UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÉMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

MAICON PAULO ASSMANN

CONTROLE SUPERVISÓRIO COM ESTRUTURAS CONCORRENTES DE REFINAMENTOS SUJEITAS À DEPENDÊNCIA DE CONTEXTO

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2018

MAICON PAULO ASSMANN

CONTROLE SUPERVISÓRIO COM ESTRUTURAS CONCORRENTES DE REFINAMENTOS SUJEITAS À DEPENDÊNCIA DE CONTEXTO

Trabalho de Conclusão de Curso 2, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná - UTFPR, Câmpus Pato Branco, como requisito parcial para obtenção do título de Engenheiro da Computação.

Orientador: Prof. Dr. Marcelo Teixeira

Coorientador: Eng. Marcelo Rosa

PATO BRANCO

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Às 8 horas e 20 minutos do dia 21 de junho de 2018, na sala V107, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Marcelo Teixeira (orientador), Marcelo Rosa (coorientador), Cesar Rafael Claure Torrico e Marco Antonio de Castro Barbosa para avaliar o trabalho de conclusão de curso com o título **Controle supervisório com estruturas concorrentes de refinamentos sujeitas à dependência de contexto**, do aluno **Maicon Paulo Assmann**, matrícula 01435540, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____	_____
Marcelo Teixeira Orientador (UTFPR)	Marcelo Rosa Coorientador (UTFPR)
_____	_____
Cesar Rafael Claure Torrico (UTFPR)	Marco Antonio de Castro Barbosa (UTFPR)
_____	_____
Profa. Beatriz Terezinha Borsoi Coordenador de TCC	Prof. Pablo Gauterio Cavalcanti Coordenador do Curso de Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

ASSMANN, Maicon Paulo. Controle supervisório com estruturas concorrentes de refinamentos sujeitas à dependência de contexto. 2018. 49 f. Trabalho de Conclusão de Curso 2 - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2018.

A Teoria de Controle Supervisório (TCS) fundamenta a síntese de controladores para Sistemas a Eventos Discretos. Ainda que ao longo de seus 30 anos de existência a TCS tenha sido estendida e melhorada em vários aspectos, ela ainda preserva uma característica que limita a sua aplicação sobre problemas industriais reais, que se refere à complexidade de modelagem. Nesse sentido, a ideia do refinamento de eventos foi recentemente proposta na literatura como forma de simplificar a modelagem e viabilizar a aplicação da TCS sobre uma gama mais ampla de problemas. No entanto, existem classes de processos sobre os quais a ideia de refinamento não se aplica diretamente. Por exemplo, os processos que contemplam mais de uma camada de refinamentos, no mesmo problema e contexto de controle. Para esses casos, a ideia de refinamento deve ser estendida para permitir e distinguir múltiplos conjuntos de refinamentos caracterizados por dependências de contexto. Este trabalho propõe uma abordagem formal que permite criar qualquer quantidade de refinamentos dependentes entre si, os quais podem então ser usados na síntese de controladores com distinguidores de eventos. A abordagem é ilustrada por meio de exemplos, em particular o de um sistema de manufatura com retrabalho de peças e um *buffer* capaz de armazenar mais de 1 peça. Nesse exemplo, é mostrado que uma solução que controla em paralelo a quantidade de ciclos e os limites do *buffer*, seria bastante custosa de ser modelada. Já usando o método proposto, mostra-se que ela se torna bastante simples e intuitiva.

Palavras-chave: Sistemas a eventos discretos, Controle Supervisório, Distinguidores, Múltiplos refinamentos.

ABSTRACT

ASSMANN, Maicon Paulo. Supervisory control with concurrent structures of refinaments subject to context dependency. 2018. 49 f. Trabalho de Conclusão de Curso 2 - Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2018.

The Supervisory Control Theory (SCT) defines the synthesis of controllers for Discrete Events Systems. Although within its 30 years of existence the SCT has been extended and improved in several ways, it still preserves a feature that limits its application on real industrial problems, that refers to the modeling complexity. In this sense, the idea of events refinement has been recently proposed in the literature as a way to simplify the modeling and to enable the application of the SCT over a wider range of problems. However, there are classes of processes over which the idea of refinement does not apply directly. For example, processes that include more than one layer of refinements in the same problem and control context. For these cases, the idea of refinement should be extended to allow and distinguish multiple sets of refinements characterized by context dependencies. This work proposes a formal approach that allows creating any number of refinements dependent from each other, which can be used, then, in the synthesis of controllers with event distinctions. The approach is illustrated using examples, in particular, a manufacturing system with rework and buffering of workpieces. In this example, is demonstrated a solution that controls in parallel the number of cycles and the buffer limits would be very hard to be modeled. Yet, using the proposed method, it is shown that the modeling can become quite simpler and intuitive.

Keywords: Discrete-Event Systems, Supervisory Control, Distinguishers, Multiple refinements.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho explora a síntese de controladores para sistemas a eventos discretos. Ocorre que alguns processos industriais contemplam uma grande quantidade de componentes inter-relacionados e, em geral, esses componentes são dependentes uns dos outros. Como tal, eles precisam ser representados de modo concorrente, o que pode tornar extensa e complexa a modelagem de requisitos. Dependendo do caso, o modelo de uma única especificação pode requerer centenas ou milhares de estados para ser construído de maneira minimamente restritiva.

Na literatura, algumas abordagens são propostas para simplificar o problema de modelagem. O refinamento de eventos, por exemplo, permite que se conheça diferentes instâncias de um mesmo evento no sistema, uma para cada circunstância em que esse evento pode ocorrer. Isso tende a atribuir simplicidade à modelagem por autômatos.

No entanto, a abordagem com refinamentos esbarra na limitação de poder simplificar a modelagem de um único requisito complexo. Muitas vezes, pode ocorrer que mais de um requisito sofrem do mesmo problema de ser complexo. Além disso, dois ou mais requisitos complexos de modelagem podem inclusive estar aninhados no mesmo contexto operacional do sistema a ser modelado. Desse modo, a modelagem de cada requisito pode compartilhar de eventos em comum, mas cada um com uma estrutura distinta de refinamentos, eventualmente dependentes.

Portanto, este trabalho explora o fato de que, em certos casos, dois ou mais requisitos complexos se integram no mesmo processo e requerem a integração de mais de um mecanismo de refinamento.

1.1 CONSIDERAÇÕES INICIAIS

Segundo Cassandras e Lafortune (2008) uma grande parte dos sistemas modernos de automação possui uma dinâmica operacional que depende de ocorrência de eventos, chamados *Sistemas a Eventos Discretos* (SEDs). Os SEDs se caracterizam por possuírem um conjunto enumerável de estados e uma dinâmica guiada pela ocorrência assíncrona de eventos, em oposição aos sistemas dirigidos pelo tempo. Estas características acabam determinando a forma como o sistema pode ser modelado. Enquanto os sistemas dirigidos pelo tempo são normalmente tratados por equações diferenciais, os SEDs são mais naturalmente representados por meio de diagramas de transições de estados, facilitando mapear formal e intuitivamente os enlaces entre os elementos (eventos$\rightarrow$transições$\rightarrow$estado), descritores fundamentais do comportamento de SEDs.

Em se tratando de ambientes fabris, que geralmente envolvem inúmeros componentes de produção, existe um intenso interesse no desenvolvimento de tecnologias que permitam coordenar as atividades operacionais dos SEDs de maneira automática, ótima e com o mínimo custo. No entanto, sintetizar um controlador ótimo de maneira empírica tende a ser muito complexo e, especialmente para sistemas de grande porte, essa tarefa costuma ser humanamente intratável.

Uma alternativa para lidar com este problema pode ser por meio da *Teoria de Controle Supervisório* (TCS) (RAMADGE; WONHAM, 1987). Essa abordagem é estruturada sobre *Teoria de Autômatos Finitos* (AFs) e *Linguagens* (HOPCROFT *et al.*, 2001) e possibilita descrever formalmente a síntese de controladores ótimos, ou seja, controladores cuja ação sobre a planta se dá de maneira minimamente restritiva, não-bloqueante e em concordância com o conjunto de especificações.

1.2 JUSTIFICATIVA

Uma das principais vantagens da TCS é o seu formalismo, que conduz a um processo de síntese de controle que permite preservar aspectos de segurança no sistema. No entanto, a TCS enfrenta alguns problemas práticos que limitam a sua aplicabilidade na indústria. Um deles é que, para alguns problemas específicos, a modelagem de uma única especificação de controle pode exigir monoliticamente combinar milhares de estados, tornando esta tarefa inviável (TEIXEIRA, 2013).

O conceito de refinamento de informação sobre um SED tem sido proposto na literatura (BOUZON *et al.*, 2008; CURY *et al.*, 2015; TEIXEIRA *et al.*, 2014) como forma de simplificar o processo de modelagem de especificações de controle. Porém, essa abordagem esbarra no fato de que apenas um requisito de controle pode ser simplificado por vez, usando para isso de uma estrutura particular de refinamentos. Muitas vezes, pode ocorrer que mais de um requisito do sistema pode ser complexo em sua modelagem. Assim, quando eles se integram no mesmo contexto operacional do sistema a ser modelado, se torna necessário que mais de uma estrutura de refinamentos sejam integradas e coordenadas semanticamente.

1.3 OBJETIVOS

Os objetivos do trabalho estão estruturados como se segue.

1.3.1 OBJETIVO GERAL

Formalizar a integração entre diferentes mecanismos de refinamento de eventos dependentes, com o propósito de simplificar a modelagem de requisitos complexos de controle, que são encontrados em processos industriais.

1.3.2 OBJETIVOS ESPECÍFICOS

- Fundamentar e exemplificar a Teoria de controle Supervisório de Sistemas a Eventos Discretos (SEDs);
- Apresentar o problema de síntese de controladores para SEDs envolvendo o refinamento e a distinção de eventos;
- Simular no *software* Supremica a abordagem de refinamentos usando um exemplo em escala didática;
- Identificar situações na modelagem de requisitos que requeiram integrar mecanismos de refinamentos;
- Construir uma alternativa para a integração de mais de uma estrutura de refinamento de eventos;
- Comparar a abordagem em relação ao método sem refinamentos;
- Modelar o sistema proposto no *software* Supremica, para demostração das vantagens na utilização do modelo de refinamento distinguidor.
- Analisar os principais aspectos da simulação do sistema proposto sintetizado.

2 REFERENCIAL TEÓRICO

2.1 SISTEMAS A EVENTOS DISCRETOS

Na literatura a definição de *sistema* é apresentada de vários modos. Para Haykin e Veen (2001, p. 22) um sistema é uma entidade capaz de manipular um ou mais sinais de forma a realizar uma determinada função, resultando em novos sinais. Por sua vez Ogata (2010, p. 3) define que um sistema é uma combinação de componentes que atuam conjuntamente realizando um certo objetivo. Um sistema não é limitado a algo físico, podendo se referenciar a fenômenos abstratos, dinâmicos, que podem ser encontrados em diversas áreas (OGATA, 2010). Alguns exemplos de sistemas incluem a robótica, redes de computadores, sistemas operacionais, processo de manufatura e entre outros.

Na modelagem computacional de sistemas, suas propriedades são estruturadas sobre dois fundamentos básicos: o *estado*, que define o status do sistema em determinada situação; e o mecanismo de *transição de estados*, que caracteriza a evolução do sistema entre os estados. Basicamente, as transições podem ser dirigidas pelo tempo ou por eventos (TEIXEIRA, 2013).

