



UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA
PROGRAMA DE PÓS - GRADUAÇÃO EM ENGENHARIA DE
TELEINFORMÁTICA

JONATHA RODRIGUES DA COSTA

ALGORITMO DE CONVERSÃO DE REDES DE PETRI COLORIDAS PARA
LADDER LOGIC DIAGRAM (LLD)

FORTALEZA - CEARÁ

2014

JONATHA RODRIGUES DA COSTA

**ALGORITMO DE CONVERSÃO DE REDES DE PETRI COLORIDAS
PARA LADDER LOGIC DIAGRAM (LLD)**

Dissertação de Mestrado apresentada ao PPGETI
- Programa de Pós-Graduação em Engenharia de
Teleinformática, da Universidade Federal do
Ceará, como parte dos requisitos para a obtenção
do grau de Mestre em Engenharia de
Teleinformática.

Orientador: Prof. Dr. Giovanni Cordeiro Barroso

FORTALEZA - CEARÁ

2014

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Pós-Graduação em Engenharia - BPGE

-
- C873a Costa, Jonatha Rodrigues da.
Algoritmo de conversão de Redes de Petri coloridas para Ladder Logic Diagram (LLD) / Jonatha Rodrigues da Costa. – 2014
101 f. : il. , enc. ; 30 cm.
- Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2014.
Área de concentração: Sinais e sistemas.
Orientação: Prof. Dr. Giovanni Cordeiro Barroso.
1. Teleinformática. 2. Sistemas de controle supervisório. 3. Controladores programáveis. I.
Título.


JONATHA RODRIGUES DA COSTA

***ALGORITMO DE CONVERSÃO DE REDES DE PETRI COLORIDAS PARA
LADDER LOGIC DIAGRAM (LLD)***


Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia de Teleinformática e aprovada em sua forma final pelo PPGETI - Programa de Pós-Graduação em Engenharia de Teleinformática da Universidade Federal do Ceará.

Aprovada: 28/01/2014.

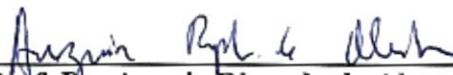
BANCA EXAMINADORA



Prof. Dr. Giovanni Cordeiro Barroso (Orientador)
Universidade Federal do Ceará



Prof. Dr. George André Pereira Thé
Universidade Federal do Ceará



Prof. Dr. Auzuir Ripardo de Alexandria
Instituto Federal de Educação, Ciência e Tecnologia do Ceará

A minha esposa Valdenice Costa e
minha filha Isabelle Camile Costa.

AGRADECIMENTOS

Ao meu orientador Prof. Dr. Giovanni Cordeiro Barroso, pela orientação efetiva, compreensão, incentivo, serenidade e valiosa colaboração.

A Universidade Federal do Ceará, que através do Departamento de Engenharia de Teleinformática, recebeu-me para a realização do Mestrado.

Aos meus colegas de trabalho e de estudos pela sinergia, discussões técnicas e enriquecimento profissional, ao professor Carlos Hairon Ribeiro Gonçalves e ao aluno de engenharia de computação Luan James Tobias Lopes do Instituto Federal de Educação Ciência e Tecnologia do Ceará – IFCE, que me ajudaram na construção da plataforma computacional em JAVA tornando possível a conversão automatizada de uma linguagem para outra, conforme estruturado neste trabalho.

A minha família pelo amor, cuidado e atenção.

Aos meus pais pelo exemplo de vida, superação e conquista.

A Jesus Cristo meu Senhor, Salvador e Santificador.

RESUMO

Neste trabalho é apresentada uma proposta de conversão de modelos de células de manufatura (FMS) em Redes de Petri Coloridas (RPC) para *Ladder Logic Diagram (LLD)*. Explora-se modelos de RPC controlados - construídos usando-se como metodologia de controle supervísório as Restrições de Controle sobre Cores Decompostas (RCCD) - e modelos não controlados. Enfatiza-se a transformação de RPC não controlada para LLD e a facilidade de inclusão do controle. Os resultados aqui descritos demonstram que a metodologia facilita o trabalho de um programador de CLP em LLD minimizando os possíveis erros durante a fase de programação. Para exemplificar a metodologia foi desenvolvido um conversor de RPC para LLD, produzido em ambiente JAVA, afim de evidenciar a dinâmica de conversão. Para a validação dos resultados do conversor é usado um robô articulado vertical controlado via CLP, que opera comparativamente com e sem supervisão, o qual fora modelado primeiramente em RPC sem supervisão, depois aplicada a técnica RCCD, a supervisão e a conversão para LLD.

Palavras-chave - Conversão RPC-LLD, Controle Supervísório, Redes de Petri Coloridas, Sistemas a Eventos Discretos.

ABSTRACT

In this dissertation, is presented a proposal of conversion of models of manufacturing cell (FMS) from Colored Petri Nets (CPN) to Ladder Logic Diagram (LLD). It explores the controlled CPN models - constructed using the methodology of supervisory Control Restrictions on Decomposed Colors (CRDC) - and not controlled models. It emphasizes the transformation of CPN not controlled for the LLD and ease for inclusion of control. The results described herein demonstrate that the methodology facilitates the work of a PLC programmer in LLD minimizing possible errors during the programming phase. To illustrate the methodology we developed a converter from CPN to LLD, produced in JAVA, in order to clarify the dynamic conversion environment. To validate the results of the converter, is controlled by PLC, a vertical articulated robot which works compared with and without supervision which had been firstly modeled in CPN without supervision, then applied to CRDC, the supervision and the conversion technique for LLD.

Keywords: Conversion CPN-LLD, Supervisory Control, Colored Petri Nets, Discrets Event Systems.

LISTA DE ILUSTRAÇÕES

Figura 1.1 - Exemplo de LLD.....	20
Figura 1.2 - Modelo de LLD convertido para RPC	20
Figura 2.1- Representação gráfica dos elementos de RP.....	23
Figura 2.2 - Lugar com token	24
Figura 2.3 –Exemplo de Rede de Petri (RP)	26
Figura 2.4 – RPs e suas extensões	27
Figura 2.5 - Rede de Petri Colorida.....	28
Figura 2.6 - Diagrama de blocos do CLP	36
Figura 2.7 - Trecho de programa nas linguagens da IEC 61131-3.....	40
Figura 3.1 - Conversão de lugar numa memória de CLP	44
Figura 3.2 - Conversão de lugar marcado numa memória de CLP	45
Figura 3.3 - Conversão de transição com um único arco de entrada.....	45
Figura 3.4 - Conversão de transição com mais de um arco de entrada	46
Figura 3.5 - Conversão de transição fonte.....	46
Figura 3.6 - Conversão de transição sumidouro	47
Figura 3.7 - Disparo da transição.....	48
Figura 3.8 - Conversão completa RPC para LLD	49
Figura 3.9 - Passos de conversão de RPC para LLD.....	51
Figura 3.10 - Conversão automática de RPC para LLD em Java.....	53
Figura 4.1 - Célula de manufatura robotizada	56
Figura 4.2 - Célula de manufatura robotizada - diagrama.....	57
Figura 4.3 - Modelagem do FMS em RPC.....	58
Figura 4.4 - Linha de programa do CLP chamando respectiva sub-rotina.....	59
Figura 4.5 - sub-rotina do CLP condição de saída de p_1	59
Figura 4.6 - Programa de CLP condição de chegada em p_1	60
Figura 4.7 - RPC com uso de RCCD.....	62
Figura 4.8 - RPC com fusionamento de controladores.....	64
Figura 4.9 - Programa de CLP com supervisor	65
Figura 4.10 – Alteração de sub-rotina do CLP	66
Figura 4.11 – Alteração de sub-rotina do CLP	66

LISTA DE TABELAS

Tabela 2.1 - Variáveis da programação de CLP	41
Tabela 2.2 - Exemplos de alocação de memórias de CLP.....	41

LISTA DE ABREVIATURAS E SIGLAS

CLP	Controlador lógico programável
CLPs	Controladores lógicos programáveis
CPN Tools	<i>Computer Tool for Colored Petri Nets</i>
CPU	<i>Computer Process Unit</i>
FC	Fusão de controladores
FMS	<i>Flexible manufacture system</i> (sistema de manufatura flexível)
IEC	<i>International Electro-technical Comission</i>
LLD	<i>Ladder Logic Diagram</i>
MV	<i>Manipulated variable</i>
PV	<i>Process Variable</i>
RCCP	Restrições de controle sobre cores decompostas
RP	Rede de Petri
RPC	Rede de Petri Colorida
RPCs	Redes de Petri Coloridas
RPs	Rede de Petri
SED	Sistema a eventos discretos
SEDs	Sistemas a eventos discretos
SP	<i>Set point</i>
TCS	Teoria do Controle Supervisório
Twido Suite	<i>Software de programação de CLP</i> proprietário da empresa Schneider Electric
UFC	Universidade Federal do Ceará
XML	<i>eXtensible Markup Language</i>

LISTA DE SÍMBOLOS

A	Conjunto de arcos orientados
b	Inteiro representativo de quantidade de invariantes de lugar
B_i	Matriz das restrições com as fichas complementares de fluxo
C	Matriz de referência de incidência da RPC
Col	Função associada ao conjunto de cores
D	Matriz de incidência da RPC
E	Função de expressão de arco
G	Gerador associado a linguagens
Gd	Função de guarda que atribui uma guarda para cada transição t
\mathcal{H}	Matriz de incidência de fusão de controlador
I	Matriz de pré-condições
$IO.x$	Endereço de entrada do barramento do CLP
Ini	Função de inicialização
INT	Conjunto de números inteiros que definem uma cor.
k	Peso de um arco qualquer que liga lugar e transição.
L	Linguagem L
\vec{L}	Pré-fechamento ou fechamento de L
L_m	Linguagem marcada
M	Vetor de marcação em uma rede de Petri.
MWx	Memória de dados de CLP
m	Quantidade de lugares de uma Rede de Petri.
$m(p_i)$	Marcação relativa ao lugar p_i de uma Rede de Petri.
$M_{MS}(p_i)$	Marcação relativa ao lugar p_i de uma Rede de Petri Colorida.
$M(p_m)$	Marcação relativa ao lugar p_i de invariante de lugar.
M_0	Marcação inicial de uma Rede de Petri.
M_s	Variável de folga da equação de invariantes de lugar
\mathbb{N}	Conjunto dos números naturais
n	Quantidade de transições de uma Rede de Petri.
N	Rede de Petri definida como quádrupla $N = (P, T, I, O)$
O	Matriz de pós-condições

P	Conjunto de lugares de uma rede de Petri.
p_i	Lugar em uma rede de Petri
Q	Conjunto de estados de G
$Q0.x$	Endereço de saída do barramento do CLP
q	Estado de um gerador
$Q_m \subseteq Q$	Conjunto de estados marcados
S	Supervisor externo
T	Conjunto de transições de uma rede de Petri.
t_j	Transição em uma rede de Petri.
V	Conjunto finito de variáveis associadas aos conjuntos de cores
x_n	Variável
Δ	Conjunto finito de conjuntos de cores não vazios
$\delta: \Sigma \times Q \rightarrow Q$	Função de transição de estados
ϵ	Sequência nula
Σ	Conjunto de eventos
$\Sigma: Q \rightarrow 2^\Sigma$	Conjunto ativo de eventos num estado
Σ^*	Conjunto de todas as sequências finitas em Σ
z_i	Inteiro representativo de quantidade de invariantes de lugar

SUMÁRIO

1 INTRODUÇÃO	16
1.1 IMPORTÂNCIA DO TRABALHO	17
1.2 REVISÃO BIBLIOGRÁFICA	18
1.3 MOTIVAÇÃO E PROBLEMATICA	21
1.4 OBJETIVO GERAL	21
1.4.1 <i>Objetivos específicos</i>	22
1.5 ESTRUTURA DO TRABALHO	22
2 CONCEITO PRELIMINARES	23
2.1 CONCEITOS DE REDES DE PETRI.....	23
2.1.1 <i>Formalismo matemático de uma RP</i>	25
2.1.2 <i>Extensões das Redes de Petri</i>	27
2.2 REDES DE PETRI COLORIDAS	28
2.3 CONTROLE SUPERVISÓRIO	30
2.3.1 <i>Controle Supervisório pelo Método dos Invariantes</i>	31
2.3.2 <i>Controle Supervisório pelo Método Restrições de Controle Sobre Cores Decompostas</i>	32
2.3.3 <i>Fusão de controladores no Método RCCD</i>	34
2.4 CONTROLADORES LÓGICO PROGRAMÁVEIS (CLPs) E LINGUAGEM.....	35
2.4.1 <i>Definição de CLP</i>	35
2.4.2 <i>Diagrama de Blocos do CLP</i>	36
2.4.3 <i>Operação do CLP</i>	38
2.4.4 <i>IEC 61131-3 e a Linguagem Ladder</i>	38
3 ALGORITMO DE CONVERSÃO DE REDES DE PETRI COLORIDAS PARA LADDER LOGIC DIAGRAM (LLD)	43
3.1 DEFINIÇÕES E CORRELAÇÃO ENTRE RPC E LLD	43
3.1.1 <i>Conversão de lugares</i>	43
3.1.2 <i>Conversão de transições</i>	45
3.1.3 <i>Disparo de uma transição</i>	48
3.2 CONVERSÃO DE UMA RPC PARA LLD	49
3.3 AUTOMATIZAÇÃO DA CONVERSÃO	50
3.3.1 <i>RPC no CPN Tools</i>	50
3.3.2 <i>Uso da ferramenta Eclipse</i>	51
3.3.3 <i>Algoritmo de conversão automática de modelo RPC para LLD</i>	51
3.3.4 <i>Aplicação em JAVA</i>	52

4 ESTUDO DE CASO: CONVERSÃO DE UM MODELO DE FMS ROBOTIZADA	55
4.1 CONTEXTO INDUSTRIAL.....	55
4.2 DESCRIÇÃO DE CÉLULA DE MANUFATURA FLEXÍVEL (FMS)	56
4.3 MODELAGEM DO FMS SEM CONTROLE	58
4.4 IMPLEMENTAÇÃO DO PROGRAMA SEM CONTROLE EM CLP	58
4.5 APLICAÇÃO DE RCCD.....	60
4.6 MODELAGEM DO FMS COM SUPERVISORES	62
4.7 A FUSÃO DOS CONTROLADORES DA RPC	62
4.8 MODELAGEM DO CONTROLE SUPERVISÓRIO APÓS A FUSÃO DE CONTROLADORES.....	64
4.9 IMPLEMENTAÇÃO DE CONTROLE NO PROGRAMA DE CLP	65
4.10 ÍTENS UTILIZADOS NO ESTUDO DE CASO.....	67
5 DISCUSSÕES	68
5.1 POR QUE NÃO REALIZAR TODO O CONTROLE NO PRÓPRIO ROBÔ?	68
5.2 É POSSÍVEL IMPLEMENTAR SEM O MÉTODO DE FUSÃO DOS CONTROLADORES?	69
5.3 POR QUE INCLUIR UMA MEMÓRIA DE BYTE EM VEZ DE UMA MEMÓRIA DE BIT PARA CADA LUGAR DA RPC?	69
5.4 DIFICULDADES ENCONTRADAS.....	70
6 CONCLUSÃO E TRABALHOS FUTUROS.....	71
REFERÊNCIAS	72
APÊNDICE A – ETAPAS DE EXECUÇÃO DO CONVERSOR RPC_LLD	75
APÊNDICE B - MODELOS DE RPC CONVERTIDOS AUTOMATICAMENTE PARA LLD.....	78
APÊNDICE C – ARTIGO PUBLICADO NO CONGRESSO SBAI 2013.....	93
ANEXO A – CÓDIGO PRINCIPAL DO CONVERSOR RPC PARA LLD.....	99

1 INTRODUÇÃO

Desde os tempos remotos tem-se falado em produtividade com um vislumbre de aumento de qualidade, expansão de mercado e conquista de novos clientes por meios variados. Esse cenário, entretanto, apresenta-se com extremo dinamismo em função das mudanças de comportamento das gerações; que desencadeiam novas necessidades de produtos acabados com particularidades e especificidades tais que deem ao consumidor final a sensação de ser único, por possuir determinado item recém lançado no mercado, ou ainda por fazer parte de um grupo seleto de pessoas que possuem tais itens. Nisto tem destaque a chamada obsolescência programada, onde um produto qualquer - como um aparelho celular – possui um *tack time* muito limitado e, mesmo conservando fidedignas as suas funções básicas (comunicação) é descartado pelas gerações novas em função de não ser um *release* (lançamento); o que, em última análise, fomenta o consumismo nas sociedades, obrigando-as a um ciclo de consumo, novas necessidades, novos produtos, descarte e novo consumo. Nesse sentido algumas empresas têm despontado no desenvolvimento de novos produtos que tentam trazer à humanidade produtos de um mundo virtual para um mundo real.

Este cenário é justamente interessante para as indústrias primárias, secundárias e terciárias e, portanto, para toda uma cadeia produtiva, por causa de efeito cascata bilateral. Dessa forma, mesmo uma simples alteração num aparelho celular (mantendo o exemplo supracitado), impõe mudanças sobre: a) a cadeia de fornecedores de matéria prima e de itens beneficiados; b) os processos produtivos padronizados e customizados; c) a cadeia de armazenagem, transporte e revenda; d) a reciclagem dos produtos e preservação ambiental; e) as políticas governamentais.

Isto obriga todo o arranjo produtivo mundial a estar sempre atualizando seu parque fabril e orientando os seus investimentos às demandas dos mercados emergentes; ou ainda, investindo em sistemas de manufatura flexíveis tais que suportem uma gama significativa de alterações dos produtos acabados - agora idealizados como *commodities* pelas novas gerações. Portanto, as empresas necessitam de uma capacidade de rápida reação tecnológica às demandas variáveis desse mercado.

Não obstante a isto, as imposições governamentais sobre produção mais limpa, segurança de máquinas e de operadores, impactos ambientais e sustentabilidade aumentam o desafio dos industriais com sanções e punições cada vez mais severas aliadas, é claro, a pressões

dos concorrentes em alerta por melhores oportunidades de mercado para, por assim dizer, conquistar o mercado doutras.

1.1 Importância do trabalho

“Atualmente, os sistemas produtivos industriais estão sendo reestruturados, principalmente com relação às novas estratégias de produção que incluem mudanças em como os produtos estão sendo desenvolvidos, produzidos e distribuídos” (WU et al, 2008). As organizações têm se preocupado em atingir e demonstrar políticas de segurança e saúde ocupacional, controlando seus riscos de forma consistente com suas estratégias e objetivos, segundo a norma de Administração de Saúde e Segurança Ocupacional (OSHA). Adicionalmente, a norma ISO 14001 especifica os requisitos relativos a um sistema de gestão ambiental, permitindo a uma organização formular uma política, e objetivos que levam em conta os requisitos legais e as informações referentes aos impactos ambientais significativos.

Sob a perspectiva supracitada, o uso de máquinas semiautomáticas e automáticas, especialmente de robôs nos chamados *Flexible manufacturing system* (FMS), destaca-se como elemento-chave para maximizar produtividade industrial e mitigar os riscos advindos dos complexos empresariais. Portanto, nutre-se no uso destes uma expectativa de solução ímpar quando destaca-se segurança, qualidade, repetibilidade, acurácia e disponibilidade. E, sobretudo aliado a um *payback* menor possível, o que maximiza a atratividade neste tipo de investimento em detrimento da simples manutenção das técnicas convencionais de produção manual.

Entretanto, a integração sincronizada de máquinas no processo industrial possui algumas nuances de riscos, se considerado que a ausência de técnicas de controle pode sugerir a aquisição **desnecessária** de tais máquinas, ou ainda em quantidade desnecessária. Em essência, a **problemática** enfrentada por todo o eixo de manufatura é de garantir que o custo industrial de tais máquinas seja absorvido pela multifuncionalidade das mesmas sem bloqueios, sem conflitos de operação, e com reduzida complexidade operacional e sincronismo; permitindo inclusive, as tão citadas na literatura técnica, paradas programadas para manutenção preventiva e preditiva.

O eixo industrial de máquinas e equipamentos dotados de controladores usa, em linhas gerais, uma linguagem gráfica de relés, a linguagem *Ladder Logic Diagram* (LLD). Esta

destaca-se como uma das mais comuns nos controladores lógicos programáveis (CLP) por fornecer aos técnicos e mantenedores, uma visão imediata sobre as *interfaces* das máquinas com o processo externo sobre o qual elas atuam; o que é de grande auxílio para esses em função da necessidade de rápida identificação das entradas e saídas nesta *interface*.

Ocorre que ainda há uma distância significativa entre a modelagem de um sistema a eventos discretos (SED), cujo estado da arte contempla o uso das Redes de Petri lugar-transição (RP) e das Redes de Petri Coloridas (RPC), e a conversão desta para *Ladder Logic Diagram* (LLD) supracitada. Isto porque as RPs têm sua representação gráfica muito complicada quando modelam um sistema mais complexo e, programar diretamente em LLD torna-se uma tarefa intuitiva e que depende da experiência do programador.

Nesse contexto, as redes de Petri coloridas (RPC), como mais uma ferramenta para modelagem de SED através de funções associadas aos arcos, oferecem a condição para modelagem de sistemas mais robustos e complexos sem prejuízo de visualização gráfica, com o mesmo poder de análise da RP e com a **expansão da possibilidade** de se trabalhar com recursos identificáveis diferenciados. Além disso, usando-se modelagem por RP ou RPC, pode-se fazer uso da Teoria do Controle Supervisório (TCS).

Dado este cenário, por conseguinte sistematiza-se um método estruturado de conversão de RPC para LLD no qual o programador pode alcançar controle sobre seu sistema, usufruir das multifuncionalidades das RPC e, por fim, converter a RPC para LLD aplicável num CLP qualquer de mercado (Siemens, Schneider, Rockwell Automation, National Instruments, etc.); considerando que a LLD é normatizada pela IEC 61131-3.

1.2 Revisão bibliográfica

Pesquisadores têm usado RP para modelar sistemas diversos a fim de poderem analisar seu comportamento e controlá-los de acordo com interesses específicos de cada área. E, especialmente no eixo industrial, onde predominam os sistemas de manufatura, alguns pesquisadores têm ainda lidado diretamente com LLD por ser uma linguagem amplamente usada nos projetos industriais de manufatura implementados em CLPs.

Gomma (2011) apresenta uma ferramenta em JAVA para a conversão de um modelo de SED em RP para *Ladder Logic Diagram* (LLD), a qual representa uma vantagem

para a programação restrita às RP, todavia restrita às redes de Petri lugar-transição e, portanto, não sendo possível diferenciar fichas num mesmo lugar – conceito das redes de Petri Coloridas.

Barghash *et al.* (2011) estudam o problema dos SEDs semiautomáticos em 3 níveis de funcionalidades, a partir das RP, e realizam implementações destes em CLP via LLD. O trabalho não explora, entretanto, as estratégias de conversão de RP para LLD, fator de elevada importância para projetistas.

Morais e Castrucci (2007) apresentam as técnicas *top-down e bottom-up* para modelagem de um SED em RP, e posterior conversão para LLD destacando uma tabela de correlação entre esses. Tais técnicas, embora mantenham uma programação bem estruturada, não permitem ao programador aplicar controle sem grandes alterações na RP e no programa, o que pode significar a necessidade de reconstruir todo o programa em LLD.

Lee e Hsu (2004) apresentam o LLD como linguagem amplamente aplicada para Controladores Lógicos Programáveis (CLP), e as RP como ferramenta alternativa para o controle de sequência de sistemas complexos. Propõem ainda uma abordagem baseada em regras para fornecer medidas unificadas para LLD e projetos RP, e apresentam os resultados indicativos de que as RP são superiores a LLD quando a sequência de controle torna-se mais complexa. O trabalho, todavia, apresenta-se restrito às RP lugar transição, e evidencia a característica de superioridade das RP sobre o LLD.

Uzam e Jones (1996) apresentam uma metodologia chamada *Token Passing Logic* para converter RP para controladores de sistemas de manufaturas em diagramas *Ladder*. Exploram as RP lugar-transição, e incluem novos conceitos como estruturas formais das redes. Tais conceitos visam representar eventos externos sobre transições (sensores) e lugares (atuadores) a fim de obter controle no SED a ser modelado; entretanto, resultando em aumento de complexidade da modelagem de SED.

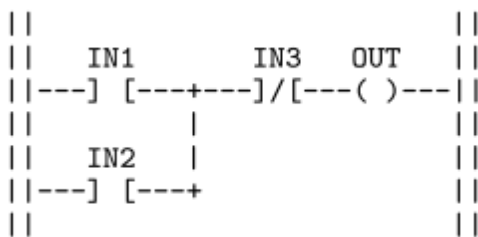
Lee e Lee (2009) apresentam um processo de conversão de LLD em modelos RP analisando atributos do programa LLD e da operação do CLP, sendo que a capacidade de análise formal é agregada da teoria de RP para programas LLD. Contudo, para tornar isso possível, conceitos pouco conhecidos, como arco de reinício, são usados e, novos conceitos são introduzidos como arco de preservação e lugar de *buffer*.

Os trabalhos apresentados por Thapa, Dangol e Wanget (2005) e por Park, Tilbury e Khargonekar (2000) destacam traduções de modelos RP para LLD. O primeiro usa uma técnica de mapeamento “um para um” a fim de gerar código de controle sem falha para um

desempenho operacional melhor, após explorar as propriedades estruturais das RP. O segundo propõe a modelagem das RP como fluxograma (sequenciamento de operação), ou com etapas onde são incluídos temporizadores(*whatdogs*) sobre os estágios de um SED modelado. Estes trabalhos usam uma notação similar ao das RP lugar transição e introduzem novos conceitos de modelagem, em RP, para torna-lá o mais próximo possível do LLD. Além disso, caso seja necessário analisar o modelo diretamente, as RP lugar transição são de difícil compreensão, pois tendem a ser demasiado grandes e, mesmo lidando com pequenos diagramas, o modelo correspondente pode tornar-se demasiado grande e ininteligível.

