredir para qualquer um dos

muitos possíveis linhas de tempo futuras. Em CTL, adiciona-se quantificadores de caminho na sintaxe para expressar se a fórmula deve seguir todos os caminhos ou pelo menos um caminho existente. Portanto, as fórmulas CTL são compostas dos operadores lógicos e temporais, e dos quantificadores de caminho apresentados acima.

Emerson and Halpern (1983) foram os primeiros a propor a lógica CTL*. Esta lógica combina as sintaxes das lógicas LTL e CTL. Ela inclui todos os operadores de LTL e os quantificadores de CTL, porém sem a necessidade de ter os operadores temporais em pares com os quantificadores de caminho. Ambas, LTL e CTL são subconjuntos próprios de CTL*, com todas as combinações de fórmulas LTL e CTL. Desta maneira, tronando a lógica CTL* mais expressiva que LTL e CTL combinados, trazendo um grande custo computacional.

Anexo B

Algoritmos On-the-Fly para Jogos Temporizados

O algoritmo a ser apresentado apenas realiza a síntese da estratégia em relação à propriedade de alcançabilidade, chamado SOTFTR [12, 13]. As generalizações deste algoritmo estendem para outras propriedades, alterando basicamente o algoritmo retroativo para o encontro do conjunto dos estados vencedores.

B.1 Algoritmo Simbólico On-the-Fly para Jogos Temporizados de Alcançabilidade

A análise do autômato temporizado é baseada na exploração do grafo, o *grafo de simulação*, onde os nodos são *estados simbólicos*. Um estado simbólico é um par (l, Y), onde $l \in L$ e Y é uma zona de $\mathbb{R}^X_{\geq 0}$.

Sejam $Z \subseteq S$ e $a \in \Sigma$, define-se os a-sucessores e os a-predecessores de S, respectivamente.

$$\begin{aligned} Post_a(Z) &= \{(l', x') | \exists (l, x) \in Z, (l, x) \xrightarrow{a} (l', x')\} \\ Pred_a(Z) &= \{(l, x) | \exists (l', x') \in Z, (l, x) \xrightarrow{a} (l', x'). \end{aligned} \tag{B.1}$$

Desta maneira, sendo o conjunto dos estados sucessores de Z definido por $Z^{\nearrow}$, apresenta-se o algoritmo SOTFTR:

Algoritmo 1 SOTFTR: Symbolic On-The-Fly Algo. for Timed Reachability Games. [12, 13]

Inicialização

$Passed \leftarrow Z_0$ where $Z_0 = (l_0, 0)^{\nearrow}$;
$ToExplore \leftarrow \{(Z_0, \alpha, Z') | Z' = Post_\alpha(Z_0)^{\nearrow}\}$;
$ToBackPropagate \leftarrow \emptyset$;
$Win[Z_0] \leftarrow Z_0 \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
$Depend[Z_0] \leftarrow \emptyset$;

Main

while $((ToExplore \cup ToBackPropagate \neq \emptyset) \wedge (l_0, 0) \notin\neq Win[Z_0]))$ **do**
{// selecionar uma transição de $ToExplore$ ou $ToBackPropagate$}
$e = (Z, \alpha, Z') \leftarrow pop(ToExplore)$ ou $pop(ToBackPropagate)$;
if $Z' \notin Passed$ **then**
$Passed \leftarrow Passed \cup \{Z'\}$;
$Depend[Z'] \leftarrow \{(Z, \alpha, Z')\}$;
$Win[Z'] \leftarrow Z' \cap (\{Goal\} \times \mathbb{R}^X_{\geq 0})$;
if $Win[Z'] \subsetneq Z'$ **then**
$ToExplore \leftarrow ToExplore \cup \{(Z', \alpha, Z'') | Z'' = Post_\alpha(Z')^{\nearrow}$;
end if
if $Win[Z'] \neq \emptyset$ **then**
$ToBackPropagate \leftarrow ToBackPropagate \cup \{e\}$;
end if
else
{// Se $T \in Passed$, assume-se $Win[T] = \emptyset$}
$Good \leftarrow Win[Z] \cup \bigcup_{\overset{c}{\rightarrow} T} Pred_c(Win[T])$;
$Bad \leftarrow \bigcup_{S \overset{u}{\rightarrow} T} Pred_u(T \backslash Win[T]) \cap S$;
$Win^* \leftarrow Pred_t(Good, Bad)$;
if $Win[Z] \subsetneq Win^*$ **then**
$Waiting \leftarrow Waiting \cup Depend[Z]$;
$Win[Z] \leftarrow Win^*$;
end if
$Depend[Z'] \leftarrow Depend[Z'] \cup \{e\}$;
end if
end while