No primeiro caso, a ocorrência de transições é mapeada continuamente no tempo, como ocorre em exemplos de sistemas de temperatura, pressão, frequência e entre outros. Para esses sistemas, a modelagem é mais natural por meio, por exemplo, de equações diferencias. Já nos casos em que a evolução do sistema é guiada por eventos, a modelagem por equações diferenciais não é adequada, devido ao fato de que a dinâmica de transição pode não depender do tempo, mas de eventos assíncronos (CASSANDRAS; LAFORTUNE, 2008). Aos sistemas que compartilham dessa dinâmica dá-se o nome de *Sistemas a Eventos Discretos* (SEDs) (CASSANDRAS; LAFORTUNE, 2008) e esses são, em geral, modelados por meio de diagramas de transições de estados.

2.2 MODELAGEM DE SED

Basicamente dois conceitos teóricos formam opções que orientam a modelagem de SEDs: *Autômatos* e *Linguagens*. Ambos são descritos na sequência.

2.2.1 TEORIA DOS AUTÔMATOS E LINGUAGENS

Uma linguagem é uma estrutura formal baseada em *eventos*, que são reunidos em um alfabeto Σ, assim como letras de um alfabeto de determinado idioma. Uma sequência concatenada de eventos leva à composição de cadeias. O conjunto de todas as cadeias possíveis de serem formadas com elementos de Σ é denotado por Σ^*. Por fim, uma linguagem qualquer L é definida como $L \subseteq \Sigma^*$ (CASSANDRAS; LAFORTUNE, 2008; HOPCROFT *et al.*, 2001).

Na prática, a modelagem por linguagens formais requer uma estrutura adicional que possa reconhecê-la a partir de uma visão mais próxima do engenheiro e do próprio sistema. *Expressões Regulares* são uma opção, mas em geral não se ajustam adequadamente à percepção do engenheiro, especialmente ao tratar de problemas extensos e complexos. O uso de *autômatos finitos*, surge então como alternativa.

Um *Autômato Finito* (AF) é uma estrutura formal constituída por diagramas de transições de estados, que permite modelar diversos problemas de um modo prático e intuitivo (CASSANDRAS; LAFORTUNE, 2008; TEIXEIRA, 2013). Uma definição formal para um AF pode ser apresentada como uma 5-tupla $\langle \Sigma, Q, q^{\circ}, Q^{\omega}, \rightarrow \rangle$, tal que:

- Σ é o alfabeto finito;
- Q é conjunto finito de estados;
- $q^o \in Q$ é estado inicial;

- $Q^w \subseteq Q$ é o subconjunto de estados marcados;
- $\rightarrow \subseteq Q \times \Sigma \times Q$ é a relação de transição de estados.

Denota-se por $q_1 \xrightarrow{\sigma} q_2$, uma transição do estado q_1 para o estado q_2 com o evento $\sigma \in \Sigma$. Por $G \xrightarrow{s} q$, denota-se que a $s \in \Sigma^*$ é possível em um dado autômato G, i.e., $q^o \xrightarrow{s} q$ para $q \in Q$ (TEIXEIRA, 2013).

Uma possível maneira de se representar a estrutura de um autômato graficamente é por meio de um grafo direcionado, de modo que os nós representam os estados e as arestas as transições entre os estados. A dinâmica das transições dos estados são desencadeadas diante a ocorrência de um determinado evento, associado a ela. O estado inicial é identificado por uma seta uniconectada apontado para ele e os estados marcados são denotados por círculos duplos, cuja significado desses estados está associado à ideia de tarefa completa ou aceitação (TEIXEIRA, 2013). Por exemplo, um autômato $G = \langle \Sigma_G, Q_G, q_G^\circ, Q_G^\omega, \rightarrow_G \rangle$, com $\Sigma = \{\alpha, \beta, \gamma\}$, $Q = \{q_0, q_1, q_2\}$, $q^o = q_0$, $Q^w = \{q_0\}$ e cuja função de transição de estados é dada por $q_0 \xrightarrow{\alpha} q_1$, $q_1 \xrightarrow{\gamma} q_2$, $q_2 \xrightarrow{\alpha} q_2$, $q_2 \xrightarrow{\beta} q_1$ e $q_1 \xrightarrow{\beta} q_0$, é representado graficamente pelo grafo da Figura 1.

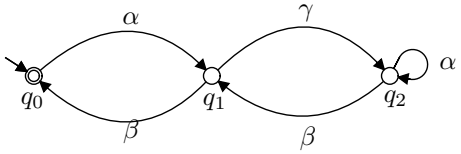

Figura 1 – Exemplo de *Autômato*

Assim, adotando autômatos e linguagens como formalismo para modelar SEDs, o primeiro passo é identificar os eventos e em seguida associá-los a uma estrutura que reflete uma versão abstrata do sistema. Ou seja, um modelo é normalmente uma abstração em relação a maneira de como o sistema é visto e interpretado, em contraste ao que é importante a ser contemplado no modelo abstraído (JAIN, 1991). Naturalmente, identificar tal abstração depende muito da experiência do especialista, por exemplo, identificar quais sinais devem ou não fazerem parte do modelo, quais sensores

são relevantes de serem modelados, quais situações observáveis em particular orientam a modelagem do sistema e entre outros requisitos relevantes.

A partir do autômato-exemplo G proposto, podem ser definidas duas linguagens em particular: a *linguagem gerada* e a *linguagem marcada*, definidas respectivamente por:

$$L(G) = \{s \in \Sigma^* | G \xrightarrow{s} q \in Q\} \text{ e}$$

$$L^w(G) = \{s \in \Sigma^* | G \xrightarrow{s} q \in Q^w\}.$$

De modo mais descritivo pode-se dizer que, a linguagem gerada $L(G)$ representa o conjunto de todas as cadeias possíveis de serem geradas a partir do autômato G, enquanto que a linguagem marcada $L^w(G)$ corresponde ao conjunto de todas as cadeias que alcançam estados marcados, tal que $L^w(G) \subseteq L(G)$ (CASSANDRAS; LAFORTUNE, 2008).

2.2.2 OPERAÇÕES SOBRE AUTÔMATOS E LINGUAGENS

Sabendo que as linguagens são conjuntos, por definição, então todas as operações convencionais sobre conjuntos (união, interseção, potência, entre outras) se aplicam a linguagem. Além dessas, outras operações podem ser definidas especificamente para lidar com elementos de linguagens (CURY, 2001; CASSANDRAS; LAFORTUNE, 2008; TEIXEIRA, 2013).

As *operações morfológicas* tem como principio estudar a estrutura de cadeias. Desta forma, seja Σ um alfabeto e $s = mnp$ uma cadeia qualquer, com $m, n, p \in \Sigma^*$, então:

- m é um prefixo de s;
- n é uma subcadeia de s;
- p é um sufixo de s.

A concatenação para duas linguagens, tal que $L_1, L_2 \subseteq \Sigma^*$, é

denotada por L_1L_2, e definida por:

$$L_1L_2 = \{s \in \Sigma^* | (s = s_1s_2) \wedge (s_1 \in L_1) \wedge (s_2 \in L_2)\}.$$

O prefixo-fechamento de uma linguagem $L \subseteq \Sigma^*$, denotado por $\overline{L}$, é definido por:

$$\overline{L} = \{s \in \Sigma^* | (\exists t \in \Sigma^*), st \in L\}.$$

Portanto, $\overline{L}$ consiste na linguagem que contém todos os prefixos de todas as cadeias contidas em L. Uma linguagem L é dita ser prefixo-fechada se $L = \overline{L}$, i.e., se qualquer prefixo de qualquer cadeia de L é um elemento de L.

Outra operação bastante útil no contexto do desenvolvimento de controladores para SEDs é a *composição síncrona*, cuja notação é dada por $\|$. Tal operação pode ser definida tanto para autômatos quanto para linguagens, sendo esperado que em ambos os casos se mantenha a equivalência de resultados (WONHAM, 2002). Por praticidade, esse trabalho define somente a operação $\|$ para autômatos, como segue.

Sejam dois autômatos $A = \langle \Sigma_A, Q_A, q_A^\circ, Q_A^\omega, \rightarrow_A \rangle$ e $B = \langle \Sigma_B, Q_B, q_B^\circ, Q_B^\omega, \rightarrow_B \rangle$, a composição síncrona entre A e B resulta no seguinte autômato:

$$A \parallel B = (\Sigma_A \cup \Sigma_B, Q_A \times Q_B, (q_A^o, q_B^o), Q_A^w \times Q_B^w, \rightarrow),$$

em que $\rightarrow$ é defina como:

- $(q_A, q_B) \xrightarrow{\sigma} (q'_A, q'_B)$, se $\sigma \in \Sigma_A \cap \Sigma_B$;
- $(q_A, q_B) \xrightarrow{\sigma} (q'_A, q_B)$, se $\sigma \in \Sigma_A \setminus \Sigma_B$;
- $(q_A, q_B) \xrightarrow{\sigma} (q_A, q'_B)$, se $\sigma \in \Sigma_B \setminus \Sigma_A$.

A *composição síncrona* pode ser estendida para n autômatos. Então, sejam os autômatos $G_i = \langle \Sigma_i, Q_i, q_i^\circ, Q_i^\omega, \rightarrow_i \rangle$, para $i = 1, 2, \cdots, n$, um modelo global G é dado mediante a composição síncrona de todos os

autômatos:

$$G = \Big|\Big|_{i=1}^{n} G_i, \text{ tal que } \Sigma = \bigcup_{i=1}^{n} \Sigma_i.$$

Conforme Wonham (2002), os comportamentos gerado e marcado da composição é tal que:

$$L(G) = \Big|\Big|_{i=1}^{n} L(G_i) \text{ e } L^w(G) = \Big|\Big|_{i=1}^{n} L^w(G_i).$$

Na modelagem de SEDs, a composição síncrona é fundamental por permitir que cada componente do sistema possa ser modelado individualmente por um autômato G_i que em tese, é mais simples do que seria se a modelagem fosse tratada como um todo pelo engenheiro. No contexto de controle de SEDs, algumas propriedades importantes são definidas em relação a conceito de *acessibilidade*, tais propriedades são definidas como:

- *Estado acessível*: Um estado $q \in Q_G$ é dito ser acessível se $\exists s \in \Sigma^*$, tal que $q_G^o \xrightarrow{s} q$ ou, equivalentemente, se $G \xrightarrow{s} q$.
- *Autômato acessível*: G é dito ser acessível se q é acessível, $\forall q \in Q_G$.
- *Autômato co-acessível*: G é dito ser co-acessível se cada cadeia $s \in L(G)$ é prefixo de uma cadeia marcada. Ou seja, se cada $s \in L(G)$ pode ser completado por algum $t \in \Sigma^*$ tal que $st \in L^w(G)$, isto é, $q_G^o \xrightarrow{st} q$ tal que $q \in Q_G^w$. Ou ainda, se $L(G) = \overline{L^w(G)}$.
- *Autômato não-bloqueante*: G é dito ser *não-bloqueio* se é co-acessível.
- *Autômato Trim*: G é dito ser trim, se ele é acessível e co-acessível.

Ressalta-se que a propriedade de *não-bloqueio* dos autômatos, é particularmente importante para o tema deste trabalho, está conectada ao conceito de acessibilidade, i.e., o comportamento gerado por um autômato G, é dito ser não-bloqueante se e somente se o autômato é co-acessível.

2.3 CONTROLE DE SED

O controle de SEDs consiste basicamente em definir uma sequência lógica de execução dentro do comportamento possível do sistema. Essa lógica pode ser implementada usando os próprios formalismos empregados na modelagem. A literatura provê algumas alternativas, como *Álgebra e Dióides* (Max-Plus) (SPACEK *et al.*, 1996; SCHUTTER; BOOM, 2008), a *Lógica Temporal* (PNUELI, 1977; KUMAR; GARG, 2005), as *Redes de Petri* (GIUA; DICESARE, 1994) e, de interesse particular, os *Autômatos e Linguagens* (CASSANDRAS; LAFORTUNE, 2008; WONHAM; RAMADGE, 1987).