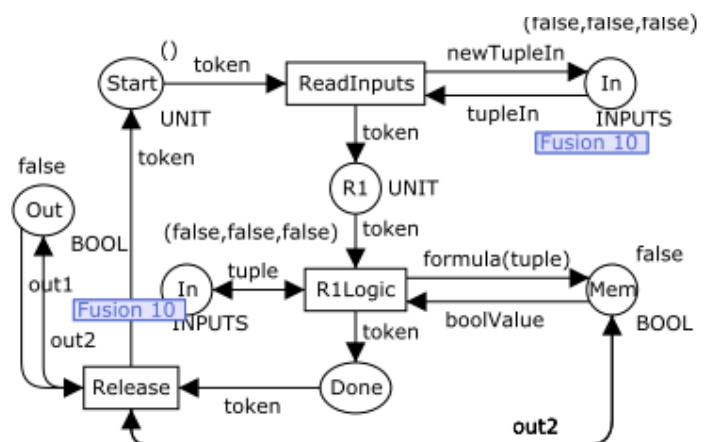
Oliveira *et al.* (2011) apresentam uma técnica de tradução de programas existentes em LLD para RPC. Nessa abordagem, desenvolve-se primeiramente um diagrama *Ladder* (pelo método intuitivo), em seguida um processo automático é executado para gerar o modelo equivalente em RPC. Na sequência, é executada a verificação do modelo em busca de erros ou situações indesejadas para permitir ao programador reconstruir o LLD até encontrar a RPC desejada. A conversão é realizada de tal forma que o modelo RPC é obtido a partir de um degrau composto apenas por contatos e bobinas, e a RPC modela: a leitura de dados do ambiente onde se encontra o CLP como disparo de uma transição *Readinputs*, o valor de memória de entrada de dados como fichas armazenadas num lugar *In*, o fluxo de energia elétrica pelo barramento do CLP como uma lugar em *R1* e a saída do CLP como lugar *Mem*. Na Figura 1.1, é apresentado um digrama LLD e, na Figura 1.2, a conversão deste para RPC, conforme proposto ainda por Oliveira *et al.* (2011).

Figura 1.1 - Exemplo de LLD



Fonte: Oliveira *et al.* (2011)

Figura 1.2 - Modelo de LLD convertido para RPC



Fonte: Oliveira *et al.* (2011)

Este trabalho de Oliveira *et al.* (2011) representa uma contribuição para a relação entre LLD e RPC e, muito embora explore o software CPN Tools¹ para implementações, limita-se à conversão de LLD para RPC e usa somente o desenvolvimento intuitivo de diagrama em LLD, o qual pode tornar-se demasiado exaustivo por depender fortemente da experiência do programador.

A conversão de retorno (RPC para LLD) não é explorada nesse trabalho. O que seria demasiado complexo, visto que o algoritmo proposto enfatiza as características físicas do CLP quando da varredura e atualização dos barramentos; tendo ainda como agravante a inclusão de inúmeros elementos de RP (lugares, transições, funções nos arcos) que ficam sem representação num programa em LLD, pois fazem parte do ciclo de *run* dos CLPs.

1.3 Motivação e problemática

Qual a distância de RPC para LLD?

Conforme destacado nos trabalhos supracitados, as pesquisas envolvendo as redes de Petri e a linguagem *Ladder Logic Diagram* (LLD) estão limitadas às RP lugar-transição e a LLD e, nesse sentido, tem destaque a **inexistência** de um método estruturado para a conversão de RPC para LLD, o qual permita a um programador de CLP usufruir do poder de abstração, análise e síntese das RPC e, na sequência, converter o referido modelo para LLD. Destaca-se que o conversor citado em Oliveira *et al.* (2011) parte de LLD para RPC mantendo o esforço de programação inicial em LLD pelo método intuitivo, ou seja, uma perspectiva contrária à proposta desta dissertação.

1.4 Objetivo geral

O objetivo geral deste trabalho é a estruturação de um método de conversão de modelos de RPCs, controladas e não controladas, para LLD.

¹ <http://cpntools.org/>

1.4.1 *Objetivos específicos*

Os objetivos específicos desta dissertação são enumerados a seguir:

- a) Explorar a modelagem de FMS em RPC com e sem controle;
- b) Explorar a programação de CLP numa FMS através de LLD;
- c) Explorar a correlação entre blocos de RPC para LLD;
- d) Desenvolver um algoritmo de conversão de RPC para LLD;
- e) Validar o algoritmo de conversão por meio do desenvolvimento de uma ferramenta de conversão automática em ambiente JAVA;
- f) Validar a conversão por meio da implementação dos resultados numa FMS;
- g) Explorar os ganhos para o programador de LLD sobre o uso do algoritmo de conversão.

1.5 **Estrutura do trabalho**

Esta dissertação está dividida em seis capítulos, os quais apresentam uma sequência que mostra passo a passo o estudo e desenvolvimento do trabalho.

No Capítulo 2 é feita a fundamentação teórica descrevendo os temas principais relacionados a este trabalho: redes de Petri, redes de Petri coloridas, controle supervisorio, restrições de controle sobre cores decompostas, controladores lógicos programáveis e as linguagens de programação de CLP.

No Capítulo 3 é apresentado um algoritmo de conversão de redes de Petri Coloridas para a linguagem *Ladder Logic Diagram (LLD)*, a correspondência entre ambas e um aplicativo desenvolvido em JAVA para realizar a conversão automatizada.

No capítulo 4 é apresentado um estudo de caso que contempla a implementação de controle supervisorio numa FMS que opera com CLP programado em LLD. Para tanto faz-se uso das RPC, RCCD, LLD e do conversor.

No capítulo 5 são apresentadas as discussões técnicas sobre a operação do FMS via CLP destacando a forma de funcionamento, os métodos adotados e os ganhos.

No Capítulo 6 são feitas as conclusões e as propostas de trabalhos futuros destacando-se os pontos importantes e contribuições do trabalho desenvolvido.

2 CONCEITO PRELIMINARES

Neste capítulo serão apresentados os conceitos de Redes de Petri (RP), Redes de Petri Coloridas (RPC), Teoria do Controle Supervisório (TCS), Controle pelo método dos Invariantes, teoria das Restrições de Controle sobre Cores Decompostas (RCCD) e Fusão de Controladores (FC), Controladores Lógicos Programáveis (CLP) e o *Ladder Logic Diagram* (LLD).

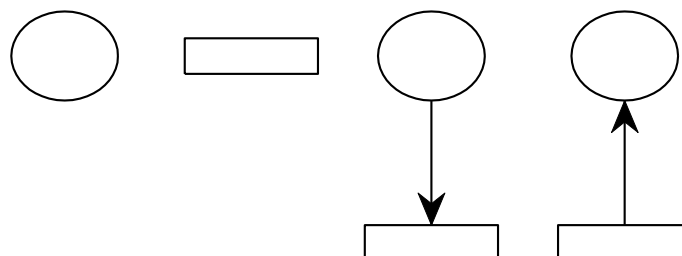
2.1 Conceitos de redes de Petri

As RP são uma ferramenta matemática e gráfica para modelagem de SED. Estas, como uma linguagem de modelagem, definem graficamente a estrutura de um sistema distribuído como um grafo direcionado com dois tipos de nós: lugares e transições, e arcos direcionados e ponderados conectando lugares a transições e transições a lugares, bem como fichas (inteiros positivos) associadas aos lugares. (MURATA, 1989).

Na Figura 2.1 é apresentada a representação gráfica destes elementos, onde os lugares (Fig. 2.1.a) são representados graficamente por círculos (ou elipses), as transições (Fig. 2.1.b) por retângulos e os arcos por setas que ligam um lugar a uma transição, ou uma transição a um lugar (nunca um lugar a outro ou uma transição a outra) (Fig. 2.1.c, d).

Figura 2.1- Representação gráfica dos elementos de RP

(a) – lugar; (b) – transição; (c) – arco; (d) – arco.

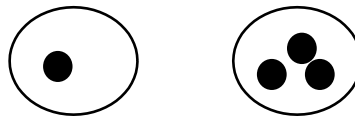


Fonte: (Autor, 2013)

Lugares podem ser marcados ou não. Um lugar marcado é representado por um ou mais pontos desenhados dentro do círculo que representa o lugar, conforme Figura 2.2. Estes pontos são denominados de *tokens* (fichas), e exatamente a quantidade de *tokens* que um lugar

possui, define o seu estado, o qual é denominado de **marcação** desse lugar. O conjunto dos lugares de uma rede de Petri, por sua vez, define o conjunto das variáveis de estado que representam o sistema modelado. Assim, as marcações de todos os lugares em um dado momento são uma característica importante da rede, representando o estado do sistema naquele momento de tal forma que o estado global, composto pelos estados de cada lugar, é chamado de **marcação da rede**.

Figura 2.2 - Lugar com token



Fonte: (Autor, 2013)

Os lugares são ainda elementos passivos, e definem o estado do sistema. As transições, por outro lado, são elementos ativos. Elas representam ações que podem ocorrer e que modificam o estado do sistema (marcação da rede). Os lugares que estão ligados a uma transição através de arcos são os que definem quando uma transição pode ocorrer e como o estado é modificado após a sua ocorrência.

Um arco ligando um lugar a uma transição (conforme visto na Fig. 2.1.c) representa uma condição que deve ser verdadeira para que aquela transição ocorra. Chama-se a este lugar de **pré-condição** da transição. “Para cada arco que liga qualquer lugar a esta transição deve existir pelo menos um *token* naquele lugar. Se esta condição for satisfeita, diz-se que a transição está habilitada para ocorrer” (MURATA, 1989). Já um arco que liga a transição a um lugar (conforme Fig. 2.1.d) representa a condição que se torna verdadeira após a ocorrência da transição. Este lugar é denominado de **pós-condição** da transição. Assim, quando esta transição ocorrer, deverá ser adicionado um *token* ao lugar que é sua pós-condição para cada arco que liga a transição a este lugar. Os *tokens* existentes nas pré-condições da transição, e que a tornaram habilitada, são removidos após sua ocorrência. Diz-se que eles foram “consumidos” pela transição. Todo este processo que acontece quando uma transição ocorre é chamado de **disparo da transição**.

Murata (1989) afirma ainda que quando mais de um arco ligar os mesmos elementos da rede, por exemplo n arcos, é convencionalmente representado por apenas um arco, acompanhado da notação numérica (rótulo) n , indicando que se trata de um arco de peso n , ou seja, que tem o mesmo valor de n arcos.

2.1.1 Formalismo matemático de uma RP

As redes de Petri podem ser descritas de formas variadas. Existe uma abordagem que as define em termos de extensão de conjunto que aceita repetição de elementos, outra que as define em termos de álgebra matricial, e também uma baseada no conceito de relações. Aqui, será utilizada a representação matricial, que é definida como se segue :

DEFINIÇÃO 2.1 Uma RP com m lugares e n transições pode ser representada por uma quádrupla :

$$N = (P, T, I, O) \quad (2-1)$$

Em que:

- ✓ $P = \{ p_1, p_2, \dots, p_m \}$ é um conjunto finito de lugares;
- ✓ $T = \{ t_1, t_2, \dots, t_n \}$ é um conjunto finito de transições, com $P \cup T \neq \emptyset$ e $P \cap T = \emptyset$;
- ✓ $I = P \times T \rightarrow \mathbb{N}$ é a função de entrada ou pré-incidência que especifica os arcos orientados dos lugares para as transições;
- ✓ $O = T \times P \rightarrow \mathbb{N}$ é a função de saída ou pós-incidência que especifica os arcos orientados das transições para os lugares, onde \mathbb{N} é o conjunto dos números naturais e $m, n \in \mathbb{N}$.

Arcos direcionados conectam lugares e transições. Um arco orientado de um lugar p_i para uma transição t_j , define p_i como sendo um lugar de entrada de t_j , denotado por $I(p_i, t_j) = 1$. Por sua vez, um arco orientado da transição t_j para o lugar p_i define p_i como sendo um lugar de saída de t_j denotado por $O(t_j, p_i) = 1$.

Se $I(p_i, t_j)$ ou $O(t_j, p_i) = k$, isto implica na existência de k arcos orientados e paralelos conectando o lugar p_i à transição t_j (ou o inverso disto). Nessa ocorrência, em uma representação gráfica, arcos paralelos conectando lugares (transições) às (aos) transições (lugares) são representados por um único arco orientado etiquetado com sua multiplicidade, ou peso k .

Uma marcação é uma atribuição de fichas aos lugares de uma RP. Fichas são um conceito primitivo em RPs (assim como lugares e transições) que residem nos lugares de tal forma que o número e a posição dessas fichas muda durante a execução de uma RP.

DEFINIÇÃO 2.2 Uma marcação M de uma RP $N = (P, T, I, O)$ é uma função do conjunto de lugares no conjunto de números inteiros não negativos, isto é, $M : P \rightarrow \mathbb{N}$.

DEFINIÇÃO 2.3 Seja P o conjunto de lugares de uma rede RP. O vetor marcação M é definido como:

$$M = (m(p_1), m(p_2), \dots, m(p_n)) \quad (2-2)$$

Em que:

$$p_i \in P;$$

$$m(p_i) \in \mathbb{N}, \text{ corresponde à marcação do lugar } p_i.$$

A associação de uma rede (estrutura) a uma marcação é chamada de rede marcada, e é definida a seguir:

DEFINIÇÃO 2.4 Uma rede de Petri marcada é um par

$$RP = (N, M_0) \quad (2-3)$$

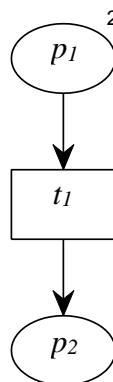
Em que:

N é uma rede de Petri tal que $N = (P, T, I, O)$;

M_0 é o vetor marcação que representa a marcação inicial da rede.

Na Figura 2.3 é apresentada uma RP tal que são colocadas 2 fichas (unidades de recursos exatamente iguais) no lugar p_1 e que serão armazenados em p_1 ou em p_2 , caso satisfeitas as condições de disparo da transição t_1 . Perceba que a marcação inicial desse exemplo vale $M_0 = [2 \ 0]$ e que a transição t_1 está habilitada, podendo ocorrer o disparo ou não.

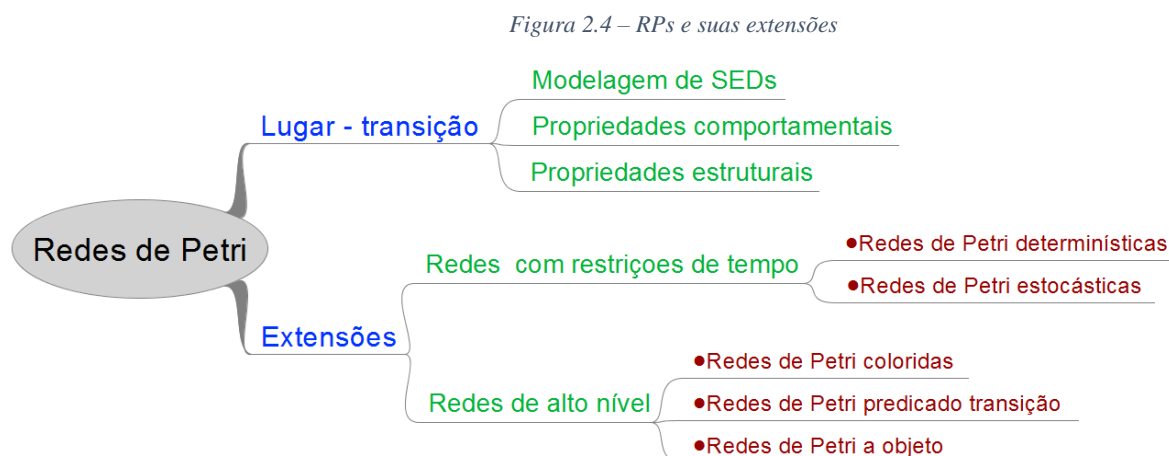
Figura 2.3 – Exemplo de Rede de Petri (RP)



Fonte: (Autor, 2013)

2.1.2 Extensões das Redes de Petri

Dado seu potencial de análise e síntese para modelagem de SED, as RP apresentam dois tipos principais de extensões, conforme apresentado na Figura 2.4.



Fonte: (Autor, 2013)

As redes de Petri **lugar – transição** representam um marco no cenário da modelagem de SED. Possuem estruturas tais como: sequência, concorrência, sincronismo, caminhos alternativos, repetição, alocação de recursos e conflito. A partir destas, tornando possível a análise de propriedades comportamentais de uma rede (alcançabilidade, limitação, vivacidade, reversibilidade, estado de passagem, cobertura, persistência e conservação) bem como, a análise das propriedades estruturais (que não dependem da marcação inicial da rede): invariantes de lugar e invariantes de transições.

As redes de Petri **determinísticas** caracterizam a modelagem de sistemas em tempo real e a análise de desempenho, ao passo que as redes de petri **estocásticas** têm foco na análise de desempenho de SED, mais precisamente sobre eventos probabilísticos.

As redes de Petri **coloridas** destacam-se pela possibilidade de modelar “dados diferentes num mesmo conjunto e concatená-los” conforme será explorado na subsecção 2.2. As redes de Petri **Predicado transição**, por sua vez, caracterizam-se pela associação de equações diferenciais que definam relações específicas entre eventos modelados; ao passo que as Redes de Petri orientadas a **objetivo** “propõem-se a fazer com que várias das características das redes de Petri e da Orientação a Objetos complementem-se e contribuam para a construção, simulação e validação de modelos” da dinâmica de sistemas (GUERRA, 2005).

Para o estudo proposto nesta dissertação as redes de Petri de alto, nível subclasse redes de petri coloridas, são a abordagem de destaque, uma vez que estas reúnem as características de interesse nesta área de estudo conforme enfatizado na introdução.

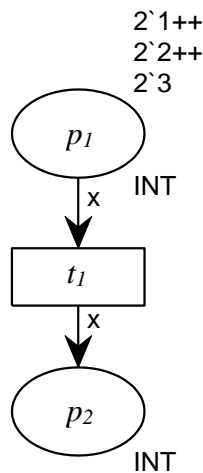
2.2 Redes de Petri Coloridas

Uma representação mais compacta de uma RP é obtida através da associação, a cada ficha, de um conjunto de dados, denominados cores da ficha. O conceito de cor é análogo ao conceito de tipo, comum nas linguagens de programação. Nesse sentido, as redes de Petri coloridas (RPC) são apresentadas como uma extensão às RP que combinam a estrutura de uma RP com uma linguagem de programação. (JENSEN e KRISTENSEN, 2009)

Desta forma, pode haver diferenciação de fichas e associação de variáveis e funções aos arcos da rede em substituição aos pesos dos arcos das RP. Tal diferenciação de fichas se dá pelas cores assumidas por cada ficha, definidas previamente. A marcação de cada lugar de uma RPC é um subconjunto do conjunto de cores associado ao lugar. Sendo assim, as RPCs são capazes de modelar sistemas mais complexos de forma mais compacta e de visualização simplificada.

Na Figura 2.5 é apresentado um modelo de RCP com destaque para a diferenciação de fichas em um mesmo lugar. Assim, os lugares p_1 e p_2 podem armazenar, respectivamente, recursos diferenciados.

Figura 2.5 - Rede de Petri Colorida



Fonte: (Autor, 2013)

Perceba a existência de um tipo “INT” (conjunto de cores inteiro) associado aos lugares. Perceba ainda que no lugar p_1 há três fichas do tipo “INT”, mas de valores diferentes; sendo duas cópias de cada valor.

O conjunto de cores, e as fichas associadas a um determinado lugar, definem as operações que serão realizadas em função das transições e expressões dos arcos.

A variável x , presente no arco que liga o lugar à transição ou a transição ao lugar, pode assumir o valor de qualquer uma das fichas do lugar p_1 . O lugar p_2 , por sua vez, apresenta um conjunto de cores igual ao do lugar p_1 .

O conjunto de cores de uma transição indica as diferentes maneiras de como ela pode disparar. Cada cor de transição, nesse caso, corresponde a uma das transições da rede ordinária equivalente, ou seja, cada transição da rede ordinária que é dobrada numa transição da rede colorida vai corresponder a uma cor do conjunto de cores.

DEFINIÇÃO 2.2.1 Segundo Jensen e Kristesen (2009), uma RPC associada a uma marcação inicial é uma nômupla:

$$RPC = (P, T, A, \Delta, V, Col, Gd, E, Ini) \quad (2-4)$$

Onde:

- ✓ $P = \{p_1, p_2, \dots, p_m\}$ é um conjunto finito de lugares;
- ✓ $T = \{t_1, t_2, \dots, t_n\}$ é um conjunto finito de transições tal que $P \cap T = \emptyset$;
- ✓ $A \subseteq P \times T \cup T \times P$ é um conjunto de arcos orientados;
- ✓ Δ é um conjunto finito de conjuntos de cores não vazios;
- ✓ V é um conjunto finito de variáveis associadas aos conjuntos de cores;
- ✓ Col é uma função associada ao conjunto de cores que atribui um conjunto de cores para cada lugar;
- ✓ Gd é uma função de guarda que atribui uma guarda para cada transição t_i ;
- ✓ E é uma função de expressão de arco que atribui uma expressão de arco para arco;
- ✓ Ini é uma função de inicialização que atribui uma expressão de inicialização para cada lugar p_i .

2.3 Controle supervisório

Uma vez modelado um SED em RP, a análise das propriedades supracitadas indicará a existência ou não de situações indesejadas no sistema e, portanto, tornando necessária a aplicação de uma estratégia de controle, ou supervisão, para impedir que o sistema acesse uma marcação morta (*deadlock*), ou mesmo um laço tal que mantenha o sistema preso (*livelock*).

De um modo geral, o desafio de encontrar um supervisor é resolvido pela Teoria do Controle Supervisório (TCS). Nesta, separa-se o sistema a ser controlado (*open loop dynamics*) do controlador (*feedback control*) e desenvolve-se: a modelagem, especificação do comportamento e síntese do supervisor. No processo, a síntese de um supervisor é realizada para um dado modelo com o objetivo de satisfazer uma especificação de comportamento desejada. Dessa forma, o supervisor é um agente externo que possui habilidade de observar os eventos gerados pelo sistema e influenciar no seu comportamento através de entradas de controle. Em malha fechada, a ação de controle do supervisor garante que o funcionamento do sistema esteja de acordo com uma especificação dada.

Na TCS, segundo Barkaoui, Chaoui e Zouari (1997) e Cury (2001), um SED é modelado por um par de linguagens, L e L_m , bem definidas, mais o conjunto de eventos Σ . L representa o conjunto de todas as sequências de eventos que o sistema pode gerar a partir de seu estado inicial e L_m é a linguagem marcada que representa as tarefas completadas do sistema.

Seja:

$$G = (Q, \Sigma^*, \delta, \Sigma, q_0, Q_m) \quad (2-5)$$

o gerador associado a estas duas linguagens. Por definição, Σ^* representa o conjunto de todas as sequências finitas em Σ , incluindo a sequência nula ϵ ; Q é o conjunto de estados de G ; $\delta: \Sigma \times Q \rightarrow Q$ é a função de transição de estados, tal que, $\delta(\epsilon, q) = q$ e $\delta(\sigma, q) = q'$ onde $q, q' \in Q$ e $\sigma \in \Sigma$; $q_0 \in Q$ é o estado inicial de G e $Q_m \subseteq Q$ é o conjunto de estados marcados. $\Sigma: Q \rightarrow 2^\Sigma$ denota o conjunto ativo de eventos num estado, tal que, $\delta(q, s)$ é definida.

Segue que:

$$L = \mathcal{L}(G) = \{s \in \Sigma^*: \delta(q_0, s) \text{ é definida}\}, \quad (2-6)$$

$$L_m = \mathcal{L}_m(G) = \{s \in \mathcal{L}(G): \delta(q_0, s) \in Q_m\}, \quad (2-7)$$

G é dito não-bloqueante (trim) quando

$$\mathcal{L}(G) = \overline{\mathcal{L}_m(G)}, \quad (2-8)$$

caso contrário é dito bloqueante.

Quando o gerador G tem um “comportamento proibido” (gera tarefa não especificada), o objetivo do controle é então restringir o comportamento do sistema não controlado (descrito por G), dentro dos limites do “comportamento não-proibido”, dado como um subconjunto de $\mathcal{L}(G)$; esse controle pode ser alcançado conectando o sistema em malha com um supervisor S (um controlador externo), sob a restrição que S nunca desabilite um evento não controlável. O sistema resultante em malha fechada denotado por S/G , a linguagem gerada $\mathcal{L}(S/G) \subseteq \mathcal{L}(G)$ e sua linguagem marcada consiste exatamente das sequências marcadas de G que estão sob o controle de S . O “comportamento não-proibido” é dado como um subconjunto de $\mathcal{L}_m(G)$. Assim, uma propriedade importante que S deve satisfazer é ser um supervisor próprio, dessa forma, o sistema sob supervisão S/G é não-bloqueante, ou seja,

$$\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}, \quad (2-9)$$

Além dos métodos de supervisão que exploram o gerador (**automatus**) pela TCS, outras técnicas tem alcançado elevada eficácia na aplicação do controle e supervisão de SEDs (**RP**). Dentre estes destacam-se: o método dos invariantes e o método das restrições de controle sobre cores decompostas, ambos estruturados sobre as RP e RPC, conforme apresentado nos itens a seguir.

2.3.1 Controle Supervisório pelo Método dos Invariantes

Os invariantes de lugar são uma das propriedades estruturais das redes Petri que dependem apenas da topologia da rede e independem de sua marcação; portanto, são importantes para a análise dinâmica de um processo modelado. São correspondentes aos conjuntos de lugares nos quais a **soma de fichas permanece constante** para todas as marcações alcançáveis pela rede. Dessa forma, a análise dos invariantes de lugar pode colaborar para a validação lógica de um modelo desenvolvido em redes de Petri.

Dada uma RP com m lugares e n transições, o objetivo do controle pelo método de invariantes de lugar é forçar os processos a obedecerem a restrições, tais que,

$$z_1 M(p_i) + z_2 M(p_j) \leq b, \quad (2-10)$$

em que z_1, z_2 e b são inteiros e $M(p_m)$ é a marcação do lugar p_i . Essas restrições são equivalentes a um conjunto de restrições generalizadas mutuamente excludentes. Nesse caso, com o auxílio de uma variável de folga $M_s > 0$, pode-se ter

$$z_1 M(p_i) + z_2 M(p_j) + M_s = b, \quad (2-11)$$

A variável de folga representará a adição de um lugar de controle cuja marcação irá satisfazer à condição de igualdade. Esse argumento torna-se suficiente para a síntese do supervisor, segundo Moody e Antsaklis (1998).

2.3.2 Controle Supervisório pelo Método Restrições de Controle Sobre Cores Decompostas

O método restrições de controle sobre cores decompostas (RCCD) é um método que usa como modelo as RPC, e é baseado na TCS desenvolvida por Moody e Antsaklis (1998).