Referências Bibliográficas

[1] Ali Ahmadzadeh, Nader Motee, Ali Jadbabaie, and George Pappas. Multi-vehicle path planning in dynamically changing environments. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, pages 2148–2153, Piscataway, NJ, USA, 2009. IEEE Press.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2 – 34, 1993. ISSN 0890-5401.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] Rajeev Alur, Joel M. Esposito, M. Kim, Vijay Kumar, and Insup Lee. Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination. In *Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume I - Volume I*, FM '99, pages 212–232, London, UK, 1999. Springer-Verlag.

[5] Michael Skipper Andersen, Rune Strøm Jensen, Thomas Bak, and Michael Melholt Quottrup. *Motion Planning in Multi-robot Systems - Using Timed Automata*. Elsevier, 2004.

[6] Isaac Asimov. *I, Robot*. Voyager, September 1968. ISBN 0586025324.

[7] Gerd Behrmann, Alexandre David, and Kim Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 33–35. Springer Berlin / Heidelberg, 2004.

[8] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. UPPAAL-Tiga: Timed Games for Everyone. In Luca Aceto and Anna Ingolfdottir, editors, *Proceedings of the 18th Nordic Workshop on Programming Theory (NWPT'06), Reykjavik, Iceland*. Reykjavik University, 2006.

[9] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer Berlin / Heidelberg, 2004.

[10] B.A. Brandin and W.M. Wonham. Supervisory control of timed discrete-event systems. *Automatic Control, IEEE Transactions on*, 39(2):329 –342, feb 1994.

[11] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387333320.

[12] Franck Cassez. Efficient on-the-fly algorithms for partially observable timed games. In Jean-François Raskin and P. Thiagarajan, editors, *Formal Modeling and Analysis of Timed Systems*, volume 4763 of *Lecture Notes in Computer Science*, pages 5–24. Springer Berlin / Heidelberg, 2007.

[13] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analy-

sis of timed games. In *IN CONCUR 05, LNCS 3653*, pages 66–80. Springer, 2005.

[14] Luiz Chaimowicz, Vijay Kumar, and Mario F. M. Campos. A paradigm for dynamic coordination of multiple robots. *Autonomous Robots*, 17:7–21, 2004.

[15] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[16] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag. ISBN 3-540-11212-X.

[17] H. Costelha and P. Lima. *Autonomous Agents*, chapter Petri Net Robotic Task Plan Representation: Modelling, Analysis and Execution, pages 65–90. INTECH, 2010.

[18] José E. R. Cury. Teoria de controle supervisório de sistemas a eventos discretos. *V Simpósio Brasileiro de Automação Inteligente*, 2001.

[19] Alexandre David, Kim Guldstrand Larsen, Shuhao Li, and Brian Nielsen. Cooperative testing of uncontrollable timed systems. *Electronic Notes in Theoretical Computer Science*, 220(1), 2008.

[20] E. Allen Emerson and Joseph Y. Halpern. "sometimes"and "not never"revisited: on branching versus linear time (preliminary report). In *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, POPL '83, pages 127–140, New York, NY, USA, 1983. ACM. ISBN 0-89791-090-7.

[21] G.E. Fainekos, H. Kress-Gazit, and G.J. 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, april 2005.

[22] Matthieu Gallien, Fahmi Gargouri, Imen Kahloul, Moez Krichen, Thanh H. Nguyen, Saddek Bensalem, and Félix Ingrand. D'une approche modulaire à une approche orientée composant pour le développement de systèmes autonomes : Défis et principes. In *3rd National Conference on Control Architectures of Robots (CAR 2008)*, Bourges, France, 19- 30 May 2008.