Nessas abordagens, a lógica de controle é implementada de maneira essencialmente empírica, ou seja, o modelo é produzido pelo engenheiro e então é checado a *posteriori*, para a análise das propriedades de interesse, tais como *não-bloqueio*, *vivacidade*, *limitação* e entre outras. Diferentemente, a *Teoria do Controle Supervisório* (TCS), apresentada a seguir, define um método automático para extrair do modelo de um SED o comportamento que se deseja sob controle.

2.3.1 TEORIA DE CONTROLE SUPERVISÓRIO

Na TCS, o comportamento de um SED e suas especificações de controle são expressados individualmente por meio de um conjunto de autômatos. Então, estes autômatos são compostos por ($||$) e uma operações matemáticas sintetiza o comportamento desejado de maneira minimamente restritivas e não-bloqueante, conforme os requisitos definidos.

Ao modelo que identifica o comportamento dos componentes de um SED, dá-se o nome de *planta*. Uma planta está em *malha aberta*, quando ela não sofre nenhuma interferência externa ou ação de controle. Um exemplo é apresentado na Figura 2.

A Figura 2 mostra o modelo de um SED simples, considerando a operação de duas máquinas M_1 e M_2, modeladas respectivamente por G^1 e G^2,

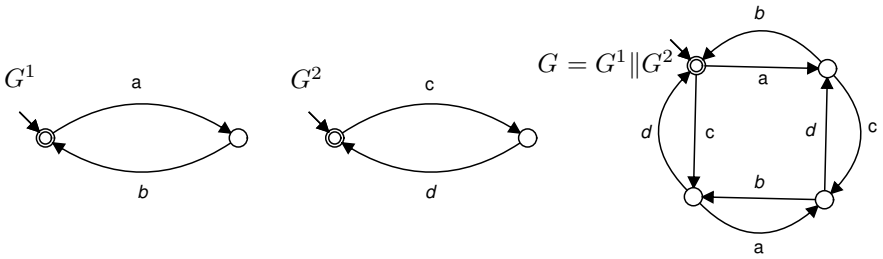

Figura 2 – Modelagem da planta de um SED

bem como a composição síncrona entre as máquinas, que resulta no modelo da planta em malha aberta $G = G^1 \| G^2$. O comportamento das máquinas M_1 e M_2 é descrito em termos de eventos de início (a e c) e final de operação (b e d).

Observa-se que o comportamento de G não impõe nenhuma coordenação sobre a planta, podendo assim não atender ao comportamento que se espera na prática. Para restringir a planta, ela é composta com modelo de *especificações*, os quais implementam uma ação proibitiva nos eventos do sistema. Ao comportamento desejado para o sistema sobre controle dá-se o nome de *malha fechada*.

Para o exemplo apresentado na Figura 2, uma possível especificação seria considerar a operação sequencial das máquinas M_1 e M_2, de forma que M_2 só pudesse iniciar após o fim de operação em M_1, e M_1 não pudesse finalizar duas vezes seguidas sem que M_2 iniciasse. Esse requisito de controle é expressado pelo modelo da especificação E na Figura 3, no qual no estado inicial é desabilitado o evento c, o inicio da operação da máquina M_2, e o evento b, o final da operação da máquina M_1 é desabilitado no outro estado.

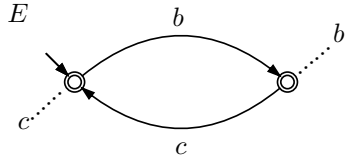

Figura 3 – Modelagem de uma especificação

Note que, a especificação E restringe G usando apenas os eventos c e b, os demais são considerados sempre habilitados e, portanto, suas ocorrências dependem da planta.

2.3.2 CONTROLABILIDADE

Para o exemplo anterior, a composição $E\|G$ modela a ação de controle de E sobre G. Portanto, é simples observar a utilidade dessa composição para implementação do controlador para G. Porém uma planta pode conter eventos que não podem ser diretamente desabilitados pelo controlador, como por exemplo a quebra de um equipamento, o sinal de um sensor, etc. Esses são eventos involuntários, que independem dos princípios de controle.

Nessa perspectiva, a TCS (RAMADGE; WONHAM, 1989) define uma estrutura de controle para a planta G, que particiona o conjunto de eventos $\Sigma = \Sigma_{\mathrm{c}} \cup \Sigma_{\mathrm{u}}$, em que Σ_{c} inclui os *eventos controláveis*, cuja a ocorrência pode ser desabilitada na planta, e o Σ_{u} que define os *eventos não-controláveis*, cuja a ocorrência não pode ser diretamente evitada.

Então, o problema de controle consiste em calcular uma versão (a *versão máxima*, uma vez que se busca máxima permissividade) de $E\|G$ que reconhece a impossibilidade de desabilitar eventos em Σ_{u}. A estrutura dá-se o nome de *supervisor*.

Formalmente, um supervisor é um mapa $S : L(G) \to 2^{\Sigma}$, associado a uma linguagem $L_S \subseteq L^w(G)$ que, após qualquer cadeia $s \in L(G)$, observa eventos elegíveis na planta G e informa, dentre eles, quais devem ser habilitados (RAMADGE; WONHAM, 1989). A malha de controle para esse cenário é ilustrada na Figura 4.

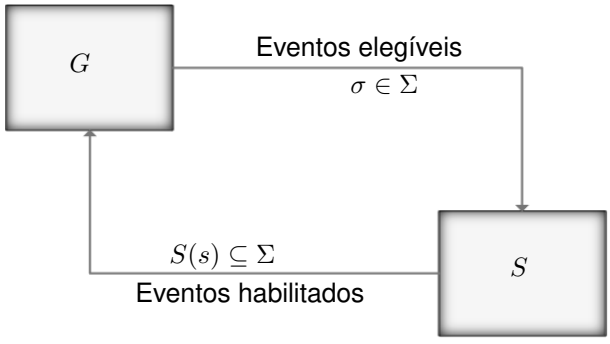

Figura 4 – Fluxo de controle

O conjunto de cadeias de $L(G)$ que sobrevive sob controle representa o *comportamento gerado em malha fechada*. O *Problema de Controle Supervisório* (PCS) se resume, então, em encontrar um supervisor S tal que sua ação sobre G satisfaça o conjunto de especificações K e, para isso, desabilite apenas eventos controláveis (RAMADGE; WONHAM, 1989).

A solução de tal problema passa pelo conceito de *controlabilidade*. Uma linguagem $K \subseteq \Sigma^*$ é *controlável* em relação a uma linguagem prefixo-fechada $L(G)$ se $\overline{K}\Sigma_{\mathrm{u}} \cap L(G) \subseteq \overline{K}$. Ou seja, se após qualquer prefixo s, um evento $\mu \in \Sigma_{\mathrm{u}}$ é elegível em $L(G)$, então μ não é desabilitado, i.e., $s\mu \in \overline{K}$.

Quando K não é controlável, é necessário reduzi-la à sua *máxima sublinguagem controlável*, definida por:

$$\sup\mathcal{C}(K,G) = \{\bigcup K' \subseteq K \mid K' \text{ é controlável em relação a } L(G)\}.$$

O processo de computar $\sup\mathcal{C}(K,G)$ é conhecido como *síntese* (CASSANDRAS; LAFORTUNE, 2008) e nada mais é do que uma operação matemática que implementa a noção de controlabilidade, i.e., o algoritmo de síntese extrai de K um sub-modelo K' que trata da impossibilidade de interferir diretamente em eventos de Σ_{u}.

Assim, $\sup\mathcal{C}(K,G)$ representa o comportamento menos restritivo possível de ser implementado por um supervisor S ao controlar G, respeitando a especificação K. Se $\sup\mathcal{C}(K,G)$ for ainda não-bloqueante, então $L^{\omega}(S/G) = \sup\mathcal{C}(K,G)$ e $L^{\omega}(S/G) = \overline{\sup\mathcal{C}(K,G)}$ é uma solução *ótima* para o PCS.

2.3.3 ESTUDO DE CASO

Considere um SED correspondente ao pequeno processo de manufatura da Figura 5, que é composto por duas máquinas, M_1 e M_2, operando de maneira sequencial, tal que as duas máquinas são interligadas por um *buffer* B capaz de suportar o armazenamento de até 3 peças empilhadas, seguindo o padrão de pilha, i.e., a última empilhada é a primeira a sair.

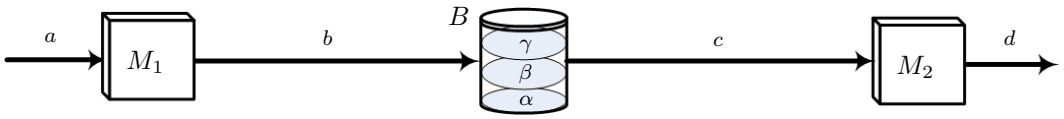

Figura 5 – Processo de manufatura com armazenamento de peças.

2.3.3.1 MODELAGEM DA PLANTA

Para esse processo, o modelo da planta é dado pelos mesmos autômatos mostrados na Figura 2, reproduzidos abaixo (Figura 6) por conveniência, já que o mesmo exemplo será explorado mais adiante, no contexto de outras políticas de modelagem.

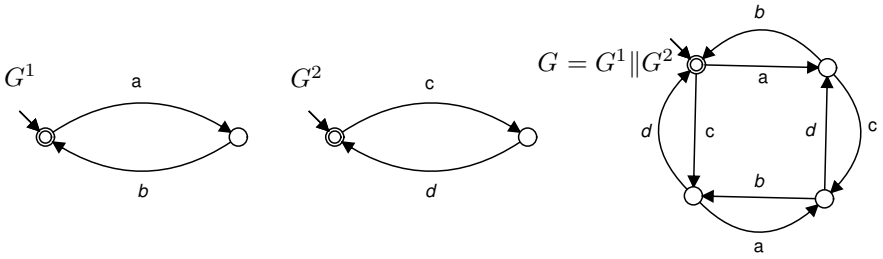

Figura 6 – Modelo da planta para o exemplo

A planta em malha aberta é dada pela composição $G = G^1 \| G^2$, tal que os autômatos modelam basicamente o início e o final de operações em G^1 e G^2.

2.3.3.2 MODELAGEM DA ESPECIFICAÇÃO

Para fechar a malha de controle sobre G, são necessárias especificações para esse comportamento. Considere que o objetivo de controle para este exemplo é evitar o *underflow* e o *overflow* do *buffer*. Uma especificação E que implementa essas restrições é apresentada na Figura 7.

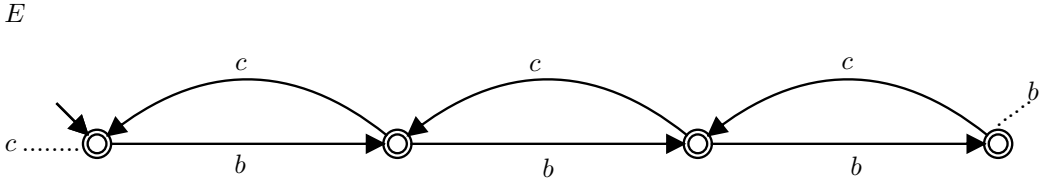

Figura 7 – Modelo da especificação de controle E

Na especificação E, que tem como objetivo controlar G, o evento c no estado inicial é desabilitado, já que a sua ocorrência antes de um evento b resultaria no *underflow*. Após empilhar a terceira peça no *buffer* B, o evento b é desabilitado, para que não ocorra mais, o que acabaria levando à condição de *overflow* no *buffer*.