Conforme Menezes, Barroso e Prata (2012), o RCCD faz a análise da matriz de incidência da RPC decompondo esta matriz em outras matrizes de coeficientes de uma mesma variável e, desta forma, propõe uma solução para síntese de supervisores em RPC.

Como um lugar em uma RPC pode conter fichas de diferentes cores, as quais podem ou não sofrer restrições (controle) independentemente, definem-se dois conjuntos de fichas:

- Fichas de restrições, que são as fichas que estão submetidas a alguma restrição (controle);
- Fichas complementares de fluxo, que são as fichas do conjunto que não sofrem restrições.

Dado um SED não controlado, a especificação de controle e seu modelo em RPC, a aplicação do método requer a definição da matriz D das funções associadas aos arcos para a separação de variáveis, segundo as cores. A matriz D , denominada **matriz de incidência**, é definida, tal que, os seus elementos são as funções ou variáveis associadas aos arcos da rede não controlada com os coeficientes de incidência.

Didaticamente, consegue-se visualizar a decomposição de cores através da separação das variáveis associadas a elas. Após a decomposição de cores conservam-se, para efeito de análise, apenas os coeficientes que definem os lugares de entrada e saída das transições. Encontra-se assim, a matriz C , **matriz de referência de incidência**, ou seja, se um elemento dessa matriz é igual a 1, isso indica que o lugar correspondente é saída de uma dada transição, se o elemento é igual a “-1”, então o lugar é entrada da transição. O elemento é igual a zero quando o lugar não é entrada nem saída da transição. A seguir, de posse da marcação inicial da rede não controlada e das restrições, pode-se obter um supervisor seguindo os seguintes passos:

1. Construir a matriz D , denominada matriz de incidência da RPC não controlada, cujos elementos representam as respectivas expressões associadas aos arcos.
2. Decompor a matriz D em uma soma de matrizes onde cada parcela é uma matriz cujos elementos são relativos a uma mesma variável x_n , tal que, $D = D^{x1} + D^{x2} + \dots + D^{xn}$.
3. Construir C , denominada a matriz de referência de incidência da RPC não controlada, a partir da matriz D , considerando-se apenas os coeficientes de incidência relativos às expressões associadas aos arcos.
4. Decompor a matriz C em uma soma de matrizes cujos elementos, coeficientes de incidência, estejam associados a uma mesma variável tomando como referencial as parcelas D^{xn} de D decomposta, tal que, $C = C^{x1} + C^{x2} + \dots + C^{xn}$.
5. Para cada restrição imposta, adicionar o número de fichas correspondente à marcação inicial das outras cores (fichas complementares de fluxo), que são representadas pela mesma variável.
6. Calcular a(s) matriz(es) de incidência(s) e a(s) marcação(ões) inicial(is) do(s) controlador(es) usando das equações (2-12) e (2-13) de Moody e Antsaklis (1998), modificadas e adaptadas para uma RPC.

$$C_c = -LC \quad (2-12)$$

$$M_{c_0} = b - LM_0 \quad (2-13)$$

2.3.3 Fusão de controladores no Método RCCD

Dado que vários controladores representados por distintos lugares de controle foram sintetizados a partir do método RCCD, pode-se definir estes controladores como um só conjunto, sem prejuízo para o sistema modelado, o que leva à Fusão de Controladores (FC). Esse controlador resultante agrega toda ação dos antigos lugares de controle (*Controladores*).

Dada uma RPC controlada através de mais de um lugar de controle, é sempre possível fazer a fusão desses controladores.

Sejam B_i e M_0 , respectivamente, a matriz das restrições com as relativas fichas complementares de fluxo e a marcação inicial da RPC controlada, então a marcação inicial e a matriz de incidência de fusão do supervisor são, respectivamente:

$$M_0 = \sum_{i=1}^r B_i + \mathcal{L}_i M_0, \quad (2-14)$$

$$\mathcal{H} = -(\sum_{i=1}^r \mathcal{L}_i)C. \quad (2-15)$$

A prova matemática deste teorema pode ser vista em Menezes, Barroso e Prata (2012).

2.4 Controladores Lógico Programáveis (CLPs) e Linguagem

Dentre os dispositivos industriais capazes de operacionalizar os SEDs por meio de programação, simulação, controle, monitoramento e supervisão, tem-se que os CLPs apresentam-se como elementos indispensáveis para cumprirem tais premissas.

Por esta razão, tem sido expandido o seu uso integrado aos variados processos industriais. O que é realizado em níveis que vão desde o controle simples de uma máquina, à integração de plantas industriais com CLPs; conectados hierarquicamente, comunicando-se por meio de redes industriais no barramento de campo, ou chamado *fieldbus*.

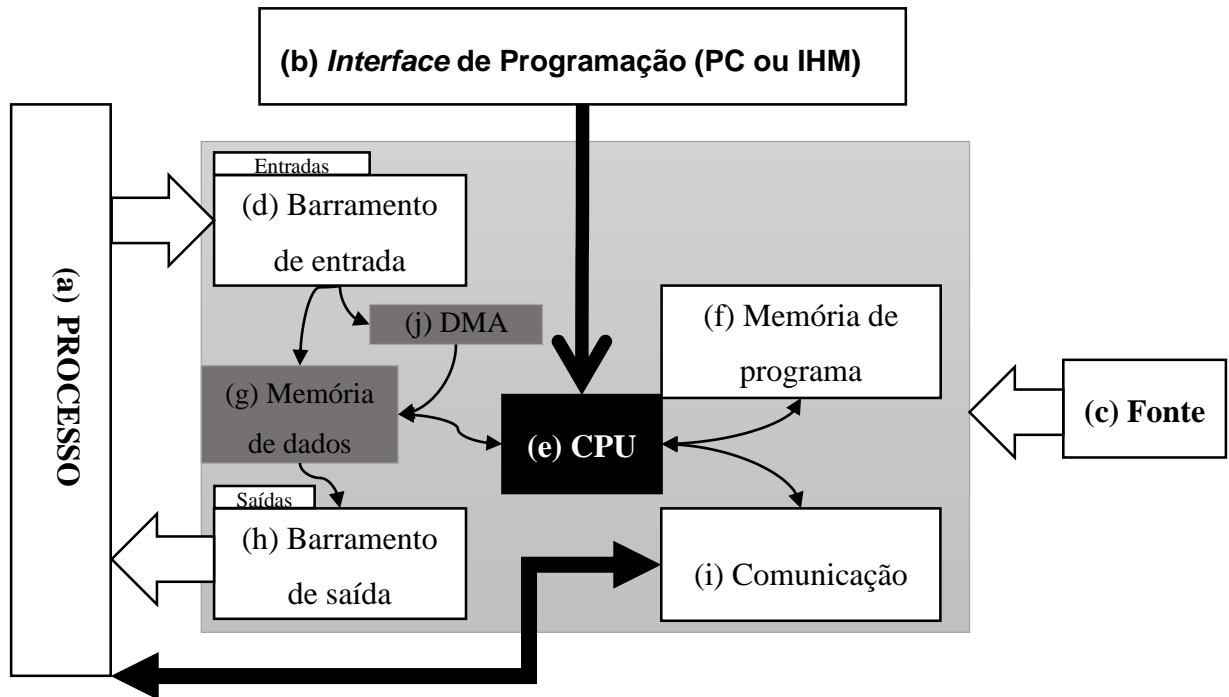
2.4.1 Definição de CLP

O Controlador Lógico Programável (CLP) foi desenvolvido no final da década de sessenta como uma alternativa ao controle de dispositivos eletromecânicos que eram feitos através de relés. A proposta era melhorar a qualidade e a eficiência dos processos, reduzindo custos com reposição dos relés e aumentando a confiabilidade do sistema de controle, segundo Georgini (2007). Nesse contexto, o CLP alavancou a automação industrial tornando-se indispensável seu uso na indústria nas décadas que se seguiram. Segundo Silvestre (2010), por ser mais robusto que um computador pessoal, o CLP é capaz de suportar temperaturas mais altas, além resistir a perturbações como ruído elétrico e vibrações, entre outras características. Quanto ao funcionamento, o CLP é capaz de funcionar com múltiplas combinações de entrada e saída.

2.4.2 Diagrama de Blocos do CLP

Na Figura 2.6 é apresentado o diagrama de blocos de um CLP.

Figura 2.6 - Diagrama de blocos do CLP



Fonte: (Autor, 2013)

Perceba que os elementos de base de um CLP (Figura 2.6) são:

- a) **processo** – o qual representa um sistema externo real com eventos a serem controlados pelo CLP;
- b) **interface de programação** – a partir da qual o programador desenvolverá o algoritmo de controle de interesse para seu processo através de um *software* via PC, ou ainda diretamente através da *Interface Humano Máquina* (IHM);
- c) **Fonte** – alimentação de tensão de trabalho dos circuitos internos do CLP; normalmente trabalha com entrada bivolt (90 – 240 Vac com filtro de ruídos - o que aumenta sua robustez no uso industrial) e oferece alimentação externa de 24 Vcc para uso nos barramentos de “saída a relé”;
- d) **Barramento de entrada** – onde, a partir do processo monitorado, é realizada a leitura de variáveis de processo (*Process Variable* - PV), que serão usadas pelo controlador

afim de atuar no mesmo. Quando necessário, o barramento é expandido com o uso de cartões analógicos a fim de processar grandezas contínuas em adição às discretas;

e) **CPU** – ou unidade de processamento de dados, é o processador do CLP; o qual realiza os cálculos para tomadas de decisão a partir do algoritmo gravado na memória de programa. Em alguns processadores mais modernos, uma memória interna auxiliar (**memória cache**) é utilizada para armazenar parte dos dados presentes na memória externa ao processador. Isto aumenta a velocidade de processamento (cálculos, cópias e demais rotinas de programa) uma vez que, de posse desses cópias de dados, o processador não precisa carregá-los a partir da memória externa para realizar os cálculos devidos.

f) **Memória de programa** – local reservado na memória do CLP para guardar o algoritmo de controle do processo, normalmente via EEPROM (*Electronic Erasable Programmed Read Only Memory*) a fim de permitir ao programador produzir diversos programas e, quando necessário, selecionar um programa específico para monitorar o processo;

g) **Memória de dados** - local reservado na memória do CLP para registrar os dados do processo (mundo externo) sobre o qual está aplicado um controle. Estas memórias podem crescer em função da necessidade do programador de expandir o registro de dados sobre: variáveis de processo (*Process Variable* - PV), variáveis manipuladas (*Manipulated variable* - MV), variáveis desejadas (*Set point* – SP) ou, sobre variáveis auxiliares do monitoramento. Destaca-se que a *expertise* do programador faz total diferença nesta etapa e, portanto, podendo tornar excessivamente complexo e com redundâncias em demasia, um algoritmo simples; ou o inverso. O que pode permitir ao industrial redução de custos quando da aquisição de CLPs;

h) **Barramento de saída** - onde, a partir dos resultados da CPU, é realizada uma modificação no valor de suas portas (saídas) de modo manter, ou corrigir, um estado no processo através das MV. Quando necessário, também este barramento é expandido com o uso de cartões analógicos a fim de permitir a manipulação de grandezas contínuas. Na prática, a maioria dos cartões analógicos possuem um mínimo de uma entrada e uma saída analógica compactados numa mesma unidade física e, acoplados mecânica e eletricamente ao CLP;

i) **Comunicação com o processo** – onde, através de uma IHM ou de um *software* de supervisão, o processo real pode ter uma representação remota “animada”; podendo ainda ser com ou sem liberdade para interferência humana.

j) **DMA** (*Direct memory Access*) – acesso direto ao sistema de memória do dispositivo, independentemente da CPU. Os dispositivos que têm os canais de DMA podem transferir dados aos periféricos com muito menos perdas gerais de processamento. Isto representa um elevado ganho de velocidade do processador e de integridade dos dados trabalhados. Sem o DMA, entretanto, quando a CPU está realizando a transferência de dados entre o processador e os periféricos, a mesma fica ocupada durante toda a duração da operação de leitura ou escrita, e portanto, indisponível para realizar outras tarefas inerentes ao processo. O que, em última análise, pode significar a perda de dados do processo monitorado e erro de operação.

2.4.3 Operação do CLP

A operação de um CLP consiste, portanto, na execução de seu ciclo de varredura ou *scan*. A execução do referido ciclo corresponde à realização completa das seguintes atividades: a) atualização das entradas; b) execução de código do programa e c) atualização das saídas. Na etapa de atualização de entradas, o CLP realiza a leitura de todas as informações de entrada, associando essas informações a *bits* internos do mesmo. Na etapa de execução do código do programa, o CLP executa as informações do código do programa. Estas informações são responsáveis por definir a relação entre a condição das informações de entrada e saída, que representam a lógica de controle que deve ser realizada pelo CLP. Por fim, na última etapa, o CLP carrega no barramento de saída os valores referentes ao resultado da lógica combinacional de entradas e processamento de dados.

2.4.4 IEC 61131-3 e a Linguagem Ladder

Considerando a necessidade de padrões para CLPs, em face à variedade de *softwares* (dialetos de programação) e *hardwares* (ausência de padrões técnicos de segurança), em 1979 foi designado, por parte da comunidade industrial internacional, um grupo de trabalho com o IEC (*International Electro-technical Commission*) voltado para este propósito. Este grupo teve como objetivo analisar o projeto completo de CLPs (inclusive *hardware*), instalação, testes, documentação, programação e comunicações. Sendo designadas 8 frentes de trabalho para desenvolver diferentes partes do padrão para CLPs. A primeira parte deste trabalho refere-se às Informações Gerais, a segunda destaca os Requerimentos de Testes de Equipamentos, a

terceira formaliza as Linguagens de Programação, a quarta trata sobre Guias de Usuários e a quinta parte disserta sobre o Serviço de Mensagens em CLPs.

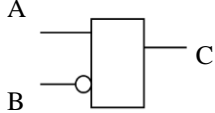
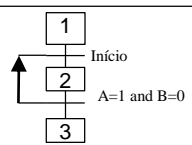
A IEC 61131-3 é, portanto, o padrão global para programação de controladores industriais que atende à necessidade profissional supracitada. Assim, a partir de uma interface de programação com elementos padrões, torna-se possível a pessoas com diferentes habilidades e formações, criar elementos distintos de um programa durante estágios diferentes do ciclo de vida de um *software*, tais como: especificação, projeto, implementação, teste, instalação e manutenção. O padrão inclui as linguagens:

- ✓ Lista de instruções (*Instruction List – IL*) – é uma linguagem textual, próxima do código de máquina, usada aplicada na solução de problemas simples onde existem poucas quebras no fluxo de execução, ou para produzir um código otimizado para trechos de performance crítica em um programa.
- ✓ Texto estruturado (*Structured Text – ST*) - é uma linguagem muito poderosa com suas raízes em Ada, Pascal e “C”. Pode ser usada na definição de blocos de função complexos, que, por sua vez, podem ser utilizados com quaisquer outras linguagens, e no detalhamento das ações e transições de um programa.
- ✓ Diagrama de blocos funcionais (*Function Block Diagram – FBD*) - é muito comum para a indústria de processos. Ele expressa o comportamento de funções, blocos de funções e programas como um conjunto de blocos gráficos interconectados, como em um desenho de circuito eletrônico. Assemelha-se à representação de um sistema em termos do fluxo de sinais entre os elementos de processamento.
- ✓ Diagrama de lógica Ladder – muito comum nas indústrias e, segue um padrão símile aos contatos abertos e fechados de um circuito sequencial a relé.

- ✓ Grafcet (*Grphe Fonctionnel de Command Etape-Transition*) - linguagem desenvolvida por universidades francesas para representação de processos sequenciais baseada nas Redes de Petri. O Grafcet se tornou um padrão europeu com a introdução do padrão IEC 848: *Preparation of function charts for control system*. E, decorrente disto, a norma IEC 61131-3 introduziu algumas modificações no referido padrão visando integrar esta linguagem às demais linguagens da norma.

Na Figura 2.7 é apresentado um trecho de programa que associa duas variáveis de entrada e uma de saída ($A \cdot \bar{B} = C$); implementado respectivamente nas linguagens: IL, ST, FBD, LLD e Grafcet.

Figura 2.7 - Trecho de programa nas linguagens da IEC 61131-3

(IL)	(ST)	(FBD)	(LLD)	(Grafcet)
LD A ANDN B ST C	C = A AND NOT B		A B C -- ---- / ----()-	

Fonte: (Autor, 2013)

Natale (2007) apresenta detalhadamente modelos de uso das linguagens supracitadas numa visão ampla de controladores indo, desde os *flip-flops*, aos circuitos combinacionais de maior complexidade, temporizados e, incluindo elementos analógicos; o que é oportuno, especialmente para programadores que ainda têm um pouco de dificuldade no contato com essas linguagens.

Cumprindo com a estruturação de um padrão para programação dos CLPs, a norma IEC 61131 define ainda **variáveis de representação direta**. Estas variáveis são o modo de acesso e uso da memória de dados (registradores) do CLP num programa. Elas permitem leitura e escrita de dados em posições conhecidas de memória, tais como entradas, saídas e endereços internos (registradores).

As variáveis de representação direta (variáveis locais) têm seu uso restrito aos programas e, sua notação utilizada é padronizada para permitir a portabilidade entre linguagens. Todas começam com o caractere %, seguido de uma ou duas letras e números inteiros.

Na Tabela 2.1 são apresentadas as variáveis dos CLPs segundo a referida norma, enquanto na Tabela 2.2 são apresentados exemplos de uso destas variáveis.

Tabela 2.1 - Variáveis da programação de CLP

Prefixo	Interpretação	Tipo de dado
I	<i>Input</i> : Recebe os valores das variáveis analógicas e discretas dos módulos de entrada.	-
Q	<i>Output</i> : Armazena os valores a serem escritos nos dispositivos externos.	-
M	<i>Memória Interna</i> : Armazena valores intermediários.	-
X	Tamanho de um bit	Booleano
None	Tamanho de um bit	Booleano
B	Tamanho de um <i>Byte</i> (8 bits)	Byte
W	Tamanho de uma palavra (<i>Word</i> - 16 bits)	Word
D	Tamanho de duas palavras (<i>Double Word</i> - 32 bits)	DWord
L	Tamanho de palavra longa (<i>Long Word</i> - 64 bits)	LWord

Fonte: (Autor,2013 – Adaptado de IEC 61131-3)

Tabela 2.2 - Exemplos de alocação de memórias de CLP

Variável	Interpretação
<i>%QX75</i> ou <i>%Q75</i>	Bit 75 da saída
<i>%IW215</i>	Entrada de palavra no endereço 215 de memória
<i>%QB7</i>	Endereço do byte de saída 7
<i>%MD48</i>	Palavra dupla na posição de memória 48
<i>%IW2.5.7.1</i>	(*)
<i>%MW1</i>	Memória de dados tipo <i>word</i> do endereço 1
<i>%MW1:=1</i>	Memória de dados tipo <i>word</i> do endereço 1 recebe o valor 1 em decimal.

Fonte: (Autor,2013 – Adaptado de IEC 61131-3)

(*) Por definição normativa, o fabricante deve especificar a correspondência entre a representação direta de uma variável e a localização física, ou lógica do item abordado na memória, entrada ou saída. Nesse sentido, quando uma representação direta é estendida com

campos inteiros separados por pontos adicionais, deve ser interpretada como um endereço físico, ou lógica hierárquica com o campo mais à esquerda, o qual representa o nível mais alto da hierarquia, com os níveis sucessivamente mais baixos que aparecem à direita. Por exemplo, a variável *% IW2.5.7.1* pode representar o primeiro "canal" (palavra) do sétimo "módulo" na quinta "prateleira" do segundo barramento de *I/O* de um sistema de controlador programável.

Para os propósitos desta dissertação **explora-se** a LLD por “... ser uma linguagem gráfica, cujos símbolos assemelham-se aos usados para representar esquemas elétricos, rapidamente assimilada e difundida” (SILVESTRE, 2010). Apesar de ter sido a primeira linguagem para programação de CLPs, o diagrama de lógica *Ladder* é, ainda hoje, uma das linguagens mais utilizadas por engenheiros, tecnólogos e técnicos da área de automação. Ainda segundo Silvestre (2010), o nome diagrama de lógica *Ladder* deve-se à sua representação gráfica semelhante à de uma escada, constituída por segmentos de reta paralelos entre si e conectados através de linhas horizontais, denominadas *rungs*, que representam a lógica de controle do LLD.

Portanto, uma vez apresentados os conceitos preliminares pertinentes ao contexto deste trabalho, e estabelecidas as correlações de importância entre tais para o desenvolvimento um algoritmo de conversão de RPC para LLD, far-se-á na secção seguinte a formalização do método de conversão aqui proposto com o objetivo de reduzir a distância entre RPC e LLD, conforme citado anteriormente.

3 ALGORITMO DE CONVERSÃO DE REDES DE PETRI COLORIDAS PARA LADDER LOGIC DIAGRAM (LLD)

Neste capítulo é apresentada a proposta de conversão de modelos de Redes de Petri Coloridas (RPC) para o *Ladder Logic Diagram (LLD)*. Para tanto, são realizadas as definições que formalizam a estrutura de correlação entre os trechos de RPC para LLD. Sendo, portanto, apresentadas as definições de: lugares não marcados, seguido de lugares marcados, transições e condições de disparo de uma transição.

Para a modelagem das RPCs fez-se uso de um *software* de simulação computacional gratuito – CPN Tools, o qual será resumidamente apresentado aqui. Para explorar a metodologia de conversão de RPC para LLD, foi desenvolvido um aplicativo em JAVA, capaz de importar os dados do código fonte de CPN Tools e gerar, numa tela em JAVA, a programação correspondente em LLD.

3.1 Definições e correlação entre RPC e LLD

Numa RPC estão presentes como elementos de base: lugares, transições, arcos e cores. Num SED controlado por CLP e programado em LLD estão exatamente presentes os blocos correspondentes a tais elementos respeitando a dinâmica da rede, e, portanto, permitindo a conversão das partes para o todo.

3.1.1 Conversão de lugares

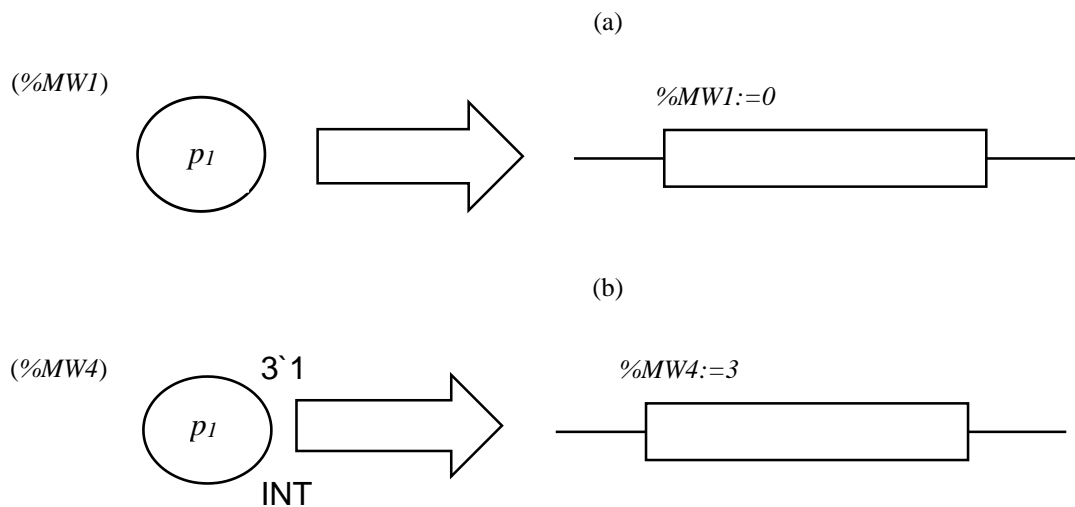
Seja a representação do sistema em LLD pelo método intuitivo clássico. Dela é possível inferir uma relação entre lugares, transições e fichas com os elementos característicos da LLD tais que:

DEFINIÇÃO 3.1 Um lugar p_i com n fichas de uma mesma cor ($n \geq 0$) é convertido para LLD como uma memória com valor igual a n .

Na Figura 3.1(a) é apresentada a conversão do lugar p_1 , não marcado, numa memória de LLD cujo valor é zero (por exemplo $\%MW1=0$).

Na Figura 3.1(b) é apresentada a conversão do lugar p_1 , cuja marcação é $M_{MS}(p_1) = 3$, numa memória de LLD cujo valor é três (por exemplo $\%MW4=3$).

Figura 3.1 - Conversão de lugar numa memória de CLP



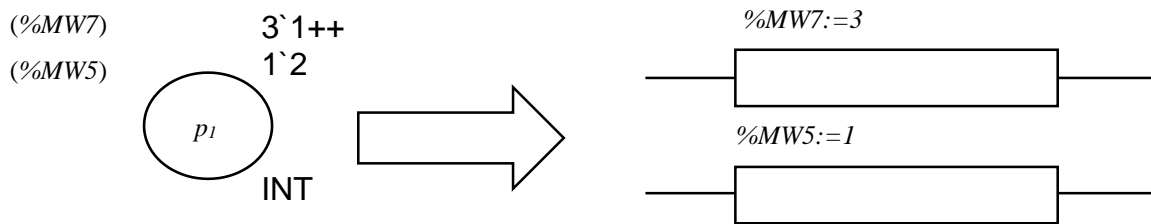
Fonte: (Autor, 2013)

O índice k da memória usada na conversão de uma cor num lugar ($\%MWk:=3$) é alocado modo aleatório durante a conversão; portanto, poderá ser usado qualquer endereço de memória disponível no CLP. Ressaltando-se que uma vez correlacionados (memória e cor de um lugar) deve-se manter esta relação específica entre ambos.

DEFINIÇÃO 3.2 Um lugar p_i **marcado** com fichas de n cores diferentes ($n \geq 1$) é convertido para LLD como n memórias cujos respectivos valores são iguais ao número de fichas daquela p_i

Na Figura 3.2 é apresentada a conversão do lugar p_1 , contendo dois valores diferentes de fichas, em duas memórias de CLP em LLD ($M_{MS}(p_1) = 3^1 \equiv \%MW1 = 3$) e ($M_{MS}(p_1) = 1^2 \equiv \%MW2 = 1$).