[23] Reinhard Gotzhein. Temporal logic and applications: a tutorial. *Comput. Netw. ISDN Syst.*, 24:203–218, May 1992.

[24] Christopher Griffin. Game theory: Penn state math 486 lecture notes, 2011. URL `http://www.personal.psu.edu/cxg286/Math486.pdf`.

[25] Michael Huth and Mark Ryan. *Logic in Computer Science: modelling and reasoning about systems (second edition)*. Cambridge University Press, 2004.

[26] Félix Ingrand, Simon Lacroix, Solange Lemai-Chenevier, and Frederic Py. Decisional autonomy of planetary rovers: Research articles. *J. Field Robot.*, 24:559–580, July 2007.

[27] Luca Iocchi, Markus Hannebauer, Jan Wendler, Enrico Pagello, Daniele Nardi, and Massimiliano Salerno. Reactivity and deliberation: A survey on multi-robot systems. In *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, volume 2103 of *Lecture Notes in Computer Science*, pages 9–32. Springer Berlin / Heidelberg, 2001.

[28] Sylvain Joyeux, Rachid Alami, and Simon Lacroix. A plan manager for multi-robot systems. In Christian Laugier and Roland

Siegwart, editors, *Field and Service Robotics*, volume 42 of *Springer Tracts in Advanced Robotics*, pages 443–452. Springer Berlin / Heidelberg, 2008.

[29] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. Where's waldo? sensor-based temporal logic motion planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3116 – 3121, april 2007.

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

[31] Xinxin Liu and Scott Smolka. Simple linear-time algorithms for minimal fixed points. In Kim Larsen, Sven Skyum, and Glynn Winskel, editors, *Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 53–66. Springer Berlin / Heidelberg, 1998.

[32] Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In Ernst Mayr and Claude Puech, editors, *STACS 95*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer Berlin / Heidelberg, 1995.

[33] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541 –580, apr 1989.

[34] Jonatas Pavei, Jean-Marie Farines, and José Cury. Coordenação de sistemas multi-robôs utilizando autômato-jogo temporizado. In *SBAI 2011*, São João del-Rei, sep 2011.

[35] Nir Piterman and Amir Pnueli. Synthesis of reactive(1) designs. In *In Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI 06)*, pages 364–380. Springer, 2006.

[36] A. Pnueli, E. Asarin, O. Maler, and J. Sifakis. Controller Synthesis for Timed Automata. In *Proc. System Structure and Control*. Elsevier, 1998.

[37] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[38] Max H. Queiroz. Controle supervisório modular de sistemas de grande porte. Master's thesis, Departamento de Energia Elétrica, UFSC, Florianópolis, SC, 2000.

[39] Max H. Queiroz and José E. R. Cury. Modular control of composed systems. In *In Proceedings of the American Control Conference*, pages 4051–4055, 2000.

[40] Max H. Queiroz and José E. R. Cury. Modular supervisory control of large scale discrete event systems. In *In Discrete Event Systems: Analysis and Control.*, pages 103–110. Kluwer Academic, 2000.

[41] Max H. Queiroz and José E. R. Cury. Controle supervisório modular de sistemas de manufatura. In *SBA: Controle e Automação*, volume 13, pages 123 – 133. scielo, 08 2002.

[42] M.M. Quottrup, T. Bak, and R.I. 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, may 2004.

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

[44] Kristin Y. Rozier. Linear temporal logic symbolic model checking. *Computer Science Review*, 5(2):163 – 203, 2011.

[45] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.

[46] Kai C. Wong and W. Murray Wonham. Modular control and coordination of discrete-event systems. *Discrete Event Dynamic Systems*, 8:247–297, October 1998.

[47] W. Wonham and P. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems (MCSS)*, 1(1):13–30, 1988. 10.1007/BF02551233.

[48] W.M. Wonham. *Supervisory Control of Discrete Event Systems*. Systems Control Group, University of Toronto, Toronto, Canadá, 2010.

[49] V. A. Ziparo, L. Iocchi, D. Nardi, P. F. Palamara, and H. Costelha. Petri net plans: a formal model for representation and execution of multi-robot plans. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, AAMAS '08, pages 79–86, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.