2.3.3.3 SÍNTESE DO CONTROLADOR

Com os modelos da planta G e da especificação E, pode-se obter o comportamento desejado para o sistema em malha fechada, dado por $K = G \| E$. Note que, nesse caso o comportamento desejado é inconsistente com a ação controlável do sistema na prática, uma vez que E está desabilitando b, um evento não-controlável, no estado que representa a terceira posição do *buffer*. A fim de resolver tal inconsistência, o supervisor é calculado $\sup\mathcal{C}(K, G)$, desabilitando a quarta ocorrência do evento a (contida em uma cadeia) na planta de forma a evitar os maus estados contido em K. A Tabela 1 contextualiza em número de estados os modelos utilizados na síntese do supervisor S.

Tabela 1 – Número de estados dos modelos usados na síntese

G	E	K	S
4	$n+1$	$4n+4$	$4n+2$
G	E	K	S
4	4	16	14

A primeira linha mostra o número de estados para um *buffer* com n posições, enquanto a segunda linha mostra o caso particular do controle de 3 posições. Note que, o esforço envolvido na modelagem de E está diretamente associado ao tamanho do *buffer*. A seguir será apresentado uma alternativa que elimina a dependência do modelo E com tamanho do *buffer*, de modo a tornar a sua modelagem mais simples, e ainda assim, levar a uma solução equivalente.

2.4 TCS COM DISTINGUIDORES

O exemplo apresentado ilustra um caso em que se tem que memorizar o comportamento do sistema via autômato, o que em geral é associado a modelos extensos e por vezes complexos. Note que o número de estados do modelo da especificação E depende do tamanho do *buffer* n, o que pode dificultar em aspectos práticos da TCS, como a implementação e a computação. Em outros casos, como os mostrados em (CURY *et al.*, 2015; TEIXEIRA *et al.*, 2014), a dificuldade de modelagem usando a TCS clássica pode chegar à intratabilidade do problema. Com isso, uma opção apresentada na literatura (CURY *et al.*, 2015; BOUZON *et al.*, 2008) para simplificar a modelagem de problemas na TCS, é usando o conceito de *distinguidores*.

Distinguidores são tipos especiais de sensores que, ao serem associados ao modelo de um SED, permitem prover mais informações sobre certos eventos do sistema, o que tende a simplificar e viabilizar a modelagem das especificações, no qual poderia ser complexas sem tais informações. Considere um SED, modelado com eventos em Σ. Na abordagem com distinguidores assume-se que cada evento $\sigma \in \Sigma$ torna-se uma máscara para um conjunto não-vazio de eventos refinados, denotado por Δ^σ, escolhidos de maneira a identificar diferentes instâncias em que σ pode ocorrer na planta. Logo, Σ é um conjunto de máscaras para o alfabeto refinado $\Delta = \Delta_c \cup \Delta_u$, onde $\Delta_c = \bigcup_{\sigma \in \Sigma_c} \Delta^\sigma$ e $\Delta_u = \bigcup_{\sigma \in \Sigma_u} \Delta^\sigma$ (TEIXEIRA, 2013; CURY *et al.*, 2015).

A relação entre os alfabetos Σ e Δ, pode ser definida por mapas. O *mapa mascarador* $\Pi : \Delta^* \to \Sigma^*$ é definido recursivamente por:

$$\Pi(\epsilon) = \epsilon,$$
$$\Pi(t\delta) = \Pi(t)\sigma, \text{ para } t \in \Delta^*, \delta \in \Delta^\sigma \text{ e } \sigma \in \Sigma.$$

Ou seja, Π relaciona cada cadeia refinada em Δ^* a uma cadeia em

Σ^*. Esse mapa pode ser estendido para linguagens $L_d \subseteq \Delta^*$ por:

$$\Pi(L_d) = \{s \in \Sigma^* | \exists t \in L_d, \Pi(t) = s\}.$$

Já o mapeamento inverso é definido por $\Pi^{-1} : \Sigma^* \to 2^{\Delta^*}$ tal que:

$$\Pi^{-1}(s) = \{t \in \Delta^* | \Pi(t) = s\}.$$

Por fim, o efeito de um distinguidor $D : \Sigma^* \to 2^{\Delta^*}$ sobre uma linguagem $L \subseteq \Sigma^*$ pode ser entendido como:

$$D(L) = \Pi^{-1}(L) \cap L_D$$

Ou seja, o distinguidor D projeta cadeias de L em cadeias em Δ^*, por meio do mapa Π^{-1}, e distingue/filtra a ocorrência dessas cadeias através de uma linguagem L_D. Quando cada cadeia $s \in \Sigma^*$ é mapeada em exatamente uma cadeia $t \in \Delta^*$, i.e., $|D(s)| = 1$, o distinguidor D é dito ser *preditível* e esse é o caso particular exato que é considerado nesse trabalho.

Considerando um SED modelado por um autômato G, o efeito do distinguidor D sobre $L(G)$ (a mesma notação é assumida para o autômato G) é dado por:

$$D(L(G)) = \Pi^{-1}(L(G)) \cap L_D;$$
$$D(L^\omega(G)) = \Pi^{-1}(L^\omega(G)) \cap L_D.$$

Agora, sejam:

- G_d o modelo que simboliza a planta refinada do SED, tal que $L(G_d) = D(L(G))$ e $L^\omega(G_d) = D(L^\omega(G_d))$; e
- E_d uma versão da especificação E, só que modelada com eventos em Δ, de modo que ambas expressam os mesmos requisitos de controle, ainda que E_d disponha de eventos refinados.

Nesse cenário, pode ser assumido que $E_d \cap L^\omega(G_d) = K_d = D(K)$,

para $K = E \cap L^{\omega}(G)$, e o PCS pode então ser reintroduzido (PCS-D) para que se encontre um supervisor não-bloqueante $S_d : \Delta^* \to 2^{\Delta^*}$, tal que $L^{\omega}(S_d/G_d) \subseteq K_d$. É demostrado por (CURY *et al.*, 2015) que, se um distinguidor D é *preditível*, então:

$$\Pi(\sup\mathcal{C}(K_d, G_d)) = \sup\mathcal{C}(K, G);$$
$$\sup\mathcal{C}(K_d, G_d) = D(\sup\mathcal{C}(K, G)).$$

Ou seja, independente do alfabeto utilizando na síntese as duas versões do problema com e sem distinguidor acabam resultam na mesma solução de controle.

2.4.1 CONSTRUÇÃO DE UM DISTINGUIDOR

A construção em si de um distinguidor pode ser dada por algumas etapas de modelagem, no entanto, isso se remete mais a uma dependência da capacidade do engenheiro. E a fim de facilitar tal construção de distinguidores, foi realizado um guia de passos descrito por Teixeira (2013) para tal desenvolvimento, este guia de passos é apresentado a seguir.

(i) *Identificação de um conjunto inicial de eventos $\sigma \in \Sigma$ a serem refinados*: o primeiro passo para a construção de um distinguidor é identificar quais eventos em Σ, quando refinados, poderiam simplificar a modelagem de uma especificação de controle;

(ii) *Definição das instâncias de refinamentos $\delta \in \Delta^{\sigma}$*: a partir do passo (i), deve-se determinar quais instâncias devem ser associadas a cada um dos eventos selecionados para o refinamento. Tais instâncias modelam diferentes circunstâncias que o evento original pode ocorrer. Dessa maneira, cada instância também carrega uma semântica particular, que deve ser implementada pelo modelo do distinguidor;

(iii) *Complementação do conjunto Δ*: a partir de um dado conjunto de

eventos a serem refinados e seus respectivos conjuntos de instâncias, a construção de um modelo que distingue a ocorrência de tais instâncias pode depender de outros refinamentos. O significado de uma certa instância de um evento pode se concretizar somente quando o sentido de outro evento é conhecido. Então, tal evento deve ser refinado e suas instâncias serem distinguidas. Portanto, é necessário revisitar os passos (i) e (ii), e assim definir corretamente o alfabeto Δ;

(iv) *Modelagem do distinguidor* H_D: neste passo, se constrói um modelo que estabelece as interdependências entre as instâncias de refinamentos em Δ. Esse tarefa consiste na estruturação de um autômato H_D, tal que $L(H_D) = L_D$. O modelo H_D, é normalmente simples no aspecto de modelagem e, em geral, pode ser feito de forma modular.

2.4.2 EXEMPLO DE UM SED COM DISTINGUIDOR

Considere o mesmo exemplo apresentado anteriormente, só que com alfabeto modificado, a fim de que se possa simplificar o modelo da especificação E, i.e., eliminar a dependência do tamanho do *buffer* na especificação E. Suponha que os eventos b e c sejam refinados a fim de que se possa controlar o *buffer* de uma maneira mais eficiente, identificando cada posição. Então, seja $\Delta^b = \{b_\alpha, b_\beta, b_\gamma, b_{ov}\}$ e $\Delta^c = \{c_{un}, c_\alpha, c_\beta, c_\gamma\}$, tal que cada evento identifica uma posição no *buffer*. O evento b_{ov} é criado para representar a possibilidade de *overflow* na planta, enquanto c_{un} aplica a mesma ideia para o *undeflow*.

Agora, as máquinas M_1 e M_2 passam a ser expressas por modelos que incluem esse novo alfabeto, como mostra a Figura 8.

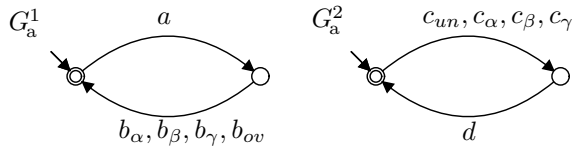

Figura 8 – Modelos para as máquinas M_1 e M_2 com refinamentos

Até aqui, o único efeito foi a aplicação do refinamento sobre G, ou seja, tais modelos correspondem a $\Pi^{-1}(G)$. Nesse caso, a planta é dada por $G_{\mathrm{a}} = G_{\mathrm{a}}^1 \| G_{\mathrm{a}}^2$, mostrada a seguir na Figura 9.

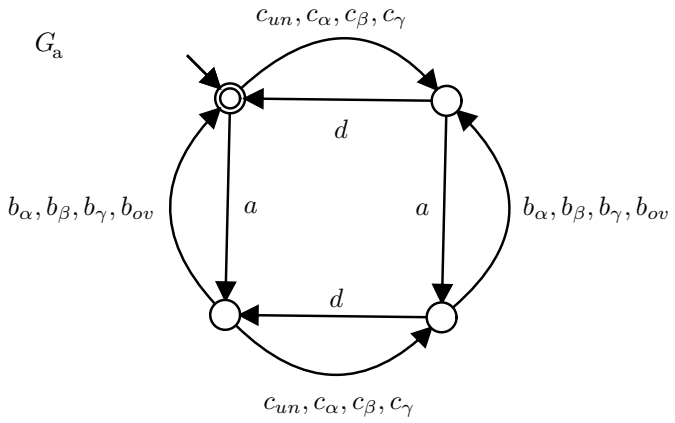

Figura 9 – Modelo da composição $G_{\mathbf{a}} = G_{\mathbf{a}}^{\mathbf{1}} \| G_{\mathbf{a}}^{\mathbf{2}}$

Agora, o modelo da planta fornece os eventos que levam o *buffer* a estar cheio ou vazio. Assim, o controle do *overflow* e do *underflow* nada mais é do que usar essa informação para impedir o problema, sem ter que memorizar inserções no *buffer*. Tais modelos são apresentados na sequência.