Figura 3.2 - Conversão de lugar marcado numa memória de CLP



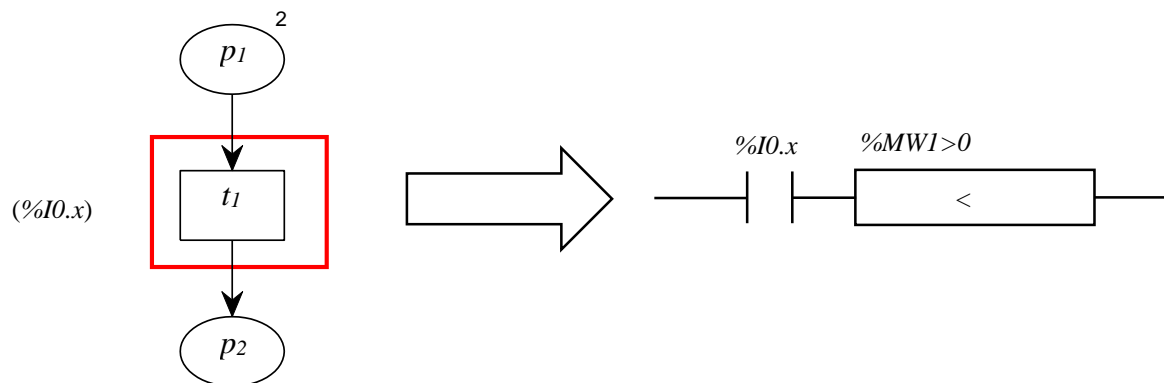
Fonte: (Autor, 2013)

3.1.2 Conversão de transições

DEFINIÇÃO 3.3 Uma transição t_i é convertida para LLD como um bloco condicionante da ação de **decremento** das fichas dos lugares de entrada de p_i e **incremento** das fichas dos lugares de saída de p_i .

Na Figura 3.3 é apresentada a conversão da transição t_i no bloco condicionante em questão. Nesta figura, $\%MWI > 0$ significa o peso do arco relativo a cada cor de ficha de cada lugar de entrada da transição, o que representa a condição básica de habilitação de uma transição em RPC.

Figura 3.3 - Conversão de transição com um único arco de entrada

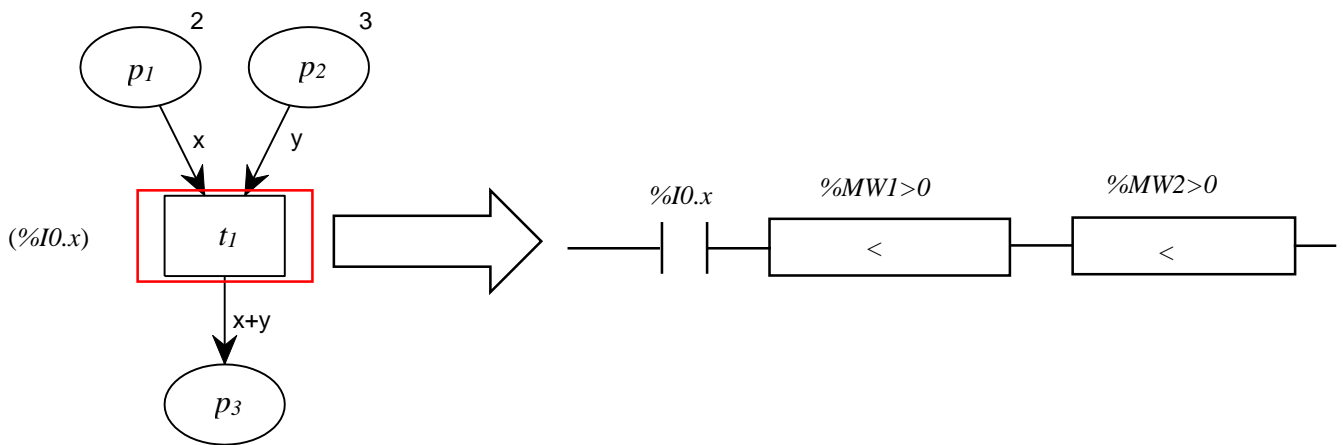


Fonte: (Autor, 2013)

DEFINIÇÃO 3.4 Uma transição t_i com mais de um lugar de entrada (sincronismo) é convertida para LLD como um bloco condicionante da ação de **decremento** das fichas dos lugares de entrada de p_i e **incremento** das fichas dos lugares de saída de p_i .

Na Figura 3.4 é apresentada a conversão da transição t_1 . Perceba que esta configuração representa um sincronismo de recursos, e portanto, a transição somente estará habilitada se forem satisfeitas as condições dos arcos (quantidade mínima de fichas em cada lugar de entrada). Neste caso, a conversão para LLD faz uso do conceito da lógica “and” para representar fielmente o sincronismo modelado na RPC.

Figura 3.4 - Conversão de transição com mais de um arco de entrada

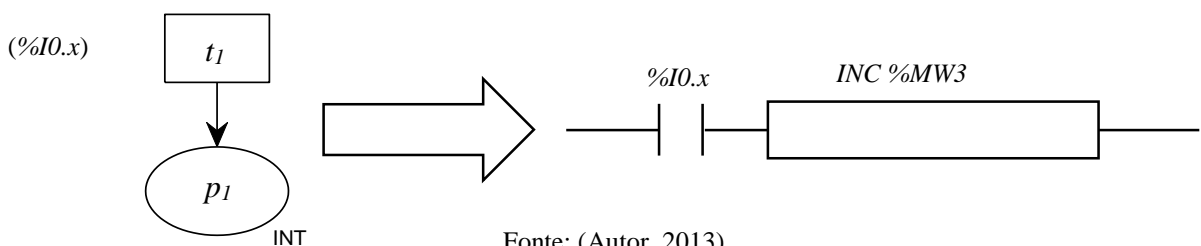


Fonte: (Autor, 2013)

DEFINIÇÃO 3.5 Uma transição t_i na qual **não haja lugares de entrada** e haja lugares de saída (transição fonte) é convertida para LLD como um bloco de **incremento** das fichas dos lugares de saída de p_i .

Na Figura 3.5 é apresentada a conversão da transição t_1 . Esta configuração representa uma transição fonte e estará habilitada sempre que ocorrer um evento externo representado por $\%IO.x$. Neste caso, a conversão para LLD faz uso do conceito da lógica “and”.

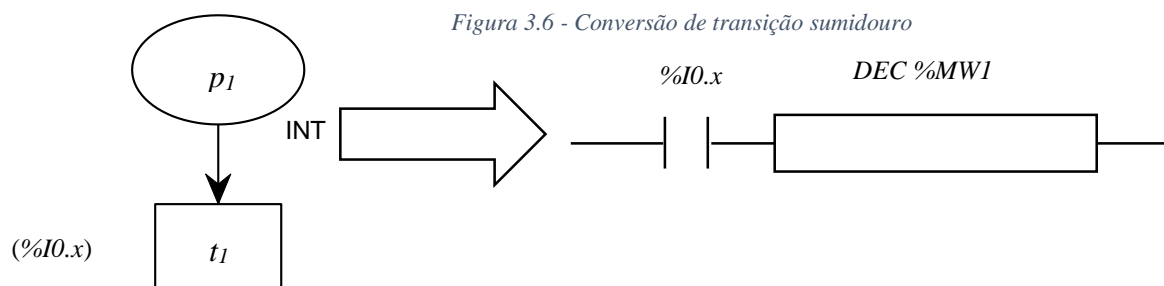
Figura 3.5 - Conversão de transição fonte



Fonte: (Autor, 2013)

DEFINIÇÃO 3.6 Uma transição t_i na qual **não haja lugares de saída** e haja lugares de entrada (transição sorvedouro) é convertida para LLD como um bloco de **decremento** das fichas dos lugares de entrada de p_i .

Na Figura 3.6 é apresentada a conversão da transição t_1 . Esta configuração representa uma transição fonte e estará habilitada sempre que ocorrer um evento externo representado por $\%I0.x$. Neste caso, a conversão para LLD faz uso do conceito da lógica “and”.

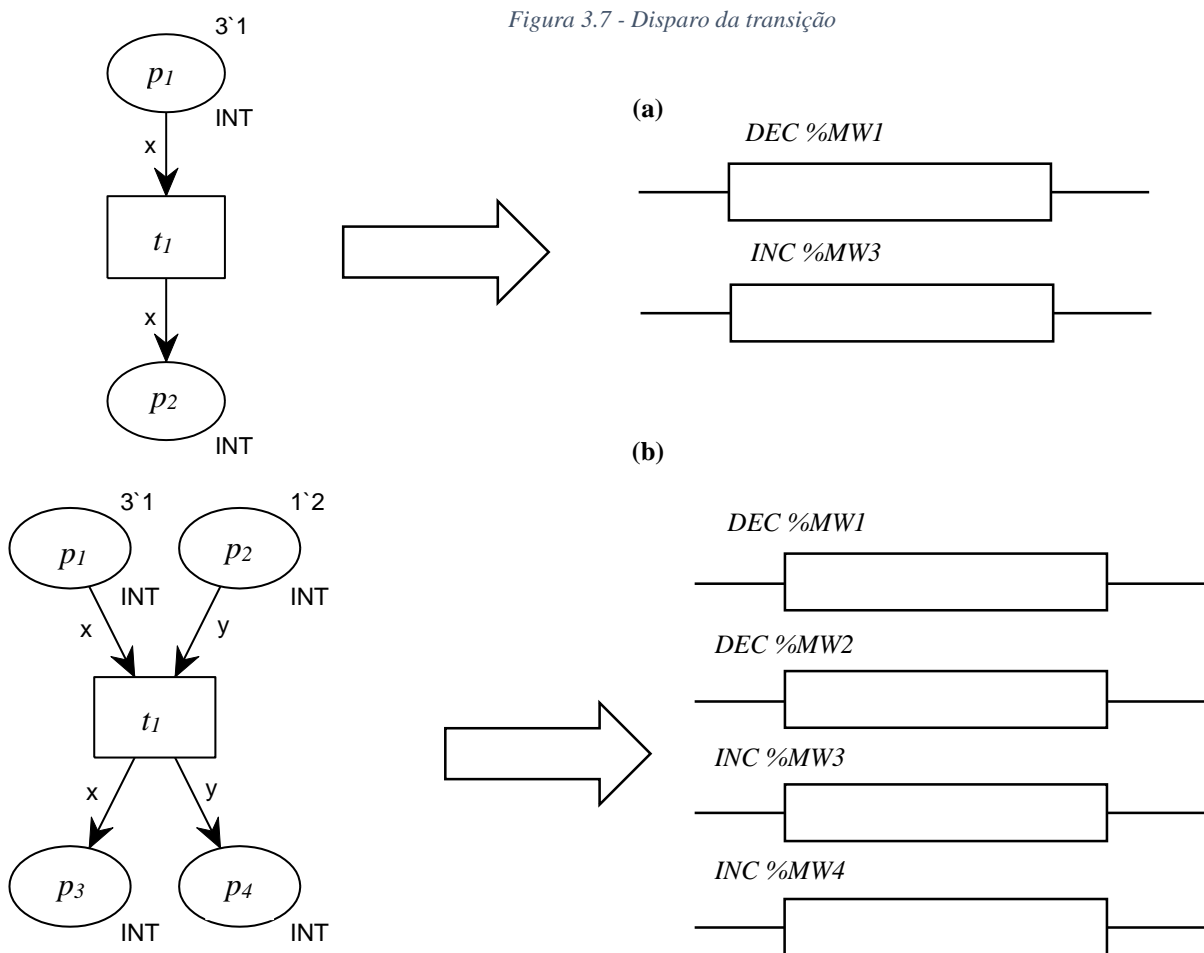


Fonte: (Autor, 2013)

3.1.3 Disparo de uma transição

DEFINIÇÃO 3.7 O disparo de uma transição t , com n lugares de entrada ($n \geq 1$) e m lugares de saída ($m \geq 1$), é convertido como o decremento e incremento das memórias correspondentes às fichas dos lugares de entrada e dos lugares de saída.

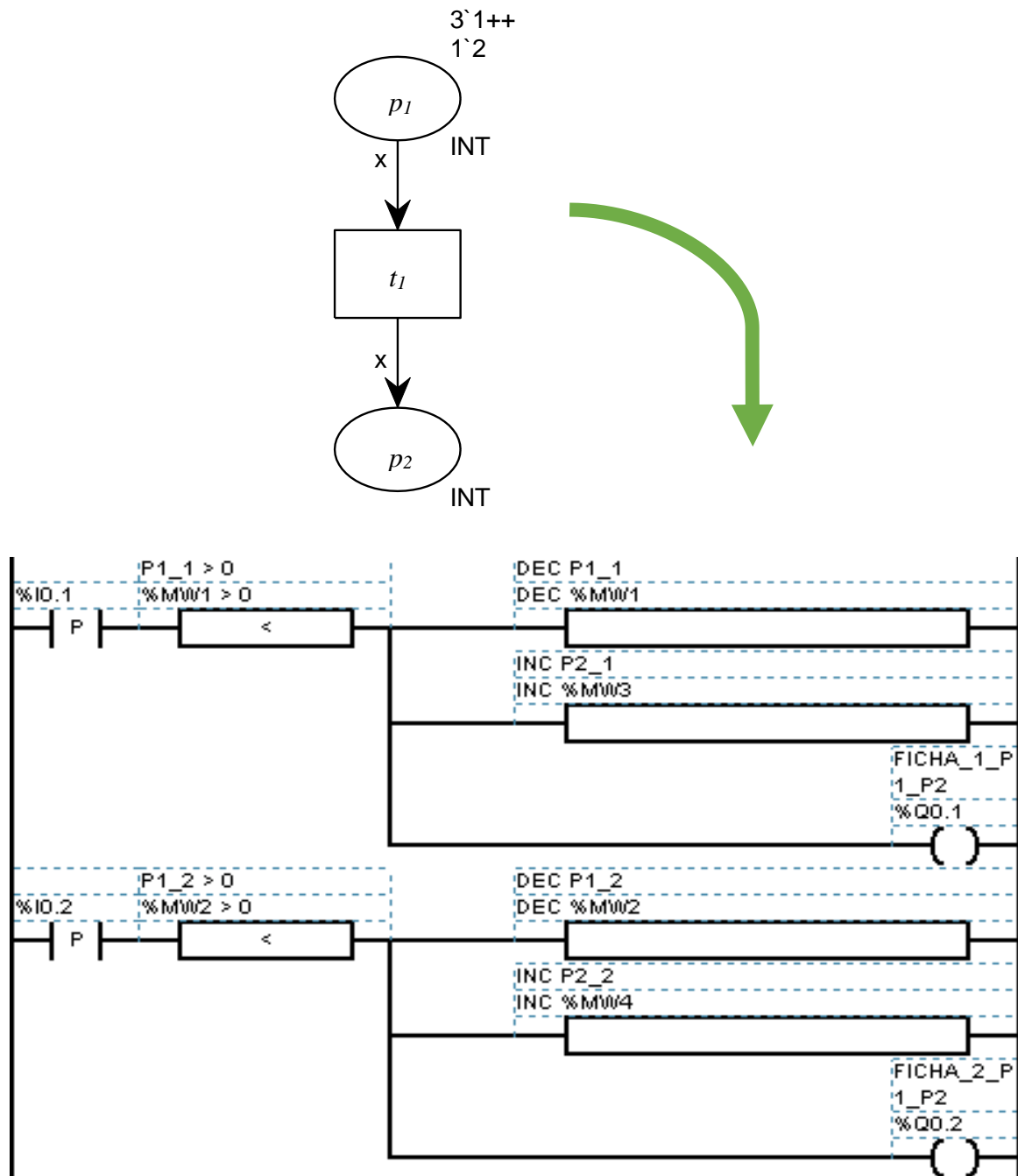
Na Figura 3.7(a) é apresentada a conversão do disparo da transição t_1 com um lugar de entrada e um lugar de saída. Perceba que o disparo da transição altera o valor das memórias de dados relativas a p_1 ($DEC \%MW1$) e p_2 ($INC \%MW3$). Na Figura 3.7(b) é apresentada a conversão do disparo da transição t_1 com dois lugares de entrada e dois lugares de saída. Perceba que o disparo da transição altera o valor das memórias de dados relativas a p_1 e p_2 ($DEC \%MW1$ e $DEC \%MW2$) e p_3 e p_4 ($INC \%MW3$ e $INC \%MW4$).



3.2 Conversão de uma RPC para LLD

A conversão completa de uma RPC para LLD, agora concatenados os passos supracitados é apresentada na Figura 3.8.

Figura 3.8 - Conversão completa RPC para LLD



Fonte: (Autor, 2013)

Adicionalmente aos trechos correspondentes de RPC para LLD, cada linha do LLD traz consigo a condição de **evento externo**, o qual ocorre no processo. Deste modo a condição de **aleatoriedade** de disparo de transições, presente nas RPCs, é mantida.

Para cumprir esta condição, as entradas do CLP são endereçadas com *%I0.x* (veja Figura 3.8) e conectadas diretamente (em série) aos trechos da transição como pré-condição. Elas representam um evento **externo** (sensor ligado ao processo) que poderá ou não solicitar o disparo de uma dada transição e de forma aleatória. De igual modo, as saídas do CLP são endereçadas com *%Q0.x* (veja Figura 3.8) e conectadas em paralelo aos trechos da transição como pós-condição. Elas representam um evento externo (atuador ligado ao processo) que somente executará trabalho quando ocorrer o disparo de uma dada transição, correspondente ao mesmo, e satisfeita a condição de teste (marcação do lugar maior que zero).

3.3 Automatização da conversão

Antes de apresentar a conversão automatizada de um modelo RPC para LLD, é necessário apresentar o arquivo de representação de RPC usado no CPN Tools, bem como a ferramenta Eclipse, usada para produzir o conversor na linguagem JAVA.

3.3.1 RPC no CPN Tools

A ferramenta computacional CPN Tools, disponível gratuitamente em <http://www.cpn-tools.org/>, é utilizada para editar, simular e analisar RPCs. Dentre as características desta têm destaque: a verificação de sintaxe e geração de código, que acontecem enquanto a rede está sendo construída. E que, para tanto, usa uma linguagem específica para gerar a parte gráfica das modelagens: o XML - *eXtensible Markup Language*.

Nesse sentido, torna-se apreciável o fato de construir uma RPC no CPN Tools e, explorar sua linguagem para gerar um conversor no qual a saída desejada seja o LLD da RPC modelada.

3.3.2 *Uso da ferramenta Eclipse*

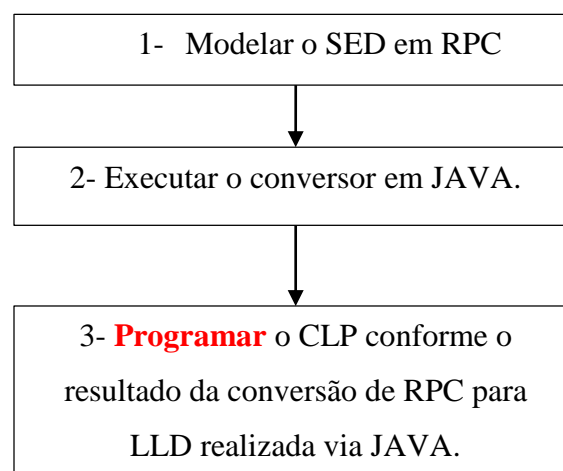
Dentre as ferramentas de manipulação de banco de dados para desenvolvedores de plataformas em JAVA, destaca-se o *Eclipse Classic*. Destinado essencialmente à programação em JAVA, torna-se, com o incrementos de *plug-ins*, uma ferramenta dedicada a outras linguagens de programação, como C/C++, PHP, XML e J2EE. A grande quantidade de extensões, a boa documentação, tanto para usuários como para programadores, e os diversos pacotes de tradução aumentam a lista de vantagens do Eclipse, fazendo ainda parte de um projeto de código inteiramente aberto.

3.3.3 *Algoritmo de conversão automática de modelo RPC para LLD*

Uma vez estruturada a conversão, é possível o uso de ferramentas computacionais que permitam ao programador desenvolver sua RPC num ambiente virtual de simulação, análise e síntese. Para tanto destaca-se o CPN Tools como aplicativo adequado para modelar uma RPC e, a posteriori, implementar a rede num CLP.

Na Figura 3.9 é apresentada a sequência que define os passos da conversão automatizada de RPC para LLD, explorada neste trabalho.

Figura 3.9 - Passos de conversão de RPC para LLD



Fonte: (Autor, 2013)

1- Modelar o SED em RPC – a modelagem de um SED em RPC no CPN Tools deve ocorrer segundo as seguintes especificações de declarações:

- Conter somente conjunto de cores (tipo) inteiro – INT;
- Conter no conjunto INT os valores de fichas definidos pelo intervalo 1 .. k . Em que k é o valor maior a ser usado naquele conjunto. Exemplo: (*colset FICHAS = int with 1..3* – define três cores (valores) diferentes para o conjunto de cores *FICHAS*);
- Conter variáveis associadas aos arcos referentes aos conjuntos de cores definidos.

2- Executar o conversor – ao executar o conversor *cpnToLadder.jar*. o mesmo, irá pedir o nome do arquivo .cpn (XML). Após isso o mesmo realiza a conversão para LLD.

3- Programar o CLP – dado o arquivo LLD, a programação do CLP é tal que pode ser implementada em qualquer modelo de CLP a partir do arquivo LLD produzido pelo conversor. Destaca-se que no conversor as entradas (saídas) do CLP são endereçadas a partir de *%I0.1* (*%Q0.1*) e incrementadas da unidade sempre que houver uma nova condição de disparo de transição. Estas entradas (saídas) representam os eventos externos do SED que diparam as transições relativas ao comportamento deste.

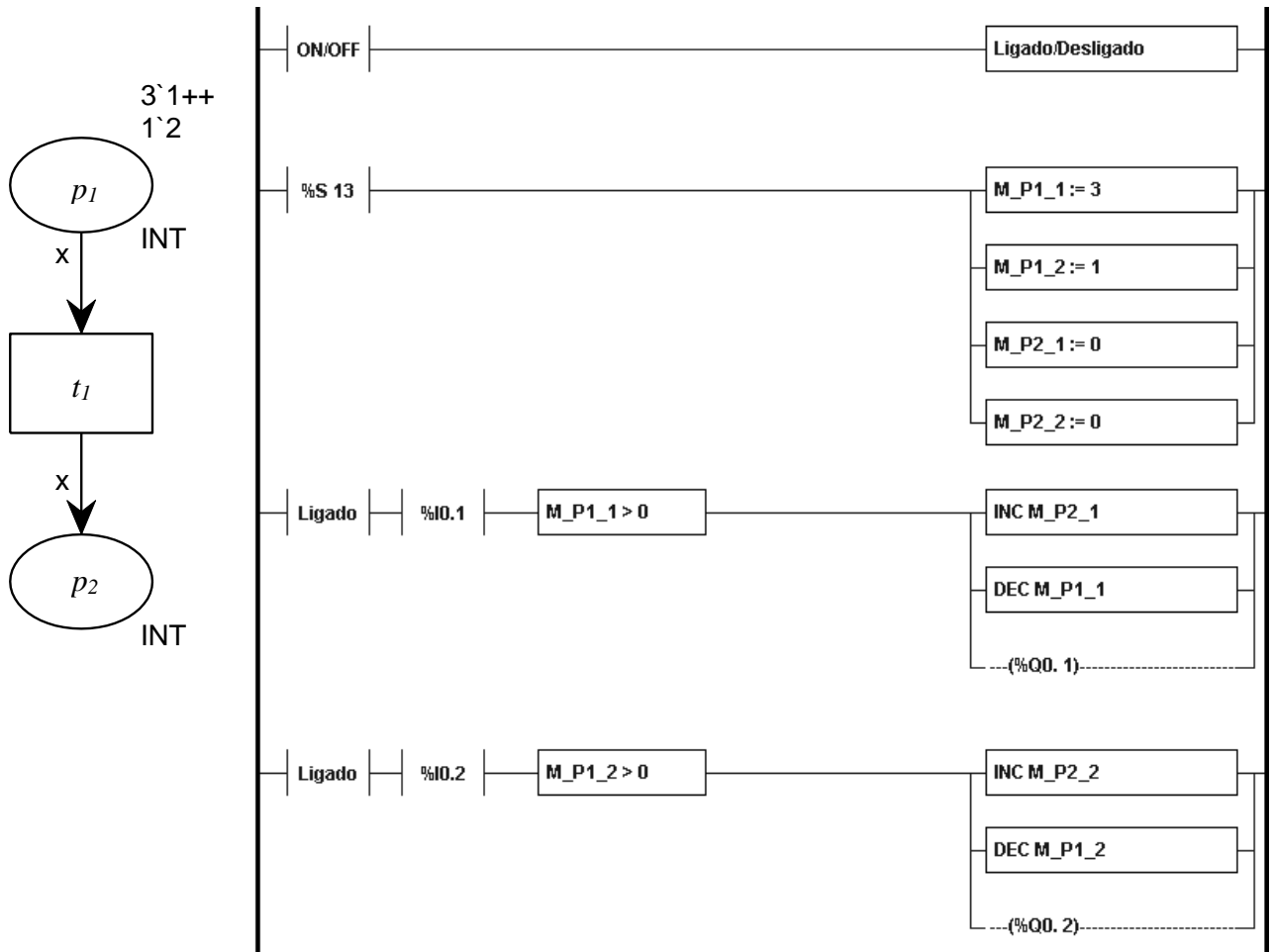
O Apêndice A contém as etapas da execução do conversor com imagens detalhas desses.

3.3.4 Aplicação em JAVA

Para esta dissertação, desenvolveu-se um aplicativo em JAVA capaz de converter RPC em LLD. No anexo A é apresentado o código de importação de dados em *XML*, e exportação para ambiente JAVA na estrutura LLD.

Na Figura 3.10 é apresentada uma RPC modelada no CPN Tools e convertida **automaticamente** para LLD em ambiente JAVA. Enquanto que na Figura 3.8 apresentou-se a mesma rede, convertida **manualmente** para LLD através de um *software* proprietário do fabricante Schneider Electric – o *TWIDO Suite*.

Figura 3.10 - Conversão automática de RPC para LLD em Java



Fonte: (Autor, 2013)

Na Figura 3.10, “*ON/OFF*” representa um interruptor qualquer de campo destinado ao início de operação do sistema (*start*). “*Ligado/Desligado*” representa uma memória tipo bit, reservada para informar se o interruptor continua na referida posição ligado ou não. Esta é usada, no programa em LLD, em série às entradas (*%IO.x*) condicionando todas as linhas de programa a este estado.

%S13 significa a ação de reservar as memórias de CLP para os lugares correspondentes à RPC, e carregar o valor respectivo das fichas em cada memória de dados, exatamente no primeiro ciclo iniciação do CLP.