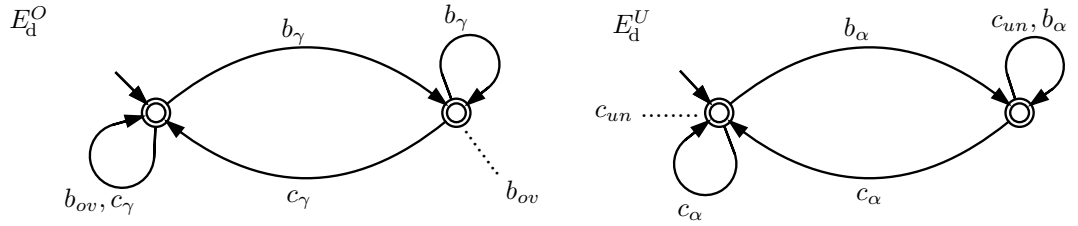

Figura 10 – Modelos para as especificações de *overflow* E_{d}^{O} e *underflow* E_{d}^{U} com refinamentos

Então, os requisitos da especificação E passam a ser modelados por $E_{\mathrm{d}} = E_{\mathrm{d}}^O \| E_{\mathrm{d}}^U$, conforme na Figura 11. O modelo E_{d}^U evita o *underflow* desabilitando em seu estado inicial o evento c_{un}, enquanto E_{d}^O o *overflow* desabilitando o evento b_{ov} no segundo estado. Além disso, o modelo E_{d} independe no tamanho do *buffer*, i.e., possuem um número fixo de estados que não varia, como no caso de E.

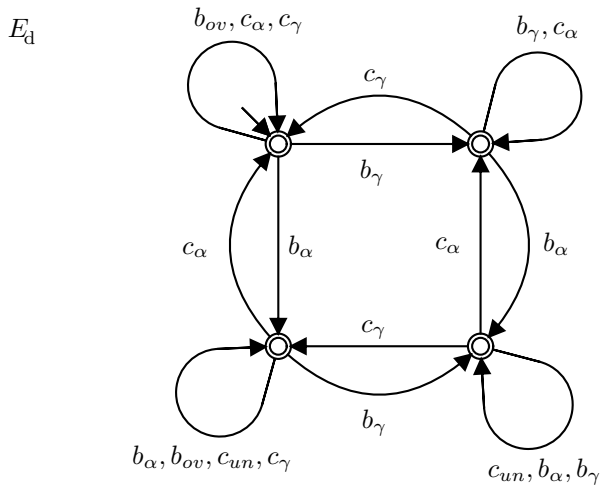

Figura 11 – Modelo da composição $\boldsymbol{E_{\mathrm{d}} = E_{\mathrm{d}}^{O} \| E_{\mathrm{d}}^{U}}$

Embora G_{a} simplifique a especificação, ele por si só não é capaz de distinguir a ocorrência de eventos refinados. O efeito disso é que o sistema de controle poderia não discernir exatamente se um dado evento na planta deveria ou não ser proibido, ainda que a especificação proibisse uma de suas instâncias. O trabalho de distinção da planta fica a cargo do modelo distinguidor, apresentado para o exemplo como na Figura 12.

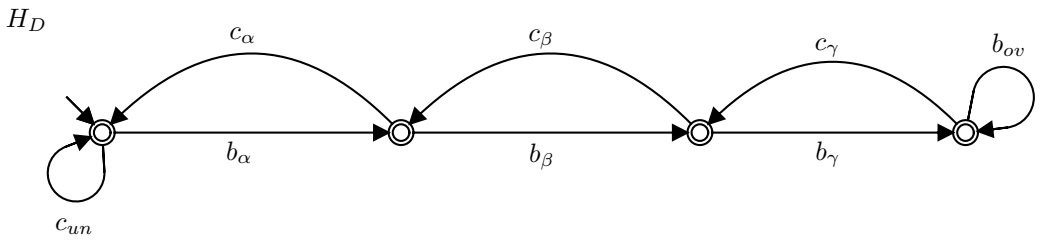

Figura 12 – Modelo do distinguidor $\boldsymbol{H_D}$

O modelo de H_D simplesmente coordena a ordem ou precedência de como as instâncias refinadas do eventos b e c ocorrem, ou seja, em cada estado de H_D ao menos uma instância refinada desses eventos está habilitada. Por exemplo, no estado inicial H_D distingue c_{un} dos demais refinamentos em Δ^c. Outro ponto a se observar é que, embora o número de estados do modelo de H_D ainda dependa de n, esse modelo pode ser modularizado devido aos

refinamentos, como mostrado na Figura 13.

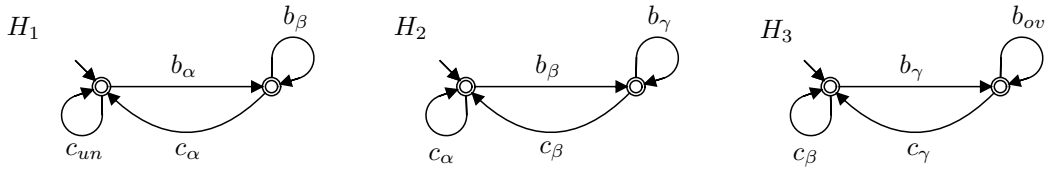

Figura 13 – Modelos dos distinguidores H_1, H_2 e H_3

Expressar o distinguidor de forma modular pode ser vantajoso, no sentido da escalabilidade do sistema, uma vez que se os limites do *buffer* forem alterados e consequentemente as instâncias refinadas dos eventos b e c, a tarefa de modelagem para essa atualização não envolveria refatorar um modelo com $n + 1$ estados, e sim adicionar ou remover um modelo com dois estados, como os apresentados na Figura 13. Além disso, é demonstrado em (CURY *et al.*, 2015; ROSA *et al.*, 2017) que a versão modular do distinguidor é mais vantajosa que sua versão monolítica, visto que tais trabalhos exploram tal característica para beneficiar a síntese e a implementação de controladores ótimos que utilizam refinamentos.

E a composição $H_D = H_1 \| H_2 \| H_3$ leva exatamente ao mesmo modelo monolítico apresentado na Figura 12, como esperado e mostrado na Figura 14.

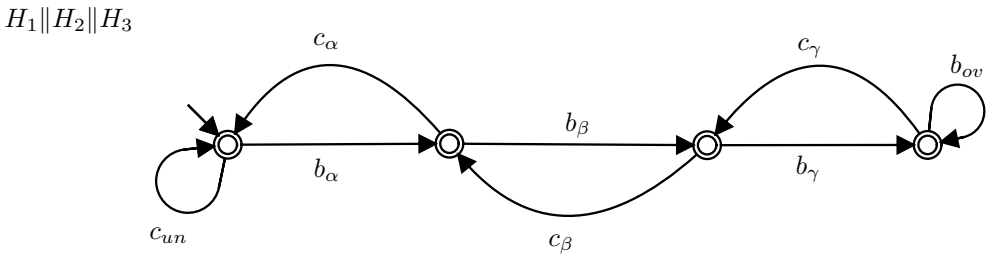

Figura 14 – Modelo da composição $H_D = H_1 \| H_2 \| H_3$

O modelo H_D pode então ser associado à planta G_{a}, o que leva à planta distinguida $G_{\mathrm{d}} = G_{\mathrm{a}} \| H_D = D(G)$. Então, a síntese passa a ser dada sobre um modelo $K_{\mathrm{d}} = G_{\mathrm{d}} \| E_{\mathrm{d}}$, o que resulta em um supervisor S_{d} que representa $\sup\mathcal{C}(K_{\mathrm{d}}, G_{\mathrm{d}})$. O número de estados dos modelos usados na síntese

são apresentados na Tabela 2. Para fins de comparação, os resultados da síntese convencional são reproduzidos.

Tabela 2 – Número de estados dos autômatos usados na síntese do PCS e do PCS-D.

G	E	K	S	
4	$n+1$	$4n+4$	$4n+2$	
G_d	E_d	K_d	S_d	H_D
$4n+4$	4	$4n+4$	$4n+2$	$n+1$

Note que, o modelo de E_{d} independe do tamanho do *buffer*. No entanto, os modelo K_{d} e K possuem a mesma quantidade de estados, então a complexidade para sintetizar os supervisores S e S_{d} é a mesma. Isso ocorre pelo fato de que, na abordagem com distinguidores, os benefícios obtidos na modelagem de E_{d} são compensados pela adição do modelo do distinguidor a planta G_{d}. Em (CURY *et al.*, 2015; ROSA *et al.*, 2017) são apresentadas alternativas de como utilizar distinguidores de modo a beneficiar as etapas de modelagem, síntese e implementação do supervisor.

3 TCS COM MÚLTIPLOS DISTINGUIDORES

Em alguns casos, pode acontecer que mais de um distinguidor precise ser usado sobre o mesmo processo como, por exemplo, para refinar eventos em diferentes etapas. Quando esses eventos são independentes uns dos outros, ou ocorrem em diferentes contextos da planta, a mesma teoria apresentada acima pode ser aplicada, ou simplesmente replicada para n casos.

Porém, quando os eventos a serem refinados dependem de outros eventos também refinados, e cada uma dessas classes de refinamentos requer um distinguidor específico, então o problema recai sobre uma estrutura de refinamentos que este trabalho nomeia de *estruturas concorrentes com dependência de contexto*.

3.1 ILUSTRAÇÃO DO PROBLEMA

Para fundamentar e justificar a proposta na prática, o processo da Figura 5 é alterado no sentido de contemplar, além do controle de *buffer*, também o retrabalho em M_1 das peças saindo de M_2. A nova versão do processo, contemplando essa alteração, é apresentada na Figura 15.

Para ilustrar a dificuldade encontrada em modelar a nova versão do processo, que agora com múltiplos contextos de refinamentos, se considera que as informações dos ciclos de trabalhos e das posições do *buffer* devem ser associados aos eventos originais, mas note que somente os refinamentos do evento c contem abas informações, já b contem somente a informação dos ciclos de trabalhos. E posteriormente é apresentado uma metologia que simplifica a modelagem deste sistema e uma verão parcial de quanto erra dificuldade de modelar neste caso é ilustrado na Figura 16.

E neste processo, é considerado que o número de retrabalhos admitidos é de 2 ciclos, o que leva a um total de 3 ciclos para cada produto a ser manufaturado. O *buffer* continua com capacidade de armazenamento de 3 peças, conforme o caso anterior, o que é identificado por α, β e γ.

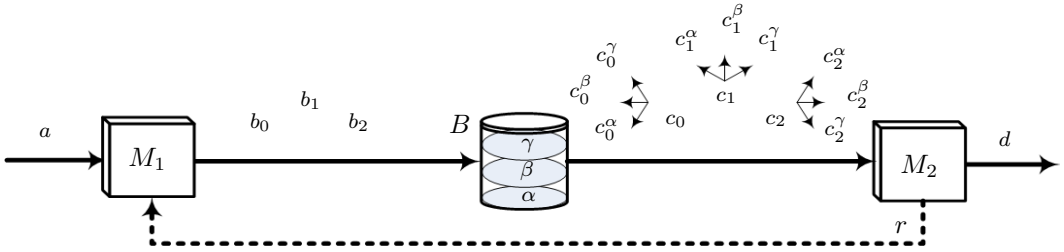

Figura 15 – Processo de manufatura estendido com retrabalho

Para esse novo cenário, o controle do *buffer* passa a depender dos eventos que representam as respectivas posições do *buffer* e, adicionalmente, de cada ciclo de retrabalho. Ou seja, cada evento que representa um contexto de retrabalho precisa ser mapeado em cada evento possível no *buffer*. Na figura 15 esse contexto de controle é ilustrado em relação a retirada de peças do *buffer* (evento c em Σ). Se antes essa ação era modelada pelos refinamentos c_{un}, c_α, c_β e c_γ em Δ, agora, cada um deles passa a se associado também ao número do respectivo ciclo de trabalho. Como o processo admite até 3 ciclos, então as instâncias correspondentes à retirada de peças do *buffer* são replicadas exponencialmente em função da quantidade de ciclos.