%IO.1 e *%IO.2* são entradas (tipo pulso) do CLP que representam o endereço dos sensores de campo acoplados ao processo para informar um estado ou mudança, do sistema

real. Enquanto, %Q0.1 e %Q0.2 são as saídas (tipo pulso) do CLP que representam os atuadores de campo.

No **Apêndice B** são apresentadas as relações de algumas RPCs convertidas para LLD; as quais foram produzidas **comparativamente** pelo conversor em JAVA, e pelo *software Twido Suite* do fabricante Schneider.

Destaca-se o elevado grau de correspondência entre ambos, bem como a facilidade de conversão. Desta forma o programador pode desenvolver sua RPC facilmente no CPN Tools e, ao terminar sua análise, convertê-la automaticamente para LLD. A partir desse momento, dispõe-se da mesma RPC agora capaz de ser implementada em qualquer CLP fazendo uso de suas interfaces gráficas para programadores, que são particulares de cada fabricante no que se refere aos arranjos de *design* gráfico; todavia similares, por efeito normativo.

Nesse contexto, uma vez apresentadas as definições que formalizam o algoritmo de conversão de RPC para LLD, a ferramenta utilizada para a modelagem das RPC (CPN Tools) e a ferramenta para produzir-se o conversor automatizado em JAVA (Eclipse), far-se-á na secção seguinte, a aplicação do conversor numa célula de manufatura industrial (FMS) como estudo de caso para validá-la.

4 ESTUDO DE CASO: CONVERSÃO DE UM MODELO DE FMS ROBOTIZADA

Neste capítulo é apresentada a proposta de conversão de modelos de células de manufatura (FMS) em Redes de Petri Coloridas (RPC) para o *Ladder Logic Diagram* (LLD). Para tanto, utiliza-se a metodologia de controle supervisorio para RPC denominada Restrições de Controle sobre Cores Decompostas (RCCD), enfatiza-se a transformação de RPC não controlada para LLD e a facilidade de inclusão do controle. E, para exemplificar a metodologia, foi utilizado o conversor de RPC para LLD, produzido em ambiente JAVA, a fim de evidenciar a dinâmica de conversão.

Para a validação dos resultados do conversor é usado um robô articulado vertical controlado via CLP, que opera comparativamente com e sem supervisão e que fora modelado, primeiramente em RPC sem supervisão, depois aplicada a técnica do RCCD, supervisão e finalmente a conversão para LLD.

4.1 Contexto industrial

A programação de um robô numa célula de manufatura pode ser simplificada pela inserção de um CLP, o qual, tratando as informações de uma rede, envia ao robô somente os pulsos de ordem de operação. Desta forma, ganha-se na redução da complexidade da programação do robô, uma vez que este será, para o sistema, um simples atuador (*slave*) do Sistema de Manufatura Flexível (FMS).

Portanto, usando-se a TCS para a síntese de um supervisor em FMS, define-se a seguinte sequência:

1. Modelar o FMS em RPC sem controle;
2. Implementar um algoritmo de trabalho no FMS via CLP, baseado no modelo RPC;
3. Baseado nas restrições, aplicar o RCCD para encontrar os supervisores;
4. Acrescentar os lugares de controle ao modelo RPC;
5. Realizar a fusão dos supervisores;
6. Modelar a RPC do FMS com a fusão de supervisores conforme passo anterior;

7. Alterar o algoritmo de trabalho no FMS via CLP para implementar o controle;
8. Programar o robô como *slave* do CLP.

A ação acima visa alcançar o contexto industrial onde há algoritmos de controle já implementados nos CLPs do FMS. Desta forma, explora-se o baixo esforço de alteração na programação do CLP para implementar o controle supervisorio por RCCD.

4.2 Descrição de célula de manufatura flexível (FMS)

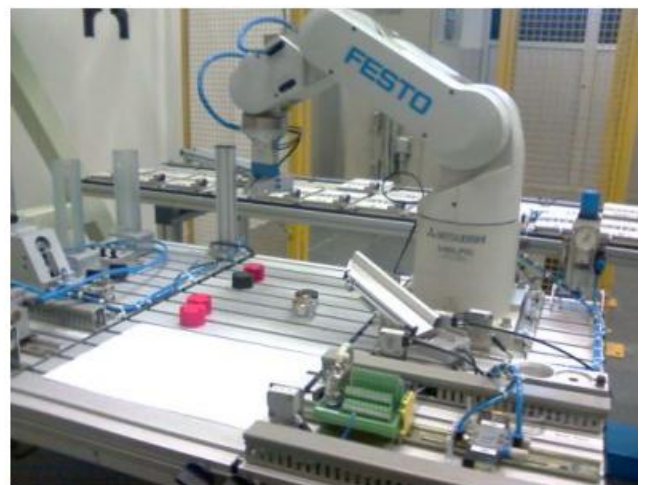
Seja a célula de manufatura apresentada na Figura 4.1.

Figura 4.1 - Célula de manufatura robotizada

(a) FMS industrial



(b) FMS didático

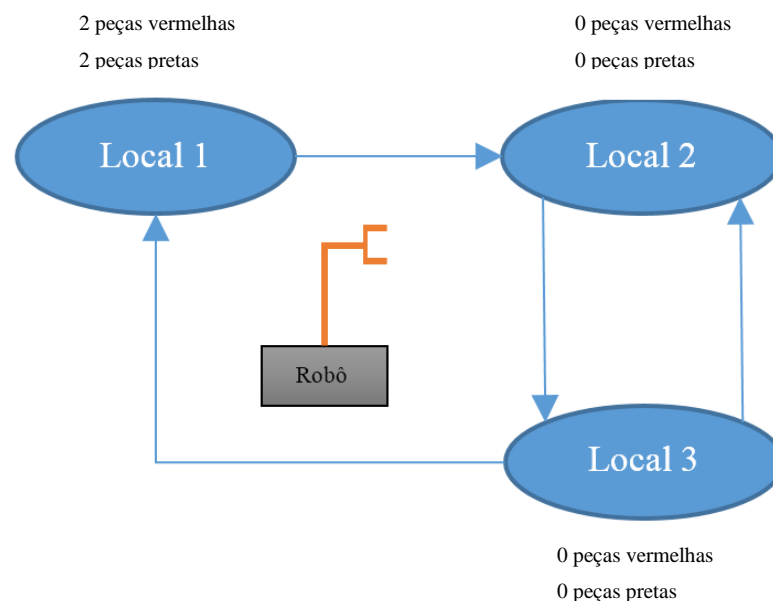


Fonte: (Autor, 2013)

O modelo do FMS apresentado na Figura 4.1, é apresentado na Figura 4.2.

Nesta figura, um robô articulado vertical de seis graus de liberdade precisa posicionar peças de diferentes cores entre 3 locais distintos, e com um *loop* de reprocesso em caso de falha de controle de qualidade; o qual é representado pela seta que liga o local 3 ao local 2.

Figura 4.2 - Célula de manufatura robotizada - diagrama



Fonte: (Autor, 2013)

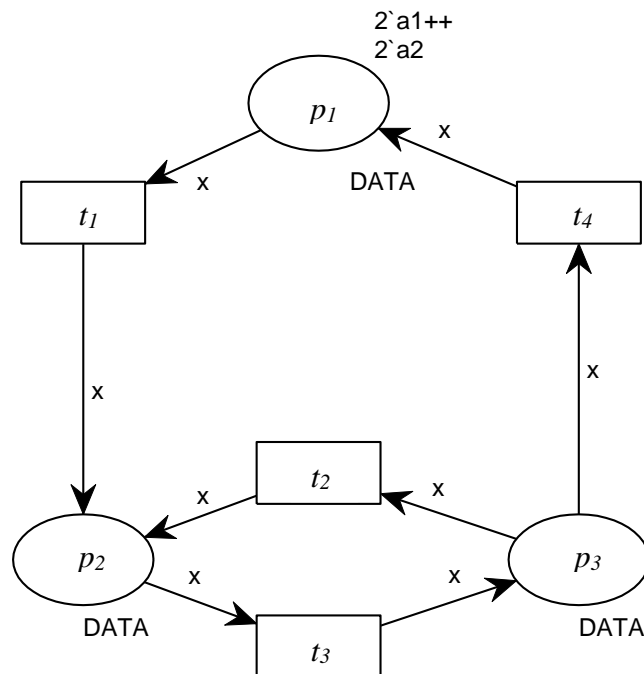
Da forma em que se encontra o sistema, o robô pode pegar peças **aleatoriamente**, e sem controle da quantidade, entre quaisquer locais. Ocorre que a capacidade real de processamento e de reprocessamento nos locais 2 e 3 é limitada a uma única peça de cada cor. Ou seja, os lugares somente podem receber duas peças para processar; sendo uma preta (a_1) e outra vermelha (a_2). Todavia, o robô na célula continua a mandar as peças entre os locais se houver disponibilidade destas no local de origem.

Esta situação caracteriza a ausência de um controle e, portanto, torna oportuna a aplicação do RCCD com as restrições de número de peças nos lugares supracitados.

4.3 Modelagem do FMS sem controle

Na rede da Figura 4.3 é apresentado o comportamento não controlado do robô. Observe que enquanto houver fichas em p_1 (peças disponíveis para a execução de operações fabris), a transição t_1 estará habilitada e, assim, pode-se colocar até n fichas no lugar p_2 . Mas, se p_2 for um lugar limitado, é necessário exercer controle para limitar o número de fichas (peças) nesse lugar. O mesmo ocorre com p_3 .

Figura 4.3 - Modelagem do FMS em RPC



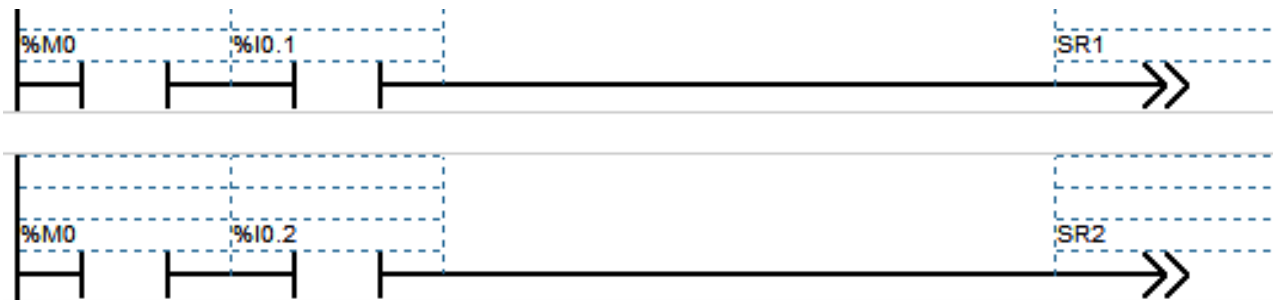
Fonte: (Autor, 2013)

4.4 Implementação do programa sem controle em CLP

A primeira ação é desenvolver um programa de CLP, baseado no modelo apresentado na Figura 4.3, sem controle. Nesta ação, a programação de CLP usou oito linhas de programa, onde as respectivas sub-rotinas são chamadas por pulsos que sensibilizam entradas de interesse. As figuras 4.4, 4.5 e 4.6, mostram **trechos** do programa de CLP, em LLD, referentes a esse processo. As entradas ($\%IO.x$) representam interruptores ou sensores de proximidade usados para chamar as rotinas específicas do robô, o que ocorre no barramento de saída do CLP ($\%QO.x$).

Na Figura 4.4 são mostradas duas rotinas de varredura, realizadas quando um sensor de proximidade ativa uma das entradas (%I0.1 e %I0.2) do CLP.

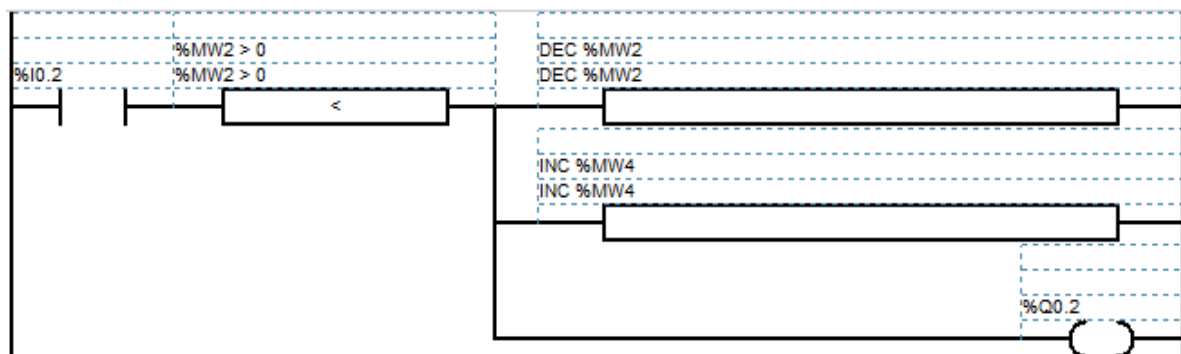
Figura 4.4 - Linha de programa do CLP chamando respectiva sub-rotina



Fonte: (Autor, 2013)

Na Figura 4.5 é apresentada a ação executada quando os blocos de memórias “%MW2” e “%MW4” recebem respectivamente decremento e incremento representando uma ficha que sai do lugar p_1 para o lugar p_2 , conforme RPC modelada na Figura 4.3, **iniciando** o processo.

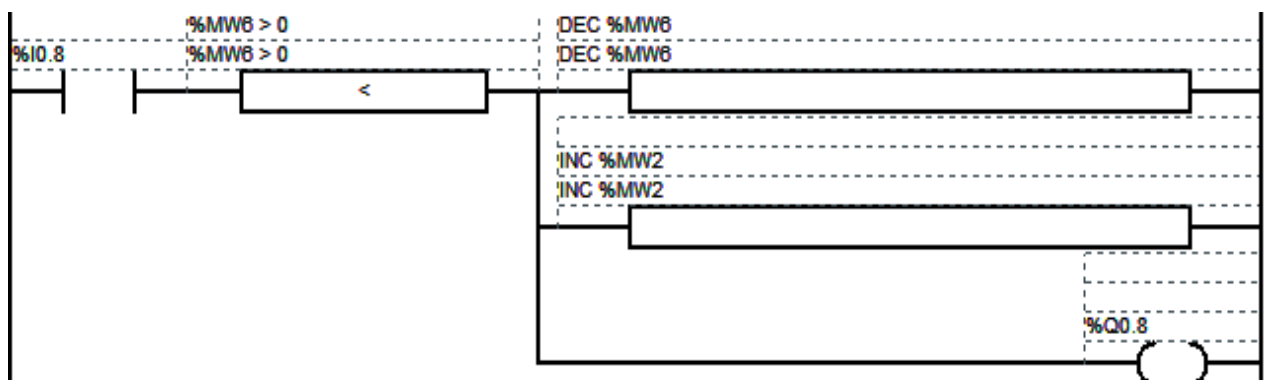
Figura 4.5 - sub-rotina do CLP condição de saída de p_1



Fonte: (Autor, 2013)

Na Figura 4.6 é apresentada a ação executada quando os blocos de memórias “%MW6” e “%MW2” recebem, respectivamente, decremento e incremento representado por uma ficha que sai do lugar p_3 para p_1 , conforme RPC modelada na Figura 4.3, **reiniciando** o processo.

Figura 4.6 - Programa de CLP condição de chegada em p_1



Fonte: (Autor, 2013)

4.5 Aplicação de RCCD

Seja a célula manufatura apresentada na Figura 4.1. Seja a Figura 4.3 a modelagem desta em RPC. Observa-se que esta possui apenas um conjunto de cores (DATA) relativo a todos os lugares da rede, e que a todos os arcos é associada a variável x . Nesse caso, de variável única, $D = D_x$ e $C = C_x$.

Impondo as especificações de restrições descritas no subsecção 4.2 deste trabalho, $M(p_2) \leq 1 \cdot "a_1"$ e $M(p_3) \leq 1 \cdot "a_2"$ tem-se da RPC não controlada a matriz de incidência relativa à variável x que:

$$D^x = \begin{pmatrix} -1(x) & 0 & 0 & 1(x) \\ 1(x) & 1(x) & -1(x) & 0 \\ 0 & -1(x) & 1(x) & -1(x) \end{pmatrix} \quad (4-1)$$

o que implica na matriz de referência de incidência, da RPC não controlada, relativa à variável x .

$$C^x = \begin{pmatrix} -1 & 0 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & -1 \end{pmatrix} \quad (4-2)$$

A matriz \mathcal{L} indica os lugares que sofrem as restrições.

$$\mathcal{L} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (4-3)$$

Assim, a matriz C_c , do supervisor é:

$$C_c = -\mathcal{L}C = \begin{pmatrix} -1 & -1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{pmatrix} \quad (4-4)$$

mostrando que o supervisor possui dois lugares de controle, pc_1 e pc_2 .

A primeira linha da matriz refere-se à restrição

$$M(p_2) \leq (1 \ a_1) \quad (4-5)$$

e diz que o lugar de controle pc_1 é entrada para as transições t_1 e t_2 e saída para t_3 .

A segunda linha refere-se à restrição

$$M(p_3) \leq (1 \ a_2) \quad (4-6)$$

e diz que o lugar de controle pc_2 é entrada de t_3 e saída de t_2 e t_4 .

A seguir, as marcações iniciais dos lugares de controle pc_1 e pc_2 são adquiridas na matriz B , sabendo-se que esta apresenta a soma das restrições com as respectivas fichas complementares de fluxo é:

$$B = \begin{pmatrix} 1 \ a_1 + 2 \ a_2 \\ 1 \ a_2 + 2 \ a_1 \end{pmatrix} \quad (4-7)$$

A marcação inicial da RPC não controlada é:

$$M_0 = \begin{pmatrix} 2 \ a_1 + 2 \ a_2 \\ 0 \\ 0 \end{pmatrix} \quad (4-8)$$

Dessa forma,

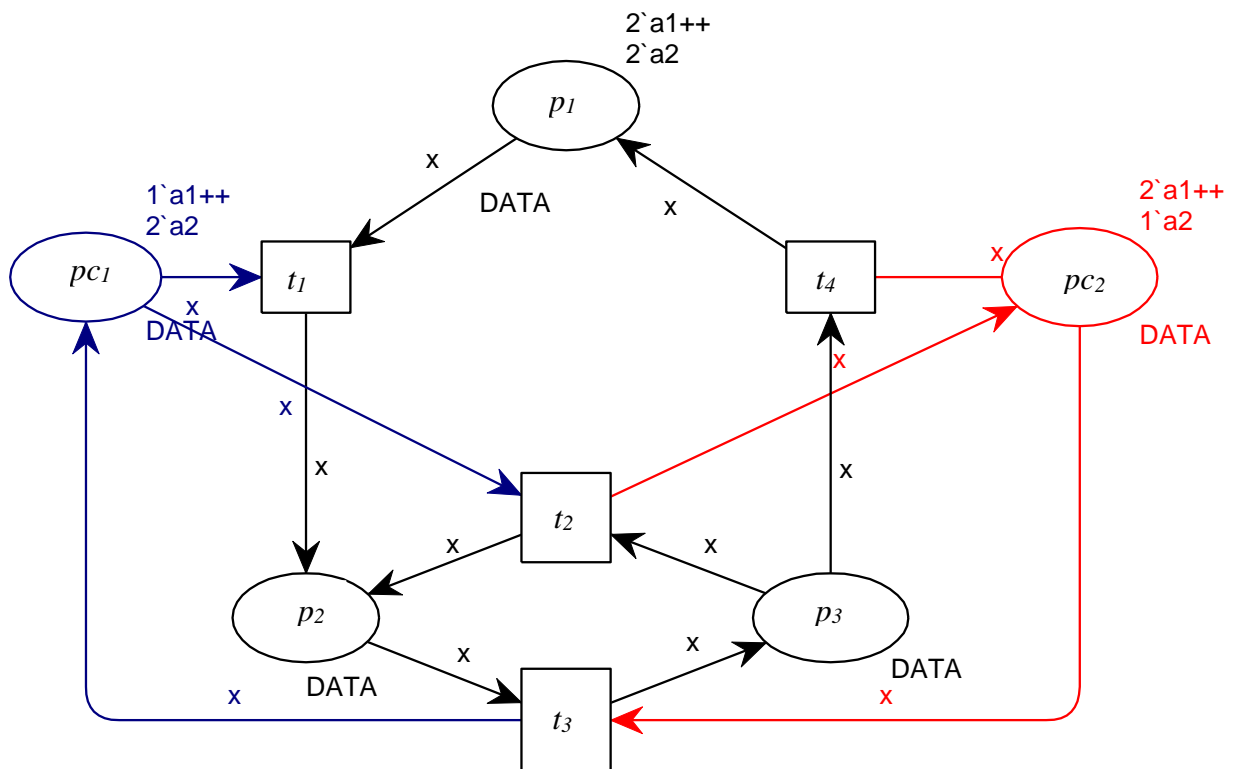
$$M_{c_0} = \begin{pmatrix} 1 \ a_1 + 2 \ a_2 \\ 1 \ a_2 + 2 \ a_1 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \ a_1 + 2 \ a_2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \ a_1 + 2 \ a_2 \\ 1 \ a_2 + 2 \ a_1 \end{pmatrix} \quad (4-9)$$

As linhas da matriz M_{c0} são as marcações iniciais dos lugares de controle pc_1 e pc_2 .

4.6 Modelagem do FMS com supervisores

O uso da técnica de RCCD permite transformar a rede da Figura 4.3, sem controle, numa rede controlada bem definida conforme Figura 4.7. O controle consiste em restringir a quantidade de peças nos lugares p_2 e p_3 de tal forma que nunca haverá mais de uma peça de valor a_1 (preta) no lugar p_3 e nunca haverá mais de uma peça de valor a_2 (vermelha) no lugar p_2 .

Figura 4.7 - RPC com uso de RCCD



Fonte: (Autor, 2013)

4.7 A fusão dos controladores da RPC

Observando a Figura 4.7 percebe-se que os controladores pc_1 e pc_2 satisfazem às condições da subsecção 4.2. Utilizando o método de fusão dos controladores, proposto em

Menezes, Barroso e Prata (2012), encontra-se um único lugar de controle em substituição aos lugares pc_1 e pc_2 . Assim, a matriz de incidência é:

$$\mathcal{H} = (-1 \ 0 \ 0 \ 1) \quad (4-10)$$

A marcação inicial da fusão é

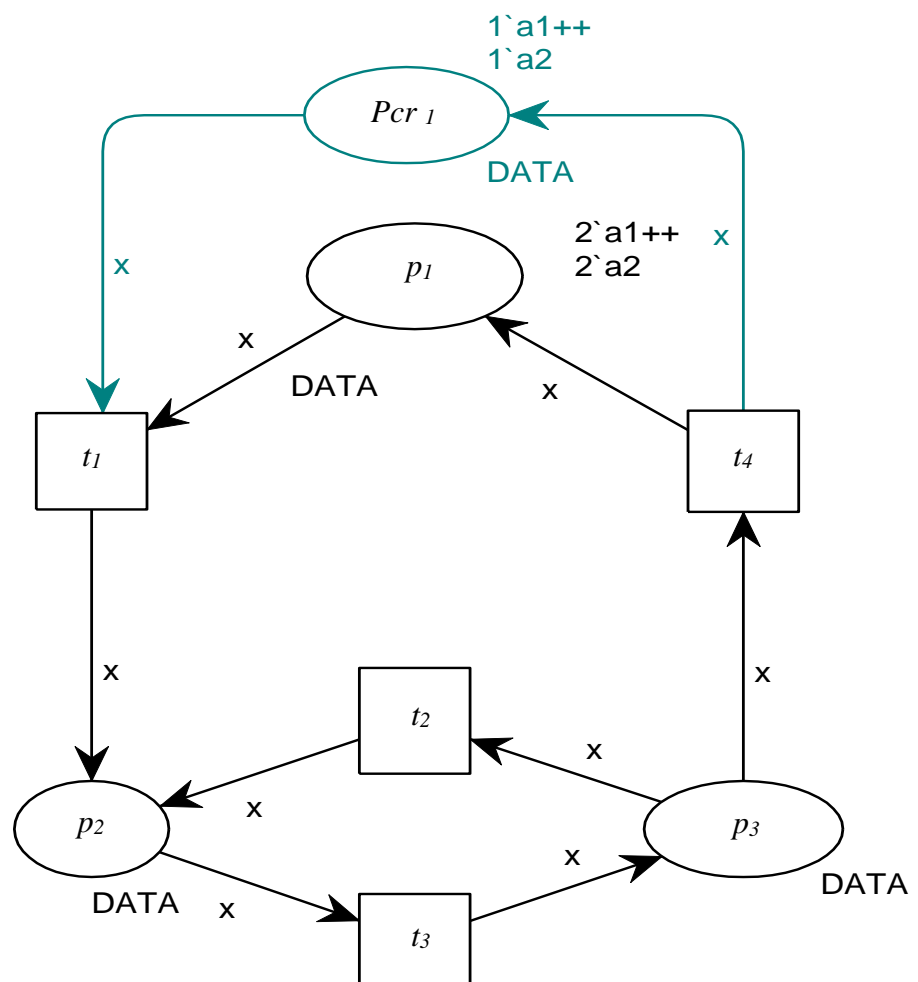
$$M_0 = (1''a_1'' + 1''a_2'') \quad (4-11)$$

O supervisor resultante da fusão (Pcr_1), nesse caso, passa a exercer o mesmo controle que os controladores pc_1 e pc_2 .

4.8 Modelagem do controle supervisorio após a fusão de controladores

Observando a rede da Figura 4.6, e fazendo-se uso da técnica de fusão de controladores apresentado subsecção 4.7 deste trabalho, obtém-se uma rede controlada com um modelo mais simplificado, o qual por esta razão, encontra elevada atratividade industrial. Na Figura 4.8 é apresentado o resultado desta rede após a fusão dos controladores com a manutenção das restrições impostas na Figura 4.7.