Um modelo capaz de distinguir o contexto de cada evento do sistema é complexo de ser construído manualmente. Uma versão preliminar dessa estrutura é apresentada na Figura 16.

No estado inicial, esse modelo permite o início de 3 contextos diferentes, com b_0, com b_1 e com b_2, representando respectivamente uma peça em cada ciclo de retrabalho. As sequências com b_1 e b_2 foram ocultadas, por simplicidade, e o complemento dessa estrutura é indicado por "...". Já o caminho com b_0 é detalhado, para ilustração.

A partir da ocorrência de um evento b_0, sabe-se que uma peça adentrou no sistema pela primeira vez (0 retrabalhos). No entanto, nada sabe-se sobre qual posição será ocupada por essa peça no *buffer* e, portanto, tem-se que mapear todas as possíveis combinações. Como o *buffer* está inicialmente vazio, a retirada dessa peça do *buffer* será apenas através do evento c_0^α (ciclo 0 e posição α).

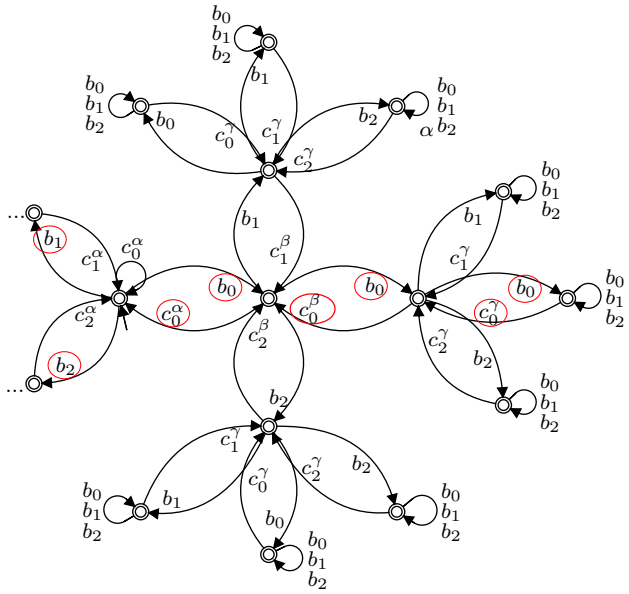

Figura 16 – Distinguidor de eventos em Δ^c no primeiro ciclo

Caso a peça for retirada, então o sistema evolui de volta ao estado inicial. Do contrário, 3 novos contextos são gerados após b_0, novamente com b_0, com b_1 e com b_2. Cada um deles representa o contexto da próxima peça, ou seja, de qual ciclo de retrabalho ela é proveniente. Para cada possibilidade, o modelo escolhe uma única opção de retirada do *buffer*, o que na verdade materializa a ideia de distinção.

O complemento da estrutura do autômato, indicado por "...", dá uma dimensão do quão complexo pode ser essa tarefa. Ressalta-se que a quantidade de ciclos poderia ser maior e/ou o *buffer* poderia contemplar mais capacidade de armazenamento, o que influencia diretamente na complexidade de modelagem desse tipo de problema.

Na literatura, não existe até então abordagens que tratam desses casos. Em Fischer e Leal (2014), os autores exploram esse fato e explicitam os efeitos desses casos sobre a resolução de um problema de controle. Diferentemente, este trabalho tem por objetivo o tratamento conjunto das diferentes instâncias de refinamentos, criando as condições teóricas para sustentar a obtenção do supervisor a partir de múltiplas e dependentes

estruturas de refinamentos, envolta do mesmo problema de controle. Para tal, o procedimento é esquematizado conforme a seguir.

3.2 FORMALIZAÇÃO DE MÚLTIPLOS REFINAMENTOS

Como exposto na seção anterior, existem problemas de controle que necessitam de diferentes níveis de refinamentos. Desse modo, um evento refinado passa a conter mais de um contexto (nível de informação) associado a ele, com intuito de esquematizar/sistematizar tal ideia, define-se uma metodologia onde cada contexto é associado a uma camada de refinamentos, de modo que o resultado final é idêntico ao caso convencional de refinamentos. Assim, o projetista poderia abordar o problema de controle com refinamentos de forma incremental ao invés de tratar o problema convencionalmente. Formalmente, essa ideia é descrita como segue.

Seja um alfabeto Σ e sua versão refinada Δ tal que

$$\forall \sigma \in \Sigma, \exists \Delta^{\sigma} \subseteq \Delta.$$

Seja Δ_1 a versão refinada de Δ tal que

$$\forall \delta \in \Delta, \exists \Delta_1^{\delta} \subseteq \Delta_1.$$

Generalizando, seja Δ_k a versão refinada de Δ_{k-1} tal que

$$\forall \delta_{k-1} \in \Delta_{k-1}, \exists \Delta_k^{\delta_{k-1}} \subseteq \Delta_k.$$

Note que a cada vez que um alfabeto é refinado, um novo nível de informação/contexto pode ser adicionado aos eventos do alfabeto “original”. Desse modo cada evento em Δ possui um contexto associado a ele, enquanto que os eventos em Δ_1 e Δ_k contêm dois e k contextos, respectivamente.

A relação entre os alfabetos $\Sigma, \Delta, \Delta_1, \cdots, \Delta_k$ é defina sequencialmente por meio dos mapas Π^{-1} e Π como

$$\Pi^{-1} : \Sigma^* \to 2^{\Delta^*};$$
$$\Pi_1^{-1} : \Delta^* \to 2^{\Delta_1^*};$$
$$\vdots$$
$$\Pi_k^{-1} : \Delta_{k-1}^* \to 2^{\Delta_k^*},$$

e

$$\Pi : \Delta^* \to \Sigma^*;$$
$$\Pi_1 : \Delta_1^* \to \Delta^*;$$
$$\vdots$$
$$\Pi_k : \Delta_k^* \to \Delta_{k-1}^*.$$

A Figura 17 ilustra como os mapas Π_i^{-1} e Π_i são utilizados para prover os diferentes níveis de informações contidas nos alfabetos refinados.

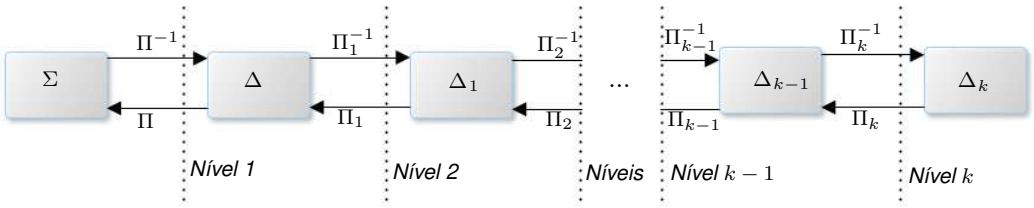

Figura 17 – Ilustração da sequência de como os mapas Π^{-1} e Π provem os diferentes níveis de informações dos refinamentos

Observe que a medida que o índice $i = 1, \cdots, k$ do mapa Π_i^{-1} é incrementado, o nível de informação embarcado em cada refinamento é maior. Diferentemente, quando i é decrementado no mapa Π_i o nível de informação é menor.

3.3 FORMALIZAÇÃO DE MÚLTIPLOS DISTINGUIDORES

Tendo a relação entre os diferentes níveis de informação definida, o próximo passo é associar um distinguidor para cada nível, uma vez que o mapa Π^{-1} simplesmente aumenta o nível de informação mas não adiciona o contexto ao evento refinado.

Assim, para o primeiro nível de refinamento, o efeito de um

distinguidor $D : \Sigma^* \rightarrow 2^{\Delta^*}$ sobre uma linguagem $L \subseteq \Sigma^*$, é como descrito no capítulo 2.4, ou seja

$$D(L) = \Pi^{-1}(L) \cap L_D.$$

Para o segundo nível de refinamento, o efeito do distinguidor $D_1 : \Delta^* \rightarrow 2^{\Delta_1^*}$ sobre uma linguagem em Δ^* pode ser estendido tal que

$$D_1(D(L)) = \Pi_1^{-1}(D(L)) \cap L_{D_1}.$$

Com isso podemos generalizar o efeito de um distinguidor $D_k : \Delta_{k-1}^* \rightarrow 2^{\Delta_k^*}$ sobre uma linguagem em Δ_{k-1}^* como

$$D_k(D_{k-1}(...(D_1(D(L)))...)) = \Pi_k^{-1}(D_{k-1}(...(D_1(D(L)))...)) \cap L_{D_k}.$$

Então o efeito de um distinguidor D_k sobre uma planta G é tal que

$$D_k(L(G)) = \Pi_k^{-1}(...(\Pi_1^{-1}(\Pi^{-1}(L(G)) \cap L_D) \cap L_{D_1})...) \cap L_{D_k};$$
$$D_k(L^\omega(G)) = \Pi_k^{-1}(...(\Pi_1^{-1}(\Pi^{-1}(L^\omega(G)) \cap L_D) \cap L_{D_1})...) \cap L_{D_k}.$$

Desse modo, a planta distinguida G_d para G construída a partir da utilização de k distinguidores é dada por $L(G_\mathrm{d}) = D_k(L(G))$ e $L^\omega(G_\mathrm{d}) = D_k(L^\omega(G))$, e o processo de síntese do supervisor para esse caso segue como apresentado na seção 2.4.

3.4 RESOLUÇÃO DO PROBLEMA PROPOSTO

No exemplo apresentado na subseção 2.4.2, a sua resolução consistiu em simplificar o modelo da especificação E, com o refinamento dos eventos b e c, a fim de controlar o *buffer* de maneira mais eficiente. Para isso, o evento b foi refinado em $\Delta^b = \{b_\alpha, b_\beta, b_\gamma, b_{ov}\}$, enquanto o evento c em $\Delta^c = \{c_{un}, c_\alpha, c_\beta, c_\gamma\}$, tal que c_{un} e b_{ov} identificam o *underflow* e *overflow*, respectivamente, enquanto os demais refinamentos identificam as posições

intermediárias, como apresentado na Figura 18.

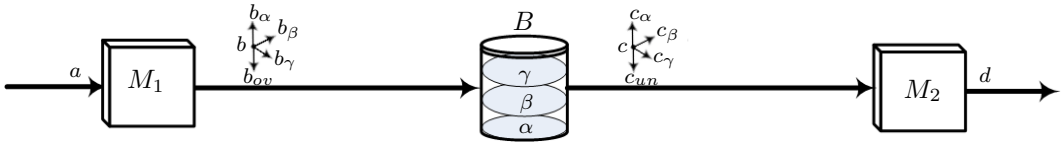

Figura 18 – Processo de manufatura com os refinamentos

Sendo assim, a resolução obtida no exemplo da subseção 2.4.2, resultou nas plantas com refinamentos da Figura 8, com sua composição mostrada na Figura 9, as especificações E_{d}^{O} e E_{d}^{U} na Figura 10, para controle do *overflow* e do *underflow* do *buffer*, e finalmente o modelo do distinguidor H_D apresentado na Figura 12, com sua versão modular como na Figura 13.

Note que podemos utilizar os resultados obtidos no exemplo da subseção 2.4.2 para resolver o problema proposto na seção 3, visto que tal problema é uma extensão do exemplo da subseção 2.4.2. Assim, para resolver o problema da seção 3 é necessário substituir a máquina M_2 por uma máquina T_U que a partir de um sensor realiza teste de qualidade sobre as peças manufaturas por M_1, então, a máquina M_2 agora passou a ser uma unidades de teste para as peças, tal teste resulta no retrabalho (evento r, com $r \in \Sigma_c$) ou na liberação da peça (evento d) como apresentado na Figura 20.

E os novos autômatos G^1 e G^2 para o sistema apresentado na Figura 20 são apresentados na Figura 19, modelando M_1 e T_U respectivamente.