Figura 4.8 - RPC com fusionamento de controladores



Fonte: (Autor, 2013)

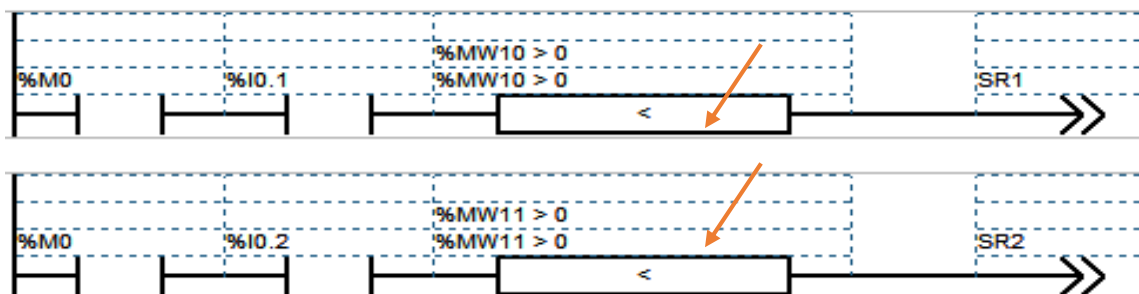
4.9 Implementação de controle no programa de CLP

Uma vez concluídas as etapas anteriores, o projetista pode gerar um novo programa em linguagem Ladder de forma automática, ou simplesmente incluir o controle em um programa Ladder já existente. Esta inclusão é feita através da implementação de blocos de memória de dados (tipo *word* - 16 bits) nas linhas de programa. A mesma é feita na condição “*and*” equivale aos dois arcos da Figura 4.8; os quais conectam as transições t_1 e t_4 ao lugar Pcr_1 .

Nas figuras 4.9, 4.10 e 4.11 são apresentadas as alterações na programação do CLP decorrentes da inclusão do controle. Percebe-se que poucas alterações no programa permitem a execução do controle.

A inclusão das memórias “ $\%MW10$ ” e “ $\%MW11$ ”, na Figura 4.9, representa a primeira alteração no programa original do CLP apresentado na Figura 4.4. Isto significa que a transição t_1 é habilitada somente quando em Pcr_1 houver fichas da cor a_1 ($\%MW10$) ou da cor a_2 ($\%MW11$).

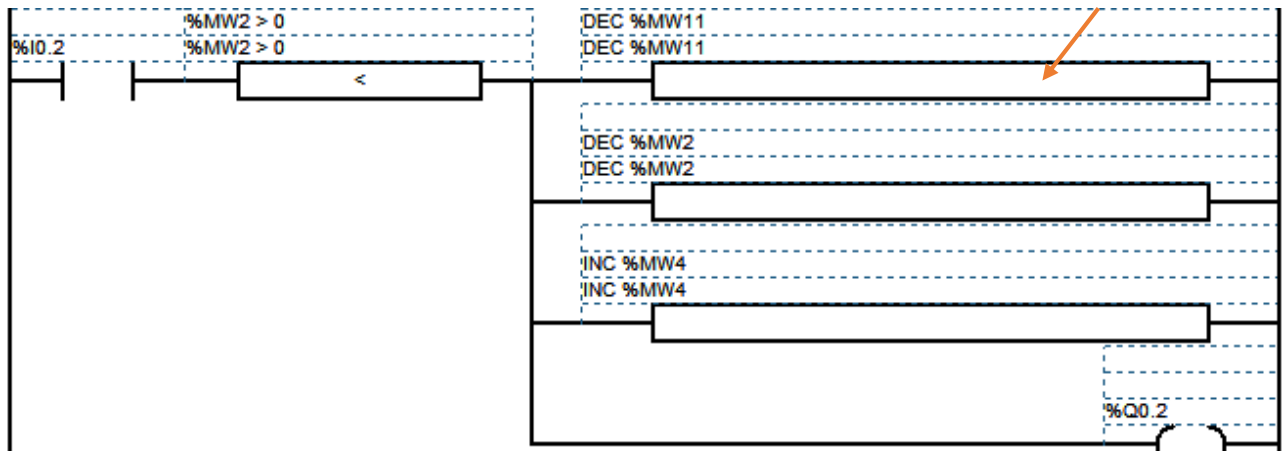
Figura 4.9 - Programa de CLP com supervisor



Fonte: (Autor, 2013)

A inclusão da memória “ $\%MW11$ ”, na Figura 4.10, apresenta a segunda alteração na programação do CLP apresentada na Figura 4.5. Isto significa que o disparo da transição t_1 relativo à ficha de cor a_2 ($\%MW11$) em Pcr_1 e p_1 ; produziu o decremento ($DEC \%MW11$ e $DEC \%MW2$) desta ficha nestes lugares, ao mesmo tempo em que incluiu-a no lugar p_2 ($INC \%MW4$).

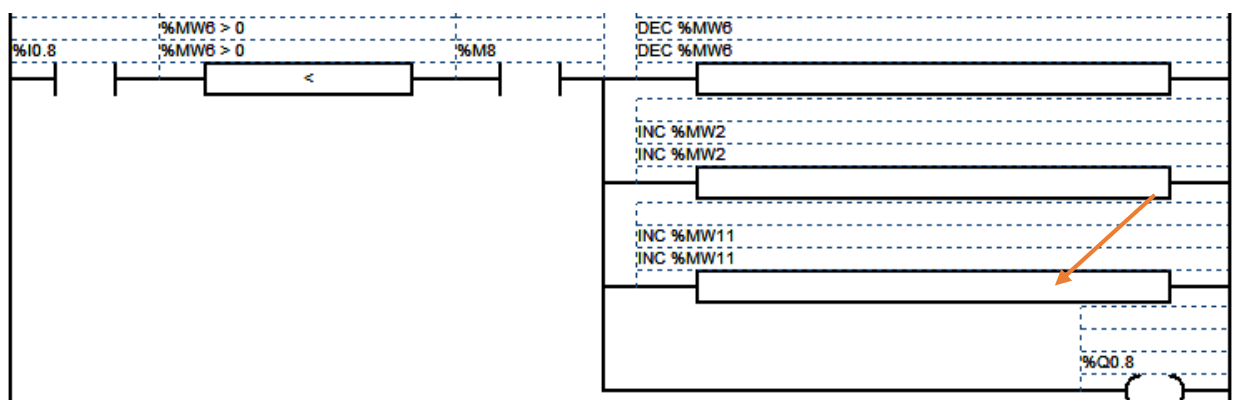
Figura 4.10 – Alteração de sub-rotina do CLP



Fonte: (Autor, 2013)

A inclusão da memória “*MW11*”, na Figura 4.11, apresenta a terceira e última alteração na programação do CLP apresentada na Figura 4.6. Isto significa que o disparo da transição t_4 relativo à ficha de cor a_2 em p_3 ; produziu o decremento (*DEC %MW6*) desta ficha neste lugar, ao mesmo tempo em que incluiu-a nos lugares Pcr_1 e p_1 (*INC %MW2* e *INC %MW11*).

Figura 4.11 – Alteração de sub-rotina do CLP



Fonte: (Autor, 2013)

4.10 Itens utilizados no estudo de caso

Neste estudo de caso foram usados os seguintes itens comerciais para realizar os testes de implementação da modelagem do SED no FMS em ambiente industrial (veja Figura 4.1):

- a) 1 CLP TWDLCAE40DRF;
- b) 1 Robô Mitsubishi modelo RV-A2J;
- c) 1 conjunto de 8 botoeiras sem identificação;
- d) 1 conjunto 8 de peças de duas cores distintas.

Veja um pequeno vídeo com a demonstração da operação FMS no endereço:
https://www.copy.com/s/yp1GrRHfWLI3/v%C3%ADdeo_rob%C3%B4_FMS.

Uma vez apresentada a aplicação do conversor numa célula de manufatura industrial (FMS) como estudo de caso para validá-la, far-se-ão no capítulo seguinte algumas discussões técnicas sobre o trabalho, bem como a explanação das dificuldades encontradas durante o desenvolvimento desse.

5 DISCUSSÕES

Neste capítulo são apresentadas discussões técnicas sobre a aplicação do método de conversão, as vantagens e desvantagens, bem como as considerações de replicabilidade.

5.1 Por que não realizar todo o controle no próprio robô?

Fundamentalmente por conta da elevada complexidade para realizar a implementação e da dificuldade de alteração das rotinas de programação, somadas às limitações de memórias e de processamento dos variados modelos de robôs encontrados.

Todavia, uma vez que a maioria dos robôs articulados verticais podem acionar rotina de seu programa por meio de um pulso gerado externamente e enviado ao mesmo, a operação acima descrita como complexa pode ser simplificada de modo que o robô não precise tratar dos conflitos de chamadas de sub rotinas, varreduras e demais. O que dessa forma permite ao mesmo manter-se em *stand by*, representando ganhos de energia elétrica, processamento, memória e aumento de vida útil, e sendo acionado somente quando um CLP tratar as entradas da célula de manufatura e enviá-las ao robô como ordem de operação.

Nesse sentido usa-se o robô como escravo do CLP, em que ambos tem programações específicas, a saber: a) robô é programado normalmente de modo a conhecer as coordenadas no espaço cartesiano onde as peças estão posicionadas, bem como as coordenadas das posições de onde se deseja que as peças sejam recolocadas; b) no CLP, a programação será tal que este receba as entradas vindas do meio externo (célula de manufatura), processe-as conhecendo as condições de habilitação das transições e dispare ao robô as saídas respectivas para executar as rotinas correspondentes.

Nessa etapa, o programador poderá incluir um buffer com sistema FIFO (*First In First Out*) ou LIFO (*Last In First Out*) para acumular as saídas de modo sequencial, ou simplesmente ignorar as novas entradas do CLP até que o robô lhe devolva um *status* de “máquina-livre”. Desta forma, um CLP trata as entradas da *FMS* e envia-as ao robô como ordem de operação.

5.2 É possível implementar sem o método de fusão dos controladores?

Sim! Tanto modelos com vários controladores quanto modelos com a fusão de controladores foram corretamente convertidos para LLD. No entanto, o modelo CPN com um único controlador, em vez de vários, é convertido num programa LLD com um menor número de memórias. No exemplo do estudo de caso (secção 4.6) seria necessário o uso de quatro memórias distintas de CLP; ao passo que, com a aplicação do método de fusão (secção 4.8) é possível atingir o mesmo resultado de controle com a redução de esforço de alteração para somente duas memórias de CLP.

5.3 Por que incluir uma memória de byte em vez de uma memória de bit para cada lugar da RPC?

O uso de uma memória de um bit ($%M$) requer uma programação mais longa para testar os valores das marcações finais de um lugar, e portanto, tornaria o programa em LLD extenso em demasia (operação que assemelhar-se-ia ao uso de um registrador de deslocamento tipo flip-flop). O uso de memória de byte ($%WM$), por sua vez, requer somente o consumo de algumas unidades de memória de dados (não muitas); o que não representa problemas considerando-se que os CLPs disponíveis em mercado trabalham com um **mínimo** de 8k bytes de memória, e portanto, suficientes para os programas desenvolvidos na referida lógica.

Em se tratando do uso de registradores mais avançados, os mesmos requerem, sobretudo, que o CLP tenha os referidos blocos / funções em seu conjunto de instruções e, portanto com custos mais elevados de *hardware* / *software*. Futuros trabalhos serão realizados, a partir deste, explorando os modelos de recursos mais avançados disponíveis nos CLPs atuais, tais como os registrados FIFO (*First Input First Output*) e LIFO (*Last Input First Output*).

5.4 Dificuldades encontradas

Durante a primeira fase deste trabalho foram realizadas várias modelagens dos SEDs em RPCs e em paralelo foram realizadas várias implementações em LLD desses modelos pelo método intuitivo.

À ocasião observou-se que as implementações dos modelos no CPN Tools e do programas em LLD no *Twido Suite* permitiam um comparativo real dos resultados. Todavia, até que fosse encontrado um padrão de correlação entre trechos (blocos) em ambos os programas, produziram-se inúmeras alterações em ambos; isto, porque cada alteração implicava na desconstrução do padrão (bloco genérico) que estava sendo construído. E, sem este, não era possível avançar para a plataforma de conversão automatizada em JAVA. Por outro lado, denotou-se, dessa atividade heurística, um ganho de conhecimento dos caminhos já percorridos tais que não alcançariam os resultados desejados.

Uma vez que no arquivo XML do modelo de RPC construído no CPN Tools dispõe da possibilidade de explorar as identificações de lugares, transições e peso dos arcos, tornou-se possível capturar a relação destes com a saída desejada em LLD a partir dos padrões identificados na etapa supracitada; e de posse disto desenvolver, via JAVA, um ambiente gráfico capaz de representar exatamente os resultados de interesse das RPCs usando a representação de LLD normatizada pela IEC 61131-3.

Posto que neste capítulo apresentaram-se discussões técnicas das soluções para o trabalho desenvolvido (fazendo uso de CLP e do LLD), bem como das dificuldades encontradas no processo, o capítulo seguinte apresentará as conclusões e trabalhos futuros como pesquisas oportunizadas a partir deste.

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi apresentada uma proposta de algoritmo de conversão de RPC para a linguagem LLD.

Para tanto fez-se a modelagem dos SED fazendo uso do aplicativo CPN Tools para explorar performance desses e suas propriedades comportamentais e estruturais. Na ocasião observou-se o comportamento do SED modelado em RPC com e sem controle supervisorio. Seguidamente, para incluir controle no SED, fez-se uso da TCS a fim explorar a identificação das situações indesejadas nestes e, a partir disto, inferir alterações na RPC que satisfizesse às restrições impostas pelo controlador. O método RCCD foi utilizado a fim de encontrar os supervisores necessários para atender às restrições dos SEDs modelados. Fez-se uso de um CLP para implementar a modelagem correspondente do SED, agora em LLD, a fim de validar em termos reais o comportamento alcançado nas simulações.

Como estudo de caso fez-se uso de uma FMS robotizada, na qual um robô opera como escravo do CLP, a fim de validar os resultados da proposta; onde evidenciou-se que com poucas alterações, num programa em LLD implementado num CLP, é possível alcançar o controle supervisorio sem comprometer as rotinas de programação já existentes no CLP, e, essencialmente, garantindo o alcance do controle das operações.

Para automatizar a conversão de RPC para LLD explorou-se o código fonte das modelagens em RPC no CPN Tools – XML – a fim de inferir sobre a correlação entre blocos. Os resultados comparativos apresentados no Apêndice B, demonstram a eficácia da conversão automatizada. Neste exploraram-se oito modelos distintos de RP e RPC construídos primeiro em RPC, depois programados correspondentemente de modo intuitivo no *software* Twido Suite e, finalmente comparados com os resultados gerados pelo conversor automático de RPC para LLD.

Como trabalhos futuros, propõe-se a expansão deste trabalho para sistemas de maior complexidade permitindo explorar as demais propriedades das RPC. Nesse sentido, estuda-se o desenvolvimento dos blocos de conversão para LLD de funções mais complexas - nas quais deseja-se implementar um função correspondente, no CLP, para cada função nos arcos da RPC modelada; RPC temporizadas – nas quais haverá a presença de *timers* de CLPs; RPC hierárquicas – nas quais far-se-á uso de sub-rotinas para acionar um conjunto específicos de instruções no CLP.

REFERÊNCIAS

BARGHASH, M. A. et al. Petri Nets and Ladder Logic for Fully-Automating and Programmable. **American J. of Engineering and Applied Sciences**, p. 252-264, 2011.

BARKAOUI, K.; CHAOUI, A.; ZOUARI, B. Supervisory control of discrete event systems based on structure theory of petri nets. **IEEE International Conference on Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation.**, 1997.

CARDOSO, J.; VALETTE, R. **Redes de Petri**. Florianópolis: [s.n.], 1997.

CURY, J. Teoria de controle supervisorio de sistemas a eventos discretos, 2001.

GEORGINI, M. **Automação Aplicada Descrição e Implementação de Sistemas Sequenciais com CLPs**. São Paulo, Brasil: [s.n.], 2007.

GOMAA, M. M. Petri net to ladder logic diagram converter and a batch process simulation. **ARPJ Journal of Engineering and Applied Sciences**, 2011.

GUERRA, F. V. D. A. **Modelagem de Sistemas com Restrições Temporais em Redes de Petri Orientadas a Objetos**. Universidade Federal de Campina Grande. Campina Grande, Paraíba, p. 2-3. 2005. (519.711).

HOPCROFT, J.; MOTWANI, R.; ULLMAN, J. Introduction to Automata Theory Languages and Computation. **3ª edição, Addison Wesley**, 2006.

HOPCROFT, J.; MOTWANI, R.; ULLMAN, J. Introduction to Automata Theory Languages and Computation, 2006.

JENSEN, K.; KRISTENSEN, L. M. Coloured Petri Nets. In: _____ **Modelling and Validation of Concurrent Systems**. Aarhus: University of Aarhus, 2009. p. 8.

LEE, J.; LEE, J. Conversion of ladder diagram to petri net using module synthesis technique. **International Journal of Modelling and Simulation**, n. 29.

LEE, J.-S.; HSU, P.-L. An improved evaluation of ladder logic diagrams and Petri nets for the sequence controller design in manufacturing systems. **Springer**, June 2004. ISSN DOI 10.1007.

MENEZES, F. A. D. A. **Síntese de Supervisores em Sistemas a Eventos Discretos Utilizando Redes de Petri Coloridas**. Universidade Federal do Ceará. Fortaleza, p. 40-41. 2011.

MENEZES, F. A. D. A.; BARROSO, G. C.; PRATA, B. D. A. Restrições de controle sobre cores decompostas: uma proposta no controle supervísório de sistemas a eventos discretos utilizando redes de petri coloridas. **Revista Controle & Automação/Vol.23 no.3/Maio e Junho**, 2012.

MOODY, J.; ANTSAKLIS, P. Supervisory Control of Discrete Event Systems Using Petri net. **Kluwer Academic Publishers**, 1998.

MORAIS, C. C. D.; CASTRUCCI, P. D. L. Engenharia de automação industrial. Rio de Janeiro: LTC, 2007. p. 279-281.

MURATA, T. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE 77(4): 541–580.**, 1989.

NATALE, F. **Automação Industrial**. 9ª. ed. São Paulo: Érica, 2007.

OLIVEIRA, E. A. D. S. et al. Convertendo Diagramas Ladder em Modelos de Redes de Petri Coloridas. **X SBAI**, São João del-Rei, Setembro 2011. ISSN 2175-8905.

PARK, E.; TILBURY, T.; KHARGONEKAR, P. Control logic generation for machining systems using Petri net formalism, v. 5, p. 3201 –3206 , 2000.

PRATA, B. A.; BARROSO, G. C.; ARRUDA, J. B. F. Um novo método para controle supervisorio de sistemas a eventos discretos baseado em redes de petri coloridas. **XV Simposio Brasileiro de Pesquisa Operacional**, 2008.

RAMADGE, P. J.; WONHAM, W. M. Supervisory control of a class of discrete event processes. **SIAM Journal on Control and Optimization** **25(1): 206–230.**, 1987.

RAMADGE, P.; WONHAM, W. The control of discrete event systems. **Proceedings of the IEEE** **77(1): 81–98.**, 1989.

SILVESTRE, R. P. **Implementação em Ladder de Sistemas de Automação Descritos por Redes de Petri Interpretadas para Controle**. Universidade Federal de Rio de Janeiro. [S.l.], p. (monografia) 2,3. 2010.

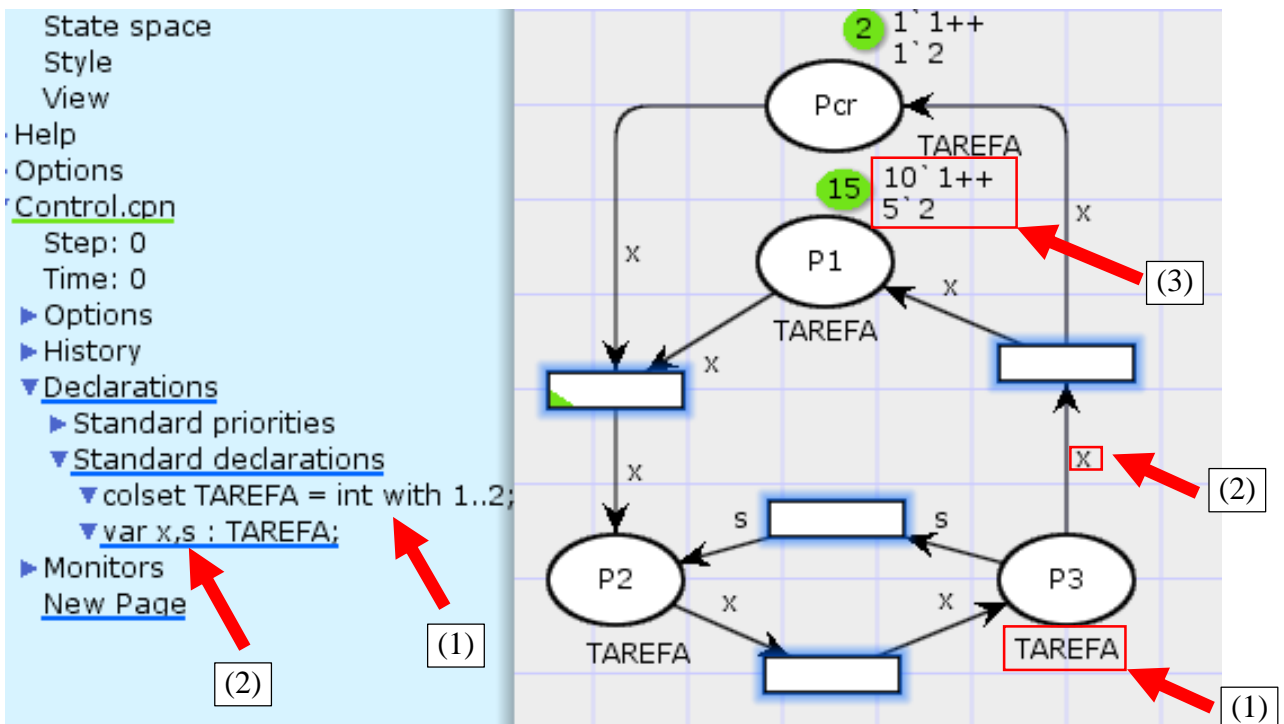
THAPA, D.; DANGOL, S.; WANG, G. N. Transformation from Petri Nets Model to Programmable Logic Controller using One-to-One Mapping Technique. **Computational Intelligence for Modelling, Control and Automation, International Conference**, n. 2, p. 228–233, 2005.

UZAM, M.; JONES, A. H. Conversion of Petri Net Controllers for Manufacturing Systems Into Ladder Logic Diagrams. **Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation**, v. 2, p. 649–655, 1996.

WU, B.; XI, L.-F.; ZHUO, B.-H. Service-oriented communication architecture for automated manufacturing system integration. **International Journal of Computer Integrated** , 2008.

APÊNDICE A – ETAPAS DE EXECUÇÃO DO CONVERSOR RPC_LLD

De posse do *software* CPNTools instalado no computador e devidamente inicializado, o programador deve criar um novo arquivo (*New net*) e salvá-lo no local de interesse. Seguidamente, deve criar o conjunto de cores (tipo inteiro) com os valores de fichas que serão usados na RPC. Na sequência, deve modelar o SED usando os lugares, fichas e arcos pertencentes aos conjuntos de cores criados (ex: “colset Fichas = int with 1..2;”), ou ao conjunto de variáveis definidas para a cor criada (ex.: “var x,s : Fichas;”), e atribuir os valores de marcações iniciais aos lugares de interesse respeitando a **faixa** definida na referida cor. A Figura abaixo ilustra esses passos.