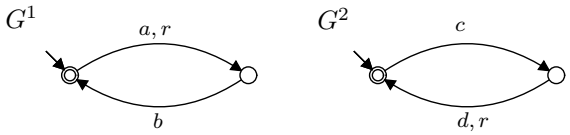

Figura 19 – Modelo das plantas M_1 e T_U para o problema proposto

Com isso podemos resolver o problema proposto neste trabalho, partindo do que foi apresentado na subseção 2.4.2. Suponha que o evento r, o conjunto de eventos $\Delta^b = \{b_\alpha, b_\beta, b_\gamma, b_{ov}\}$ e $\Delta^c = \{c_{un}, c_\alpha, c_\beta, c_\gamma\}$ sejam refinados agora de tal modo que possam simplificar tanto o problema do *buffer*, quanto do

retrabalho, i.e., tais eventos passariam a incorporar informações sobre o ciclo de trabalho. Então, para o evento r com a possibilidade de 2 retrabalhos, seus refinamentos são $\Delta_1^r = \{r_1, r_2\}$, para o conjunto de evento Δ^b, cada um de seus eventos é refinado em novo conjunto de refinamentos como $\Delta_1^{b_\alpha} = \{b_0^\alpha, b_1^\alpha, b_2^\alpha\}$, $\Delta_1^{b_\beta} = \{b_0^\beta, b_1^\beta, b_2^\beta\}$, $\Delta_1^{b_\gamma} = \{b_0^\gamma, b_1^\gamma, b_2^\gamma\}$ e $\Delta_1^{b_{ov}} = \{b_0^{ov}, b_1^{ov}, b_2^{ov}\}$. O mesmo acontece para os eventos em Δ^c, o resultado disso é $\Delta_1^{c_\alpha} = \{c_0^\alpha, c_1^\alpha, c_2^\alpha\}$, $\Delta_1^{c_\beta} = \{c_0^\beta, c_1^\beta, c_2^\beta\}$, $\Delta_1^{c_\gamma} = \{c_0^\gamma, c_1^\gamma, c_2^\gamma\}$ e $\Delta_1^{c_{un}} = \{c_0^{un}, c_1^{un}, c_2^{un}\}$. O efeito de $\Pi_1^{-1}(\Pi^{-1}(G))$ é ilustrado na Figura 20.

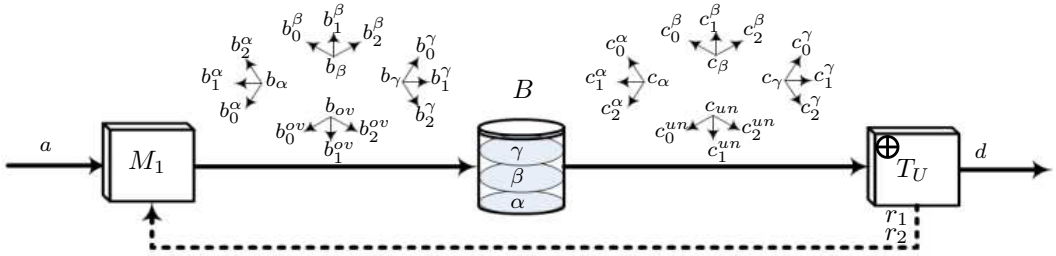

Figura 20 – Processo de manufatura estendido com retrabalho e os refinamentos

Em termos de autômatos, as máquinas M_1 e T_U são modeladas em $\Delta_1 = \Pi_1^{-1}(\Pi^{-1}(\Sigma))$, respectivamente, por $G_{\mathrm{a1}}^1 = \Pi_1^{-1}(\Pi^{-1}(G^1))$ e $G_{\mathrm{a1}}^2 = \Pi_1^{-1}(\Pi^{-1}(G^2))$ da Figura 21.

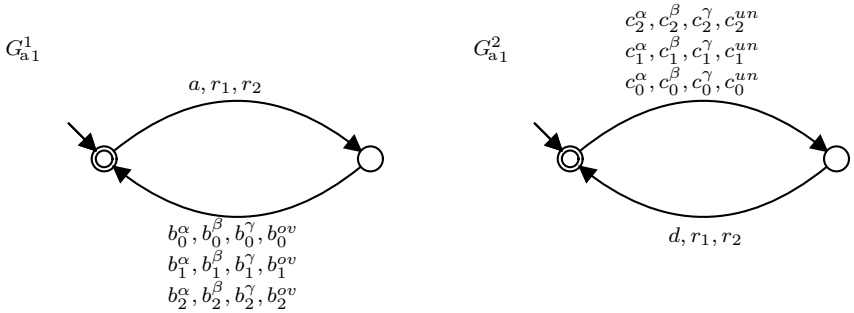

Figura 21 – Modelos para as máquinas M_1 e T_U em Δ_1

Note que os modelos das plantas agora fornecem os eventos que levam o *buffer* a estar cheio ou vazio, para todas as sequências de retrabalho a serem realizadas nas peças.

Agora as especificações E_{d}^U e E_{d}^O da Figura 10 são modeladas em

Δ_1 por $E^U_{\mathrm{d}1} = \Pi_1^{-1}(E^U_{\mathrm{d}})$ e $E^O_{\mathrm{d}1} = \Pi_1^{-1}(E^O_{\mathrm{d}})$ da Figura 22.

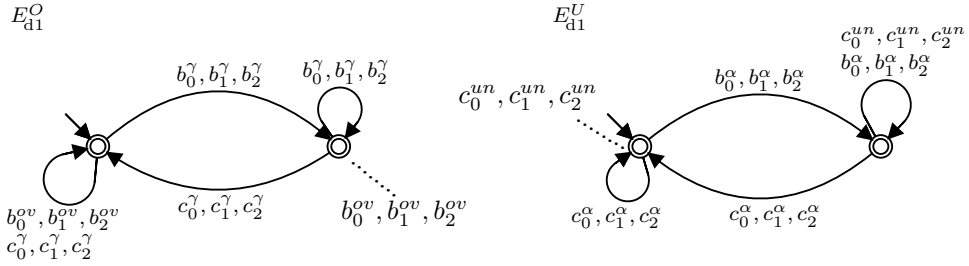

Figura 22 – Modelos E^O_{d} e E^U_{d} em Δ_1

O modelo $E^U_{\mathrm{d}1}$ evita o *underflow* de todas as possibilidades no *buffer* que foi resultado dos retrabalhos realizados nas peças, desabilitando em seu estado inicial os eventos $c^{un}_0, c^{un}_1, c^{un}_2$, enquanto $E^O_{\mathrm{d}1}$ o *overflow* desabilitando os eventos $b^{ov}_0, b^{ov}_1, b^{ov}_2$ no segundo estado. Além disso, os modelos continuam independente do tamanho do *buffer*, i.e., possuem um número fixo de estados que não varia, como no caso de E.

O modelo de H_D antes contemplava os eventos que não dependiam do retrabalho, com isso o novo distinguidor para este contexto agora passa a ser modelado por $H_{D_1} = \Pi_1^{-1}(H_D)$ da Figura 23. Observe que, H_{D_1} ainda continua distinguindo os refinamentos dos eventos b e c em relação as posições de B. Por exemplo, no estado inicial H_{D_1} distingue $\Delta_1^{b_\alpha}$ dos demais refinamentos Δ_1^δ para $\delta \in \{b_\beta, b_\gamma, b_{ov}\}$.

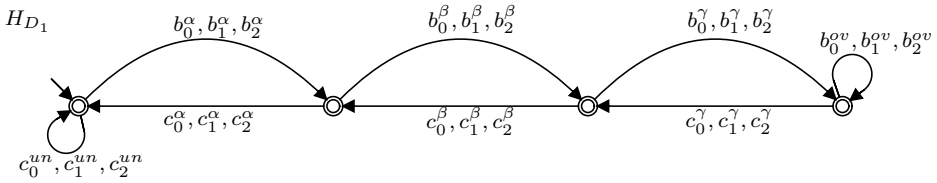

Figura 23 – Modelo de H_{D_1} em Δ_1

Sendo ainda possível modelar o distinguidor H_{D_1} através de 3 distinguidores H_1, H_2 e H_3 apresentados na Figura 24 com 2 estados cada distinguidor. E a composição $H_{D_1} = H_1 \| H_2 \| H_3$ leva exatamente ao mesmo modelo apresentado na Figura 23.

Considerando que o requisito de controle para o exemplo é limitar

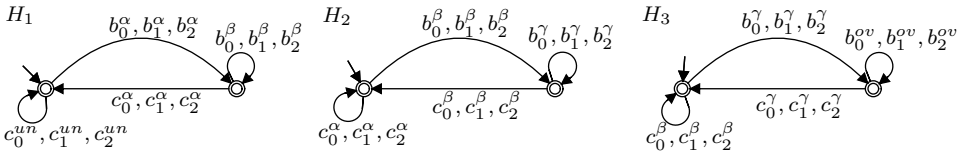

Figura 24 – Modelos dos distinguidores H_1, H_2 e H_3, modo modular de H_{D_1}

a um número de retrabalho para cada peça, em Teixeira (2013, p. 56), este requisito é demostrado de tal forma que, sua resolução do modo monolítico nos eventos originais é obtida com 32 estados para 1 retrabalho realizado e para 2 retrabalhos com 512 estados, só que em um sistema com mais componentes que esta resolução. Note que, este sistema proposto pode conter mais componentes, como apresentado em Teixeira (2013, p. 56) e para cada componente inserido e cada retrabalho no sistema, o número de estados aumenta consideravelmente. O principio de resolução para esse requisito é o mesmo adotado em Teixeira (2013, p. 77), só que em uma escala reduzida. Com isso, utilizando as informações contidas em Δ_1 para tal requisito de controle, pode ser expresso pelo autômato E^s_{d1} da Figura 25.

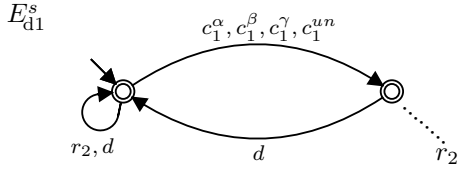

Figura 25 – Modelo da especificação E^s_{d1}

Observe que o modelo E^s_{d1} evita o início do segundo ciclo de retrabalho desabilitando o evento r_2 após o término do primeiro ciclo de trabalho para qualquer peça independente da posição em B (eventos $c_1^\alpha, c_1^\beta, c_1^\gamma$ e c_1^{un}).

Resta agora definir o modelo do distinguidor que vai distinguir os eventos em Δ_1 em relação ao ciclo de trabalho, uma vez que H_{D1} os distingue em função das posições de B. Para isso, são definidos os modelos $H^{b_1}_{\mathrm{d1}}$, $H^{b_2}_{\mathrm{d1}}$, $H^{c_1}_{\mathrm{d1}}$, $H^{c_2}_{\mathrm{d1}}$ e H^r_{d1} das Figuras 26, 27 e 28, as quais especificam as sequências de operações, tal que o modelo do distinguidor preditível em relação ao ciclo de trabalho é dada pela composição $H^{ret}_{\mathrm{d1}} = H^{b_1}_{\mathrm{d1}} \| H^{b_2}_{\mathrm{d1}} \| H^{c_1}_{\mathrm{d1}} \| H^{c_2}_{\mathrm{d1}} \| H^r_{\mathrm{d1}}$.