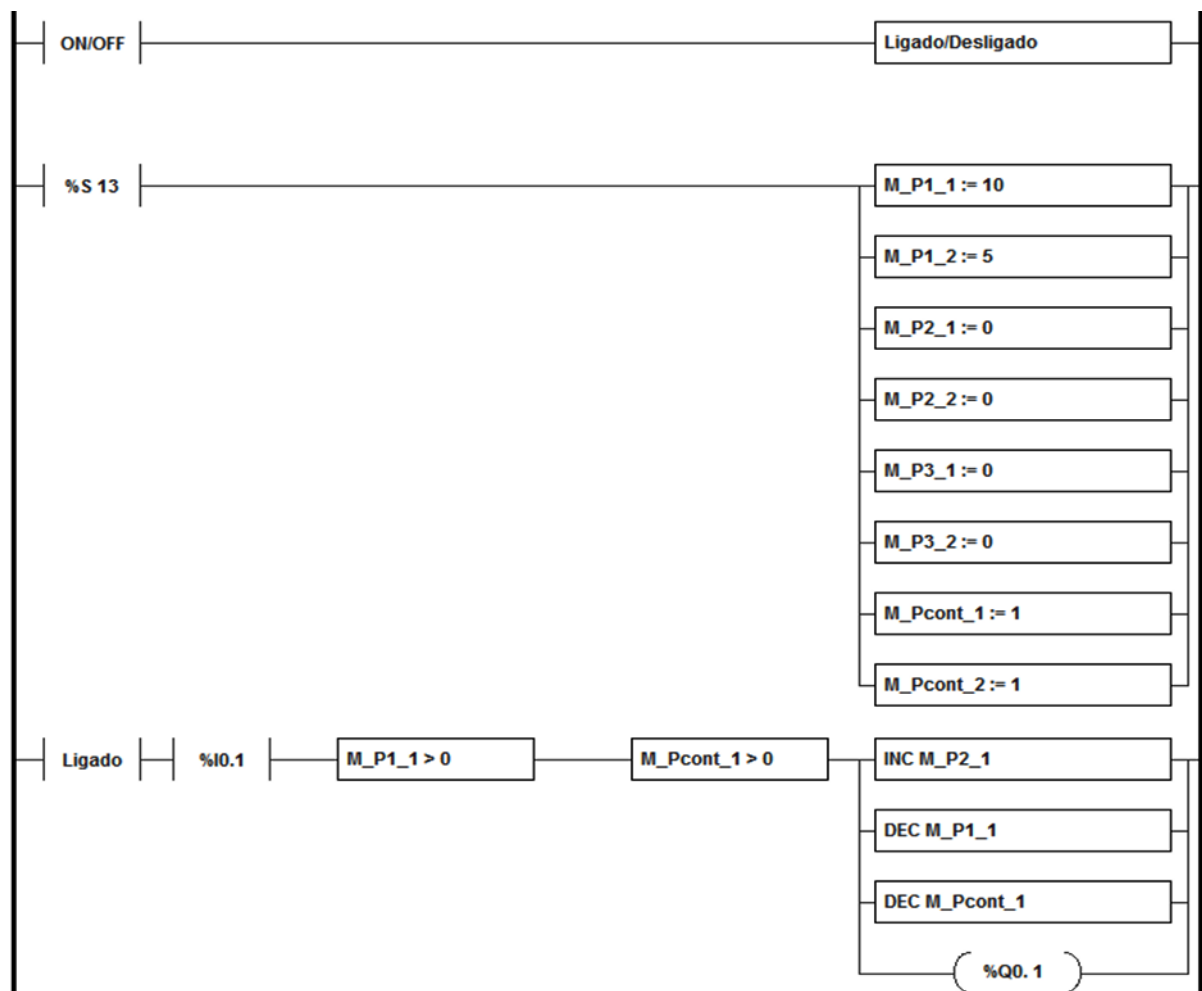


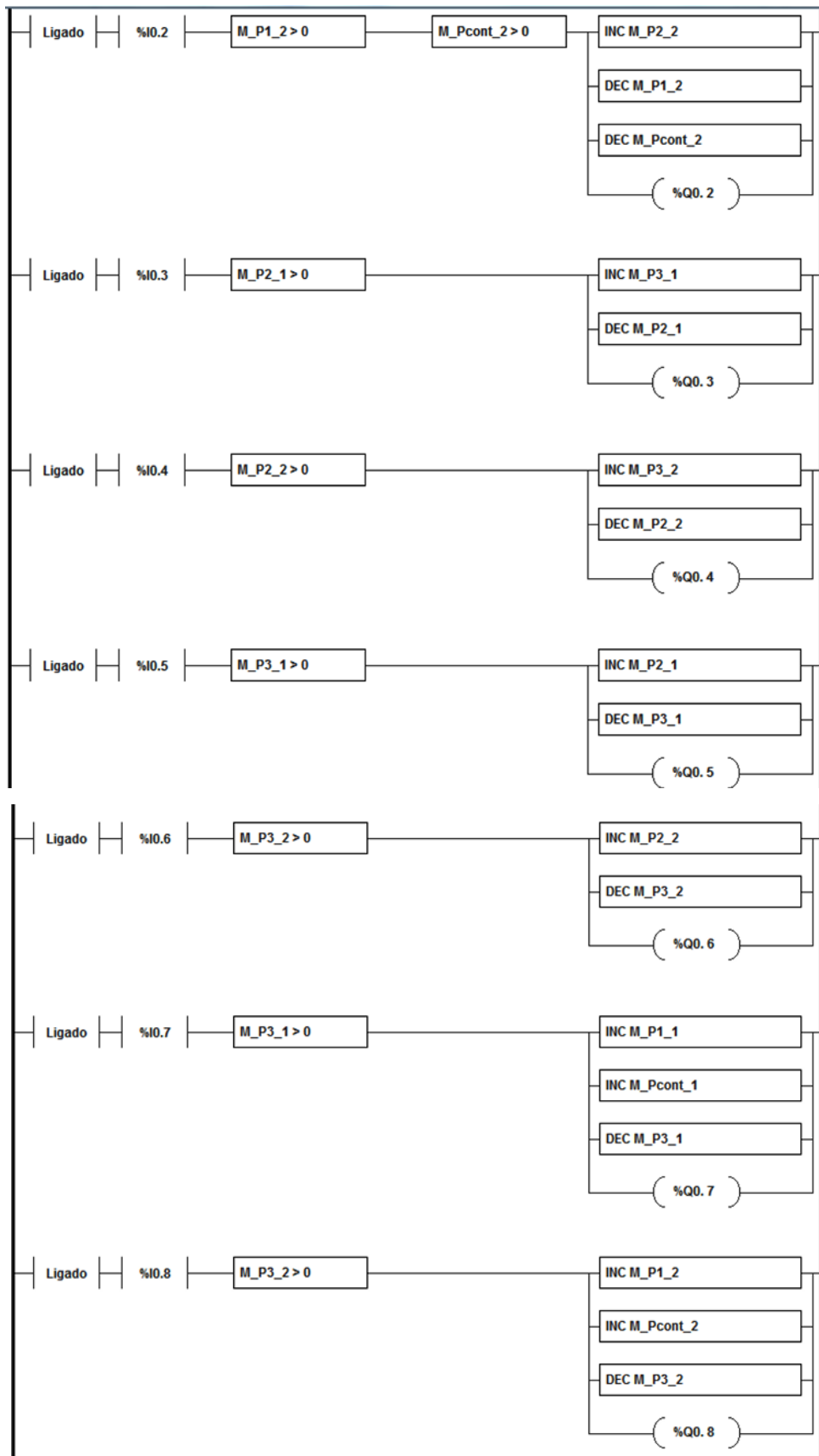
Perceba que a RPC acima modelada tem um conjunto de cores **TAREFA** (1), o qual contém uma faixa específica de tipos inteiros (**int with 1..2**) e contém ainda duas variáveis **var x,s** (2), as quais serão usadas nos arcos. Dessa forma, os locais do SED modelado na RPC tem duas cores (valores) diferentes para o conjunto TAREFA e com quantidades variáveis; por exemplo: no local p_1 (3) há 10 fichas da **cor 1** e 5 fichas da **cor 2** totalizando 15 fichas.

A modelagem no CPNTools é realizada convencionalmente pela atribuição de variáveis, arcos, transições e lugares até que o programador complete a representação do SED (finalize-a) e salve-a.

Seguidamente, de posse do arquivo de conversão de RPC para LLD (*cpnToLadder.jar*), o usuário deve executá-lo (duplo clique no arquivo). Ao visualizar a tela do mesmo que solicita o endereço do arquivo do CPNTools que será convertido, deve navegar pelo browser da tela ao local do endereço onde a modelagem (anteriormente realizada) foi salva e, ao encontrar a RPC desejada, clicar em “ok”.

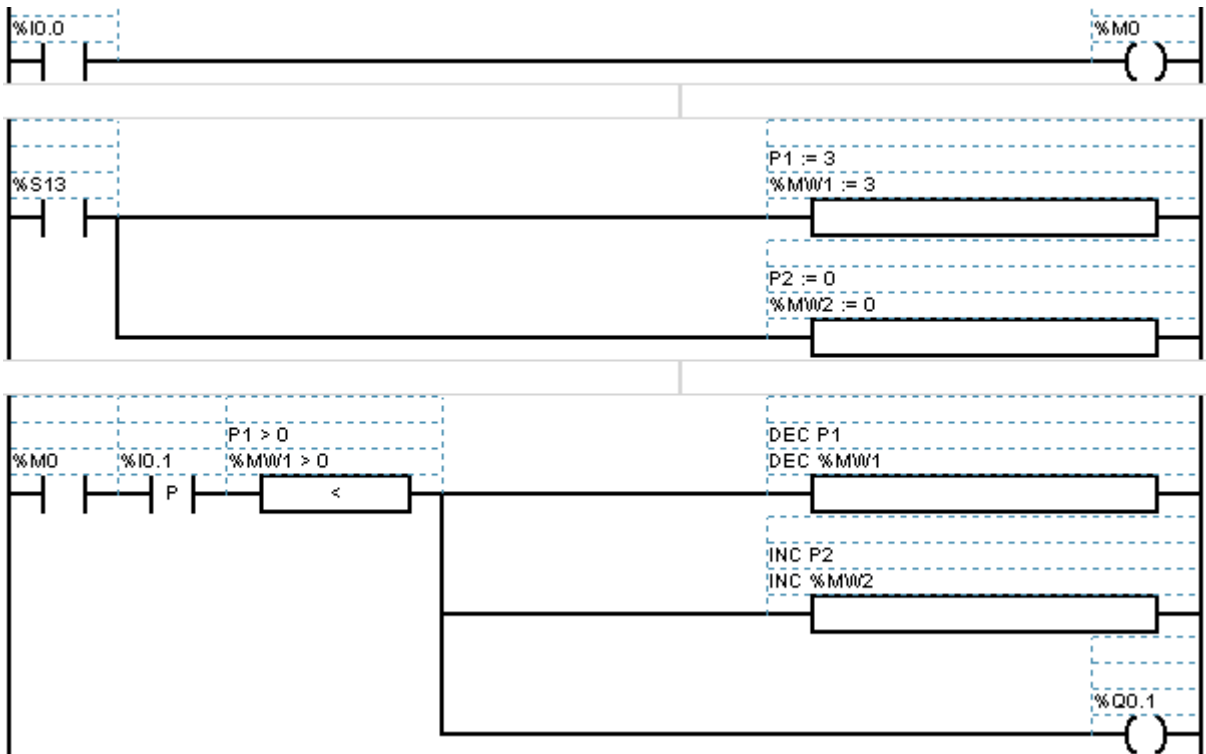
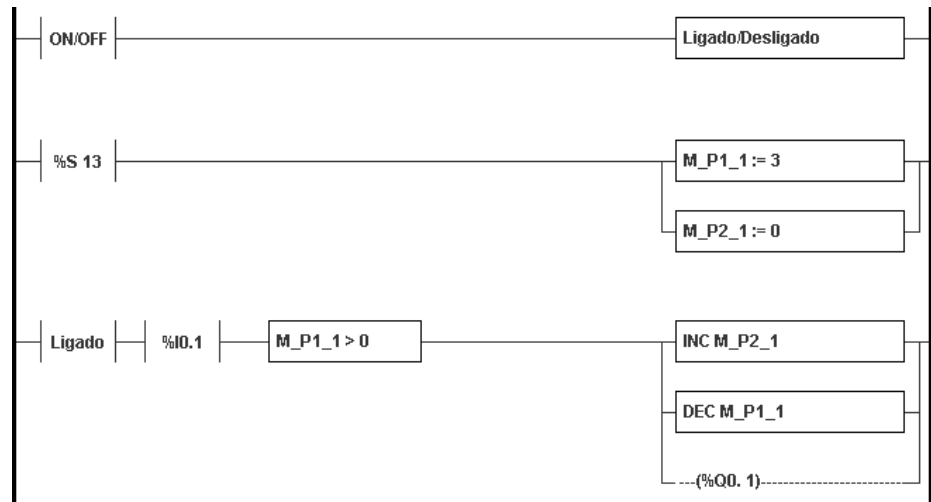
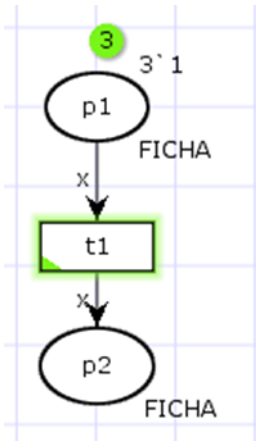
O conversor apresentará uma tela com a programação de CLP na linguagem LLD exatamente conforme definido neste trabalho e correspondendo à estrutura LLD. A partir desse ponto, o programador pode construir a programação de LLD em qualquer CLP que esteja à sua disposição guiando-se pelo resultado da conversão automatizada, o que deve ser realizado segundo a *interface* do referido modelo de CLP disponível. A Figura a seguir ilustra a imagem resultante da conversão de RPC para LLD.



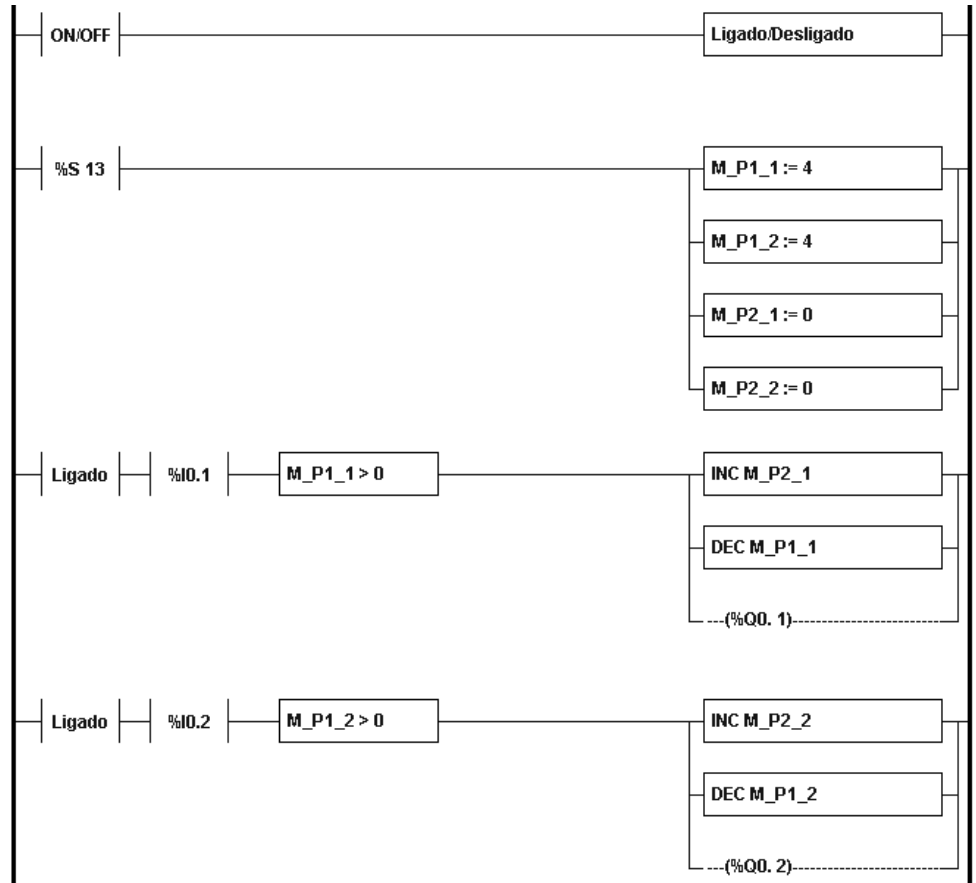
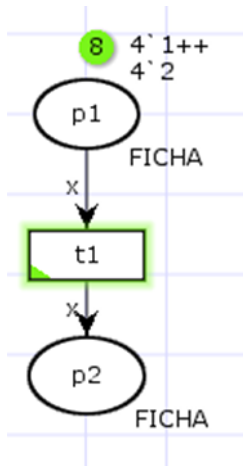


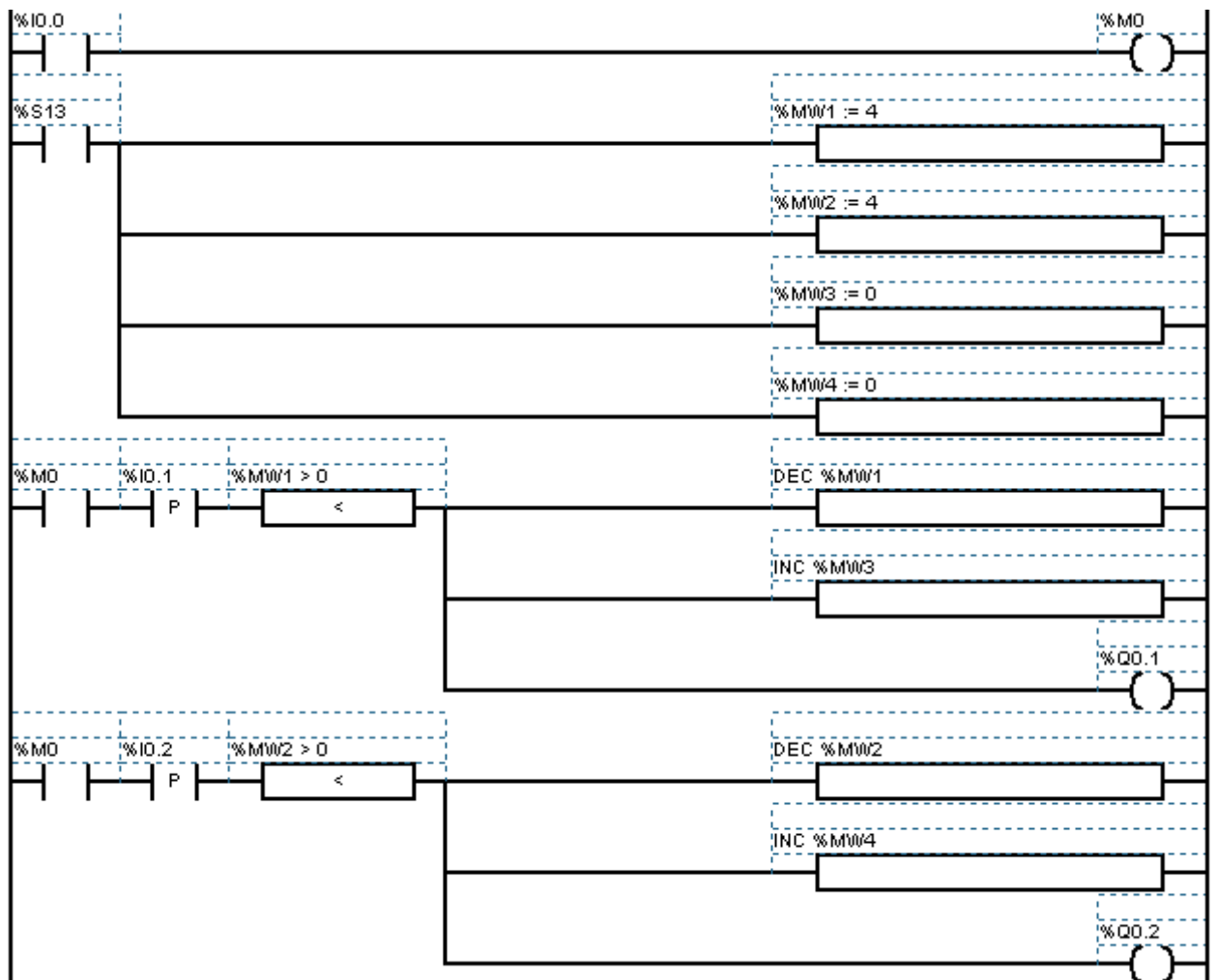
APÊNDICE B - MODELOS DE RPC CONVERTIDOS AUTOMATICAMENTE PARA LLD

Rede 1 – RP para LLD usando o conversor automatizado (ao lado) e correspondência em LLD usando método intuitivo via Twido Suite (abaixo).

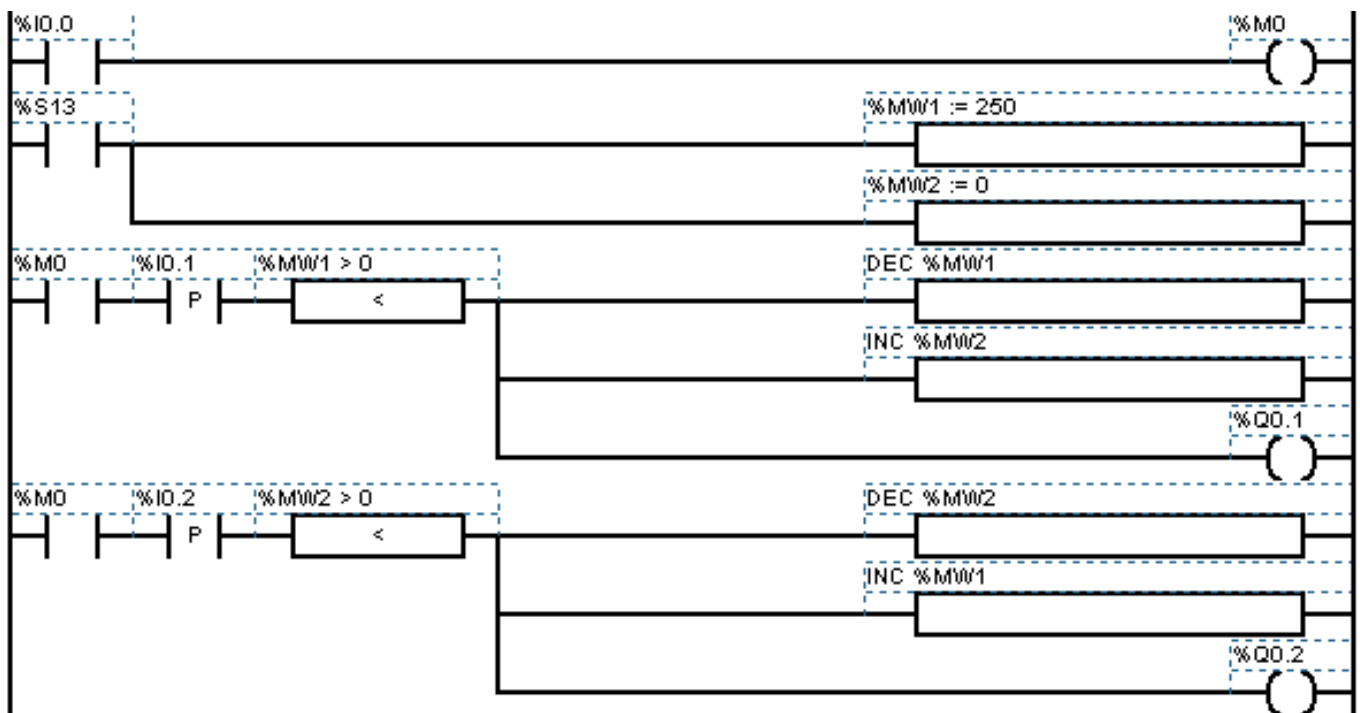
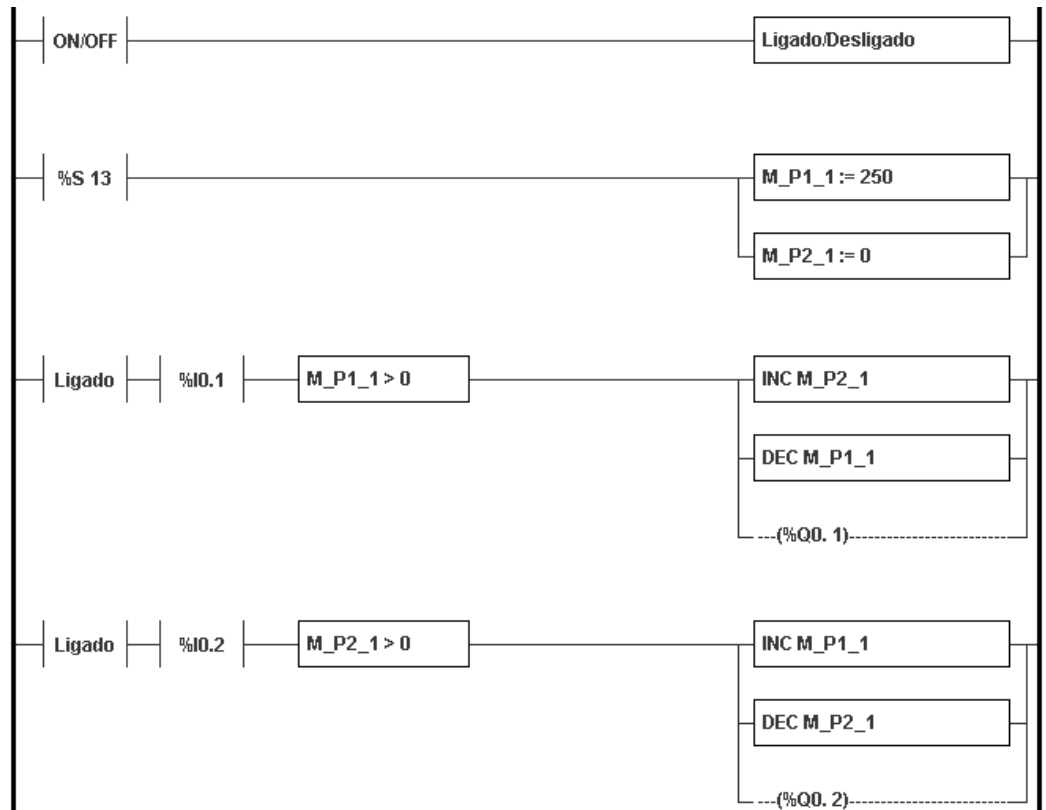
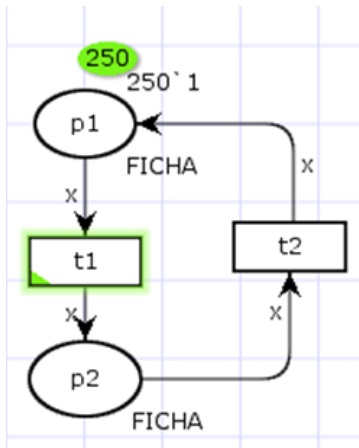


Rede 2 (Rede 1 colorida) – RPC para LLD usando o conversor automatizado (**ao lado**) e correspondência em LLD usando método intuitivo via Twido Suite (**abaixo**).