Os modelos $H^{b_1}_{\mathrm{d1}}$ e $H^{b_2}_{\mathrm{d1}}$ da Figura 26, distinguem os refinamentos

de b em Δ_1 em função do ciclo de trabalho independente da posição de B. Por exemplo, $H_{\mathrm{d1}}^{b_1}$ distingue os eventos b_1^x, com $x \in \{\alpha, \beta, \gamma, ov\}$, associado ao segundo ciclo de trabalho de b_0^x e b_2^x no segundo estado, ao passo que $H_{\mathrm{d1}}^{b_2}$ os eventos b_2^x associados ao terceiro ciclo de trabalho no mesmo estado. Já a distinção de b em relação ao primeiro ciclo de trabalho (b_0^x) é dada pela composição $H_{\mathrm{d1}}^{b_1} \| H_{\mathrm{d1}}^{b_2}$ no estado inicial. Os modelos $H_{\mathrm{d1}}^{b_1}$ e $H_{\mathrm{d1}}^{b_2}$ são apresentados na Figura 26.

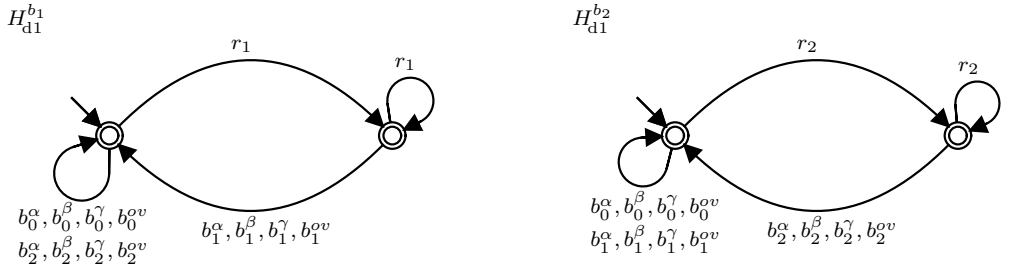

Figura 26 – Modelos dos distinguidores $H_{\mathrm{d1}}^{b_1}$ e $H_{\mathrm{d1}}^{b_2}$

Os modelos $H_{\mathrm{d1}}^{c_1}$ e $H_{\mathrm{d1}}^{c_2}$ distinguem os refinamentos de c em Δ_1 de maneira análoga a $H_{\mathrm{d1}}^{b_1}$ e $H_{\mathrm{d1}}^{b_2}$. E os modelos dos autômatos de $H_{\mathrm{d1}}^{c_1}$ e $H_{\mathrm{d1}}^{c_2}$ são apresentados na Figura 27.

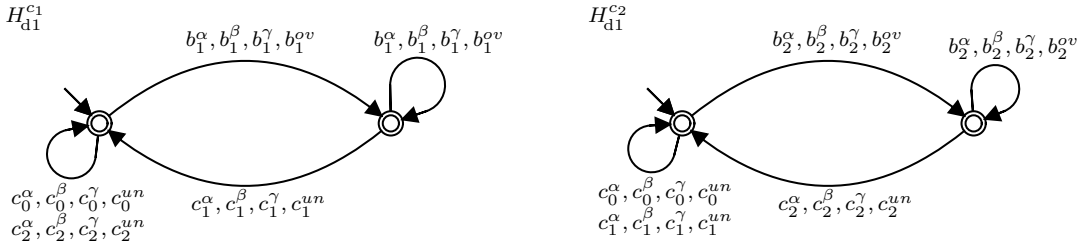

Figura 27 – Modelos dos distinguidores $H_{\mathrm{d1}}^{c_1}$ e $H_{\mathrm{d1}}^{c_2}$

O modelo H_{d1}^{r} distingue os refinamentos de r como devem ser coordenados os trabalhos realizados nas peças.

Agora, o processo de síntese do supervisor pode ser conduzido utilizando os seguintes modelos $G_{\mathrm{d}} = G_{\mathrm{a1}}^1 \| G_{\mathrm{a1}}^2 \| H_{D_1} \| H_{\mathrm{d1}}^{ret}$ e $E_{\mathrm{d}} = E_{\mathrm{d1}}^U \| E_{\mathrm{d1}}^O \| E_{\mathrm{d1}}^s$. Então, a síntese passa a ser dada sobre um modelo $K_{\mathrm{d}} = G_{\mathrm{d}} \| E_{\mathrm{d}}$, o que resulta em um supervisor S_{d} que representa $\sup\mathcal{C}(K_{\mathrm{d}}, G_{\mathrm{d}})$. O número de estados dos

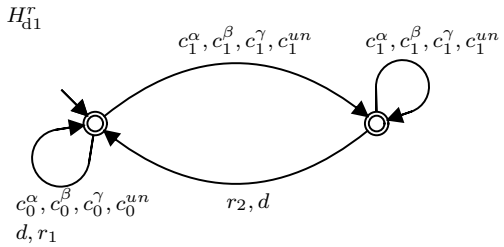

Figura 28 – Modelo do distinguidor H^r_{d1}

modelos usados na síntese são apresentados na Tabela 3.

Tabela 3 – Número de estados dos autômatos para resolução do problema proposto

G_a	H_D	G_d	E_d	K_d	S_d
4	63	96	8	55	45
G			$E = \Pi(E_{\mathrm{d}} \| H_D)$	K	S
4			24	55	45

A primeira linha da tabela apresenta os modelos utilizados na síntese com abordagem proposta neste trabalho e para efeitos comparativos, a segunda linha da tabela mostra os modelos para síntese para o caso sem refinamentos. Note que, neste caso a especificação passaria de 8 para 24 estados e estaria atrelada a quantidade de retrabalhos e ao tamanho do *buffer*, enquanto que E_{d} permanece fixo.

Porém, o custo para se síntese o supervisor com ou sem refinamentos é o mesmo, visto que K_{d} e K são equivalentes, i.e., $\Pi(K_{\mathrm{d}}) = K$, consequentemente os supervisores resultantes também são equivalentes. No entanto, a abordagem com refinamentos possui as vantagens adicionais de permitir simplificar a síntese utilizando aproximações (CURY *et al.*, 2015) e, alem disso, explorar a implementação modular (ROSA *et al.*, 2017) que reduz o custo em termo de memória.

4 CONCLUSÕES

Este trabalho explora uma dificuldade encontrada na síntese de controladores para SEDs, que se refere à complexidade de modelagem. Em alguns casos, representar um problema utilizando apenas os eventos originais, providos naturalmente pela planta, pode ser extremamente custoso devido à grande quantidade de combinações de eventos, estados e transições a serem consideradas no modelo. Isso ocorre basicamente quando o engenheiro precisa memorizar longas sequências de eventos até que ele identifique um ponto de evolução do sistema a ser efetivamente controlado.

Como solução, foi proposta uma abordagem que projeta múltiplas camadas de refinamentos que enriquecem os eventos originais do sistema. Em tese, prover mais detalhes sobre eventos, tende a facilitar o trabalho de expressar as especificações.

A solução proposta foi ilustrada por meio de um exemplo, um sistema de manufatura com retrabalho e um *buffer* capaz de armazenar peças, os resultados demonstram que foi possível viabilizar as tarefas de modelagens, para a qual o esforço de modelar eram inviável de serem tratadas manualmente. Em contraste, dentro da presente proposta, o problema foi resolvido com um conjunto de autômatos de apenas 2 estado cada.

Como possíveis limitações da abordagem, ressalta-se o fato de que ela não reduz o esforço de síntese e também, a construção tanto dos refinamentos quanto dos módulos distinguidores de refinamentos na proposta, é uma atividade manual.

Como perspectivas futuras, estima-se que os seguintes pontos podem ser explorados:

- Integrar a abordagem à ideia de abstrações, para que resulte também em benefícios computacionais, em adição aos benefícios de modelagem;
- Implementação de ferramentas para a geração automática de modelos e

refinamentos;

- Contextualização dos passos para a construção de múltiplos refinamentos;
- Aplicar a abordagem a mais contextos de diferentes sistemas industriais e exemplos apresentados na literatura.

REFERÊNCIAS

BOUZON, G.; QUEIROZ, M. H. de; CURY, J. E. R. Supervisory control of des with distinguishing sensors. In: **International Workshop on Discrete Event Systems, WODES'08**. Gothenburg, Sweden: [s.n.], 2008. p. 22–27.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to Discrete Event Systems**. 2. ed. NY: Springer Science, 2008.

CURY, J E R. Teoria de controle supervisório de sistemas a eventos discretos. **V Simpósio Brasileiro de Automação Inteligente**, 2001.

CURY, J. E. R.; QUEIROZ, M. H. de; BOUZON, G.; TEIXEIRA, M. Supervisory control of discrete event systems with distinguishers. **Automatica**, v. 56, p. 93 – 104, 2015.

FISCHER, G.; LEAL, A. Investigação do uso de distinguidores na síntese de supervisores em função de mudanças na planta. In: **Congresso Brasileiro de Automática**. [S.l.: s.n.], 2014.

GIUA, A.; DICESARE, F. Blocking and controllability of petri nets in supervisory control. **Automatic Control, IEEE Transactions on**, v. 39, n. 4, p. 818–823, 1994.

HAYKIN, SIMON; VEEN, Barry Van. **Sinais e sistemas**. [S.l.]: Bookman, 2001.

HOPCROFT, J. E.; ULLMAN, J. D.; MOTWANI, R. **Introduction to Automata Theory, Languages and Computation**. 2. ed. [S.l.]: Addison Wesley, 2001.

JAIN, R. **Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurements, Simulation And Modeling**. 1. ed. [S.l.]: John Wiley & Sons, Inc., 1991.

KUMAR, R.; GARG, V. K. On computation of state avoidance control for infinite state systems in assignment program framework. **IEEE Transactions Automation Science and Engineering**, v. 2, n. 1, p. 87–91, Jan. 2005.

OGATA, Katsuhiko. **Engenharia de controle moderno**. 5. ed. São Paulo: Pearson Prentice Hall, 2010.

PNUELI, Amir. The temporal logic of programs. In: **18th Annual Symposium on Foundations of Computer Science**. [S.l.: s.n.], 1977. p. 46–57.

RAMADGE, P J; WONHAM, W M. Supervisory control of a class of discrete event process. **SIAM Journal of Control and Optimization**, v. 25, n. 1, p. 206–230, 1987.

RAMADGE, P J; WONHAM, W M. The control of discrete event systems. **Discrete Event Dynamic Systems**, v. 77, p. 81–98, 1989.

ROSA, Marcelo; TEIXEIRA, Marcelo; DENARDIN, Gustavo W.; TORRICO, Cesar R. C.; CURY, José E. R. Efficient implementation of distinguished controllers for discrete-event systems. **IFAC-PapersOnLine**, Elsevier, v. 50, n. 1, p. 1187–1192, 2017.

SCHUTTER, B. De; BOOM, T. van den. Max-plus algebra and max-plus linear discrete event systems: An introduction. In: **Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on**. [S.l.: s.n.], 2008. p. 36–42.

SPACEK, P.; MOUDNI, A. El; ZERHOUNI, S.; FERNEY, M. Control of nonautonomous discrete event systems using dioid algebra. In: **Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on**. [S.l.: s.n.], 1996. v. 1, p. 609–615.

TEIXEIRA, M. **Explorando o uso de distinguidores e de autômatos finitos estendidos na teoria do controle supervisório de sistemas a eventos discretos**. 137 p. Tese (Doutorado) — Universidade Federal de Santa, 2013.

TEIXEIRA, M.; MALIK, R.; CURY, J.E.R.; QUEIROZ, M.H. de. Supervisory control of des with extended finite-state machines and variable abstraction. **Automatic Control, IEEE Transactions on**, v. 60, n. 1, p. 118–129, 2014.

WONHAM, W.M. **Notes on Discrete Event Systems**. 2002. University of Toronto.

WONHAM, W M; RAMADGE, P J. On the supremal controllable sublanguage of a given language. **SIAM Journal of Control and Optimization**, v. 25, n. 3, p. 637–659, 1987.