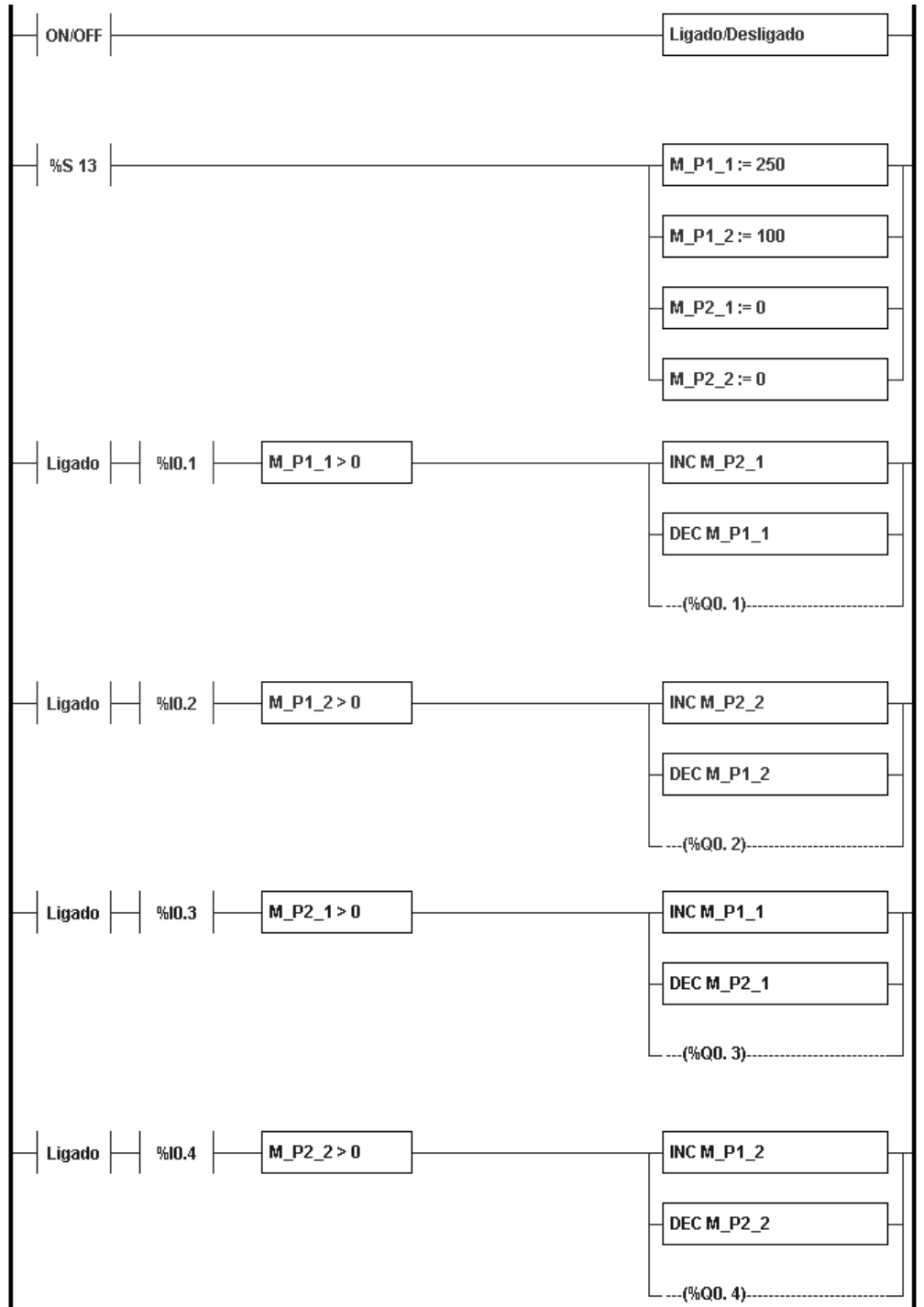
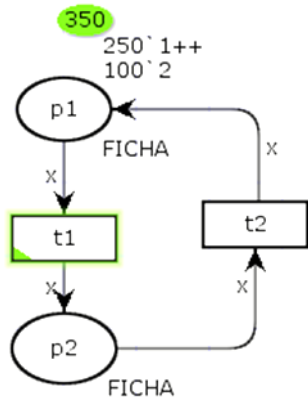


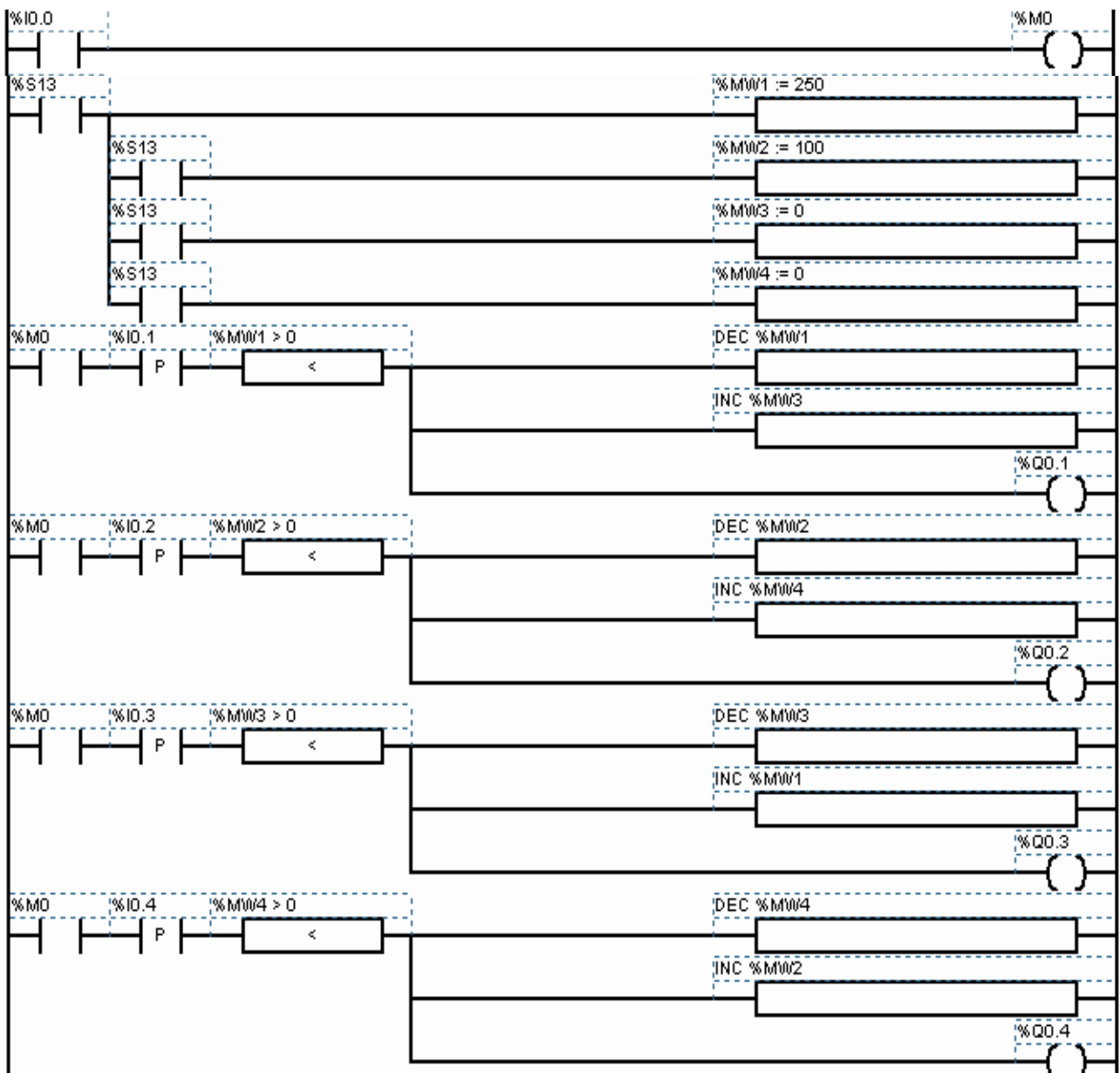


Rede 3 – RP para LLD usando o conversor automatizado (ao **lado**) e correspondência em LLD usando método intuitivo via Twido Suite (**abaixo**).

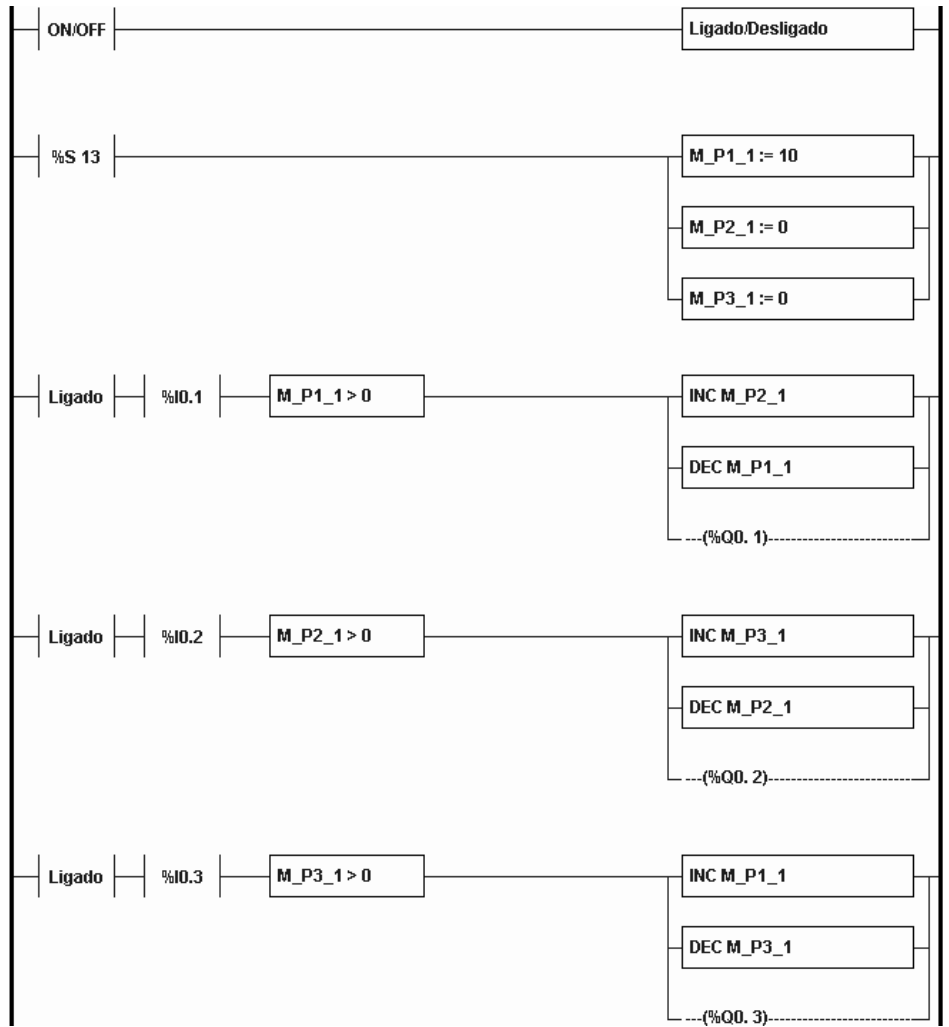
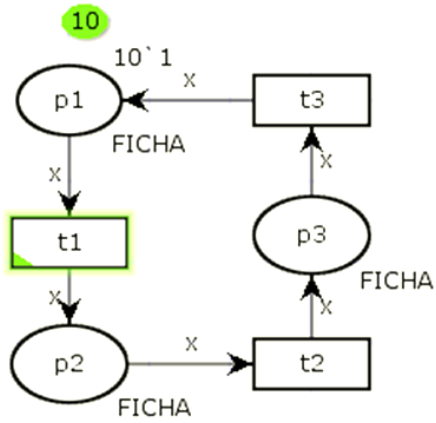


Rede 4 (Rede 3 colorida) – RPC para LLD usando o conversor automatizado (ao lado) e correspondência em LLD usando método intuitivo via Twido Suite (abaixo).

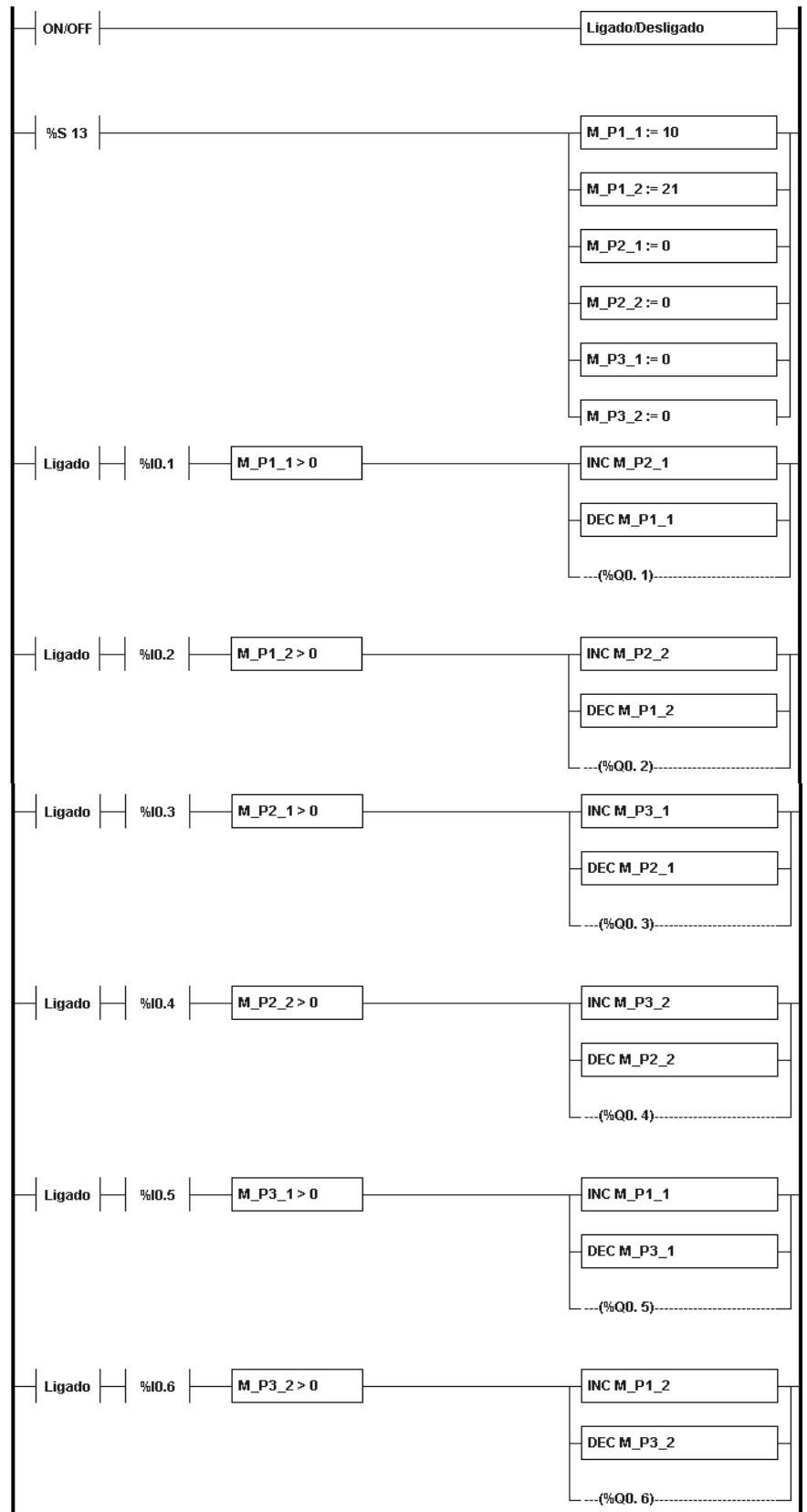
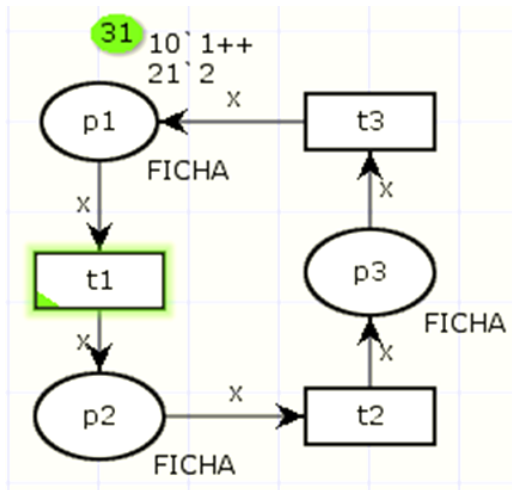


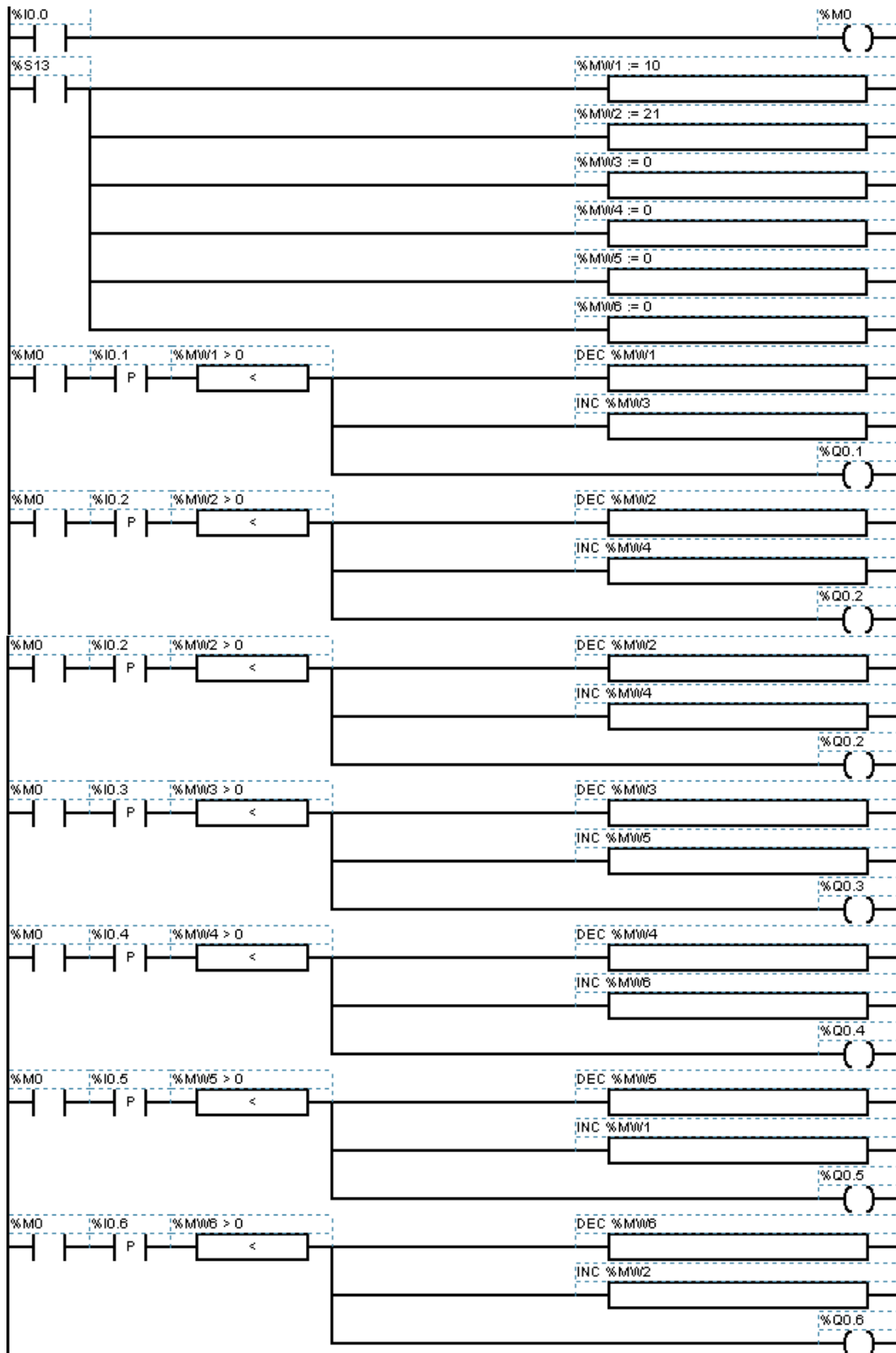


Rede 5 – RP para LLD usando o conversor automatizado (ao lado) e correspondência em LLD usando método intuitivo via Twido Suite (abaixo).

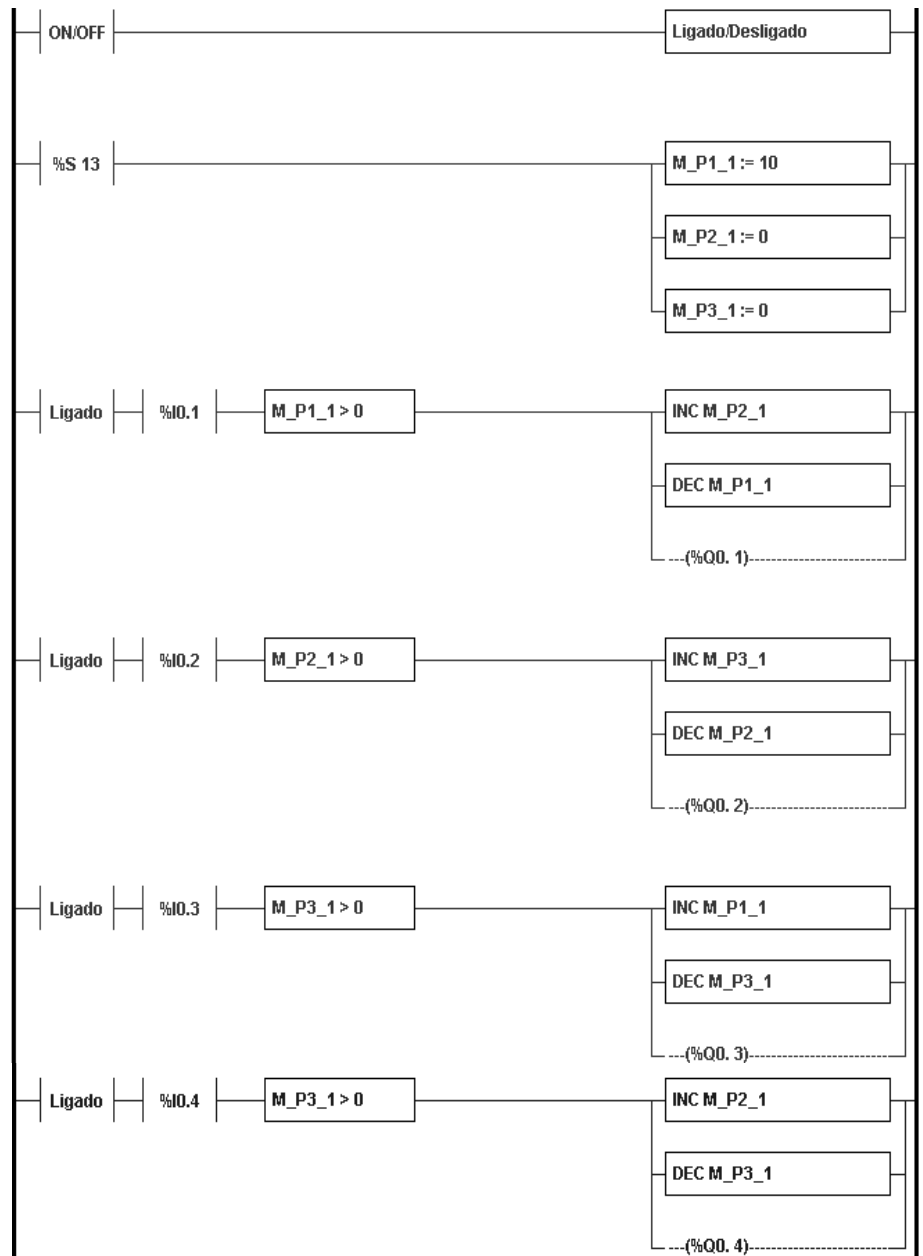
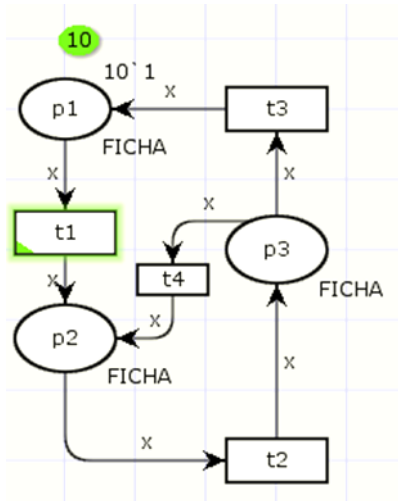


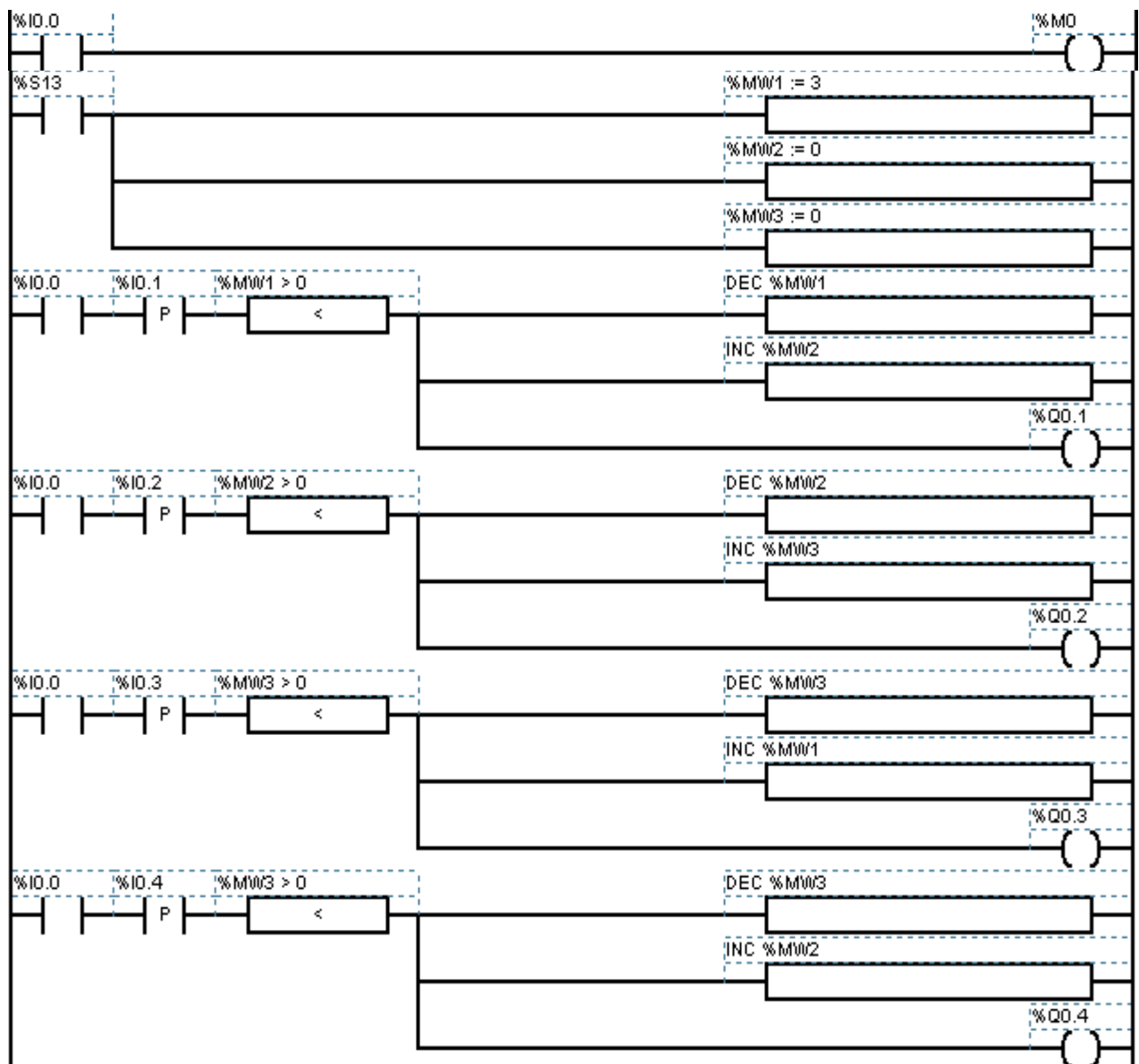
Rede 06 (Rede 5 colorida) – RPC para LLD usando o conversor automatizado (ao lado) e correspondência em LLD usando método intuitivo via Twido Suite (abaixo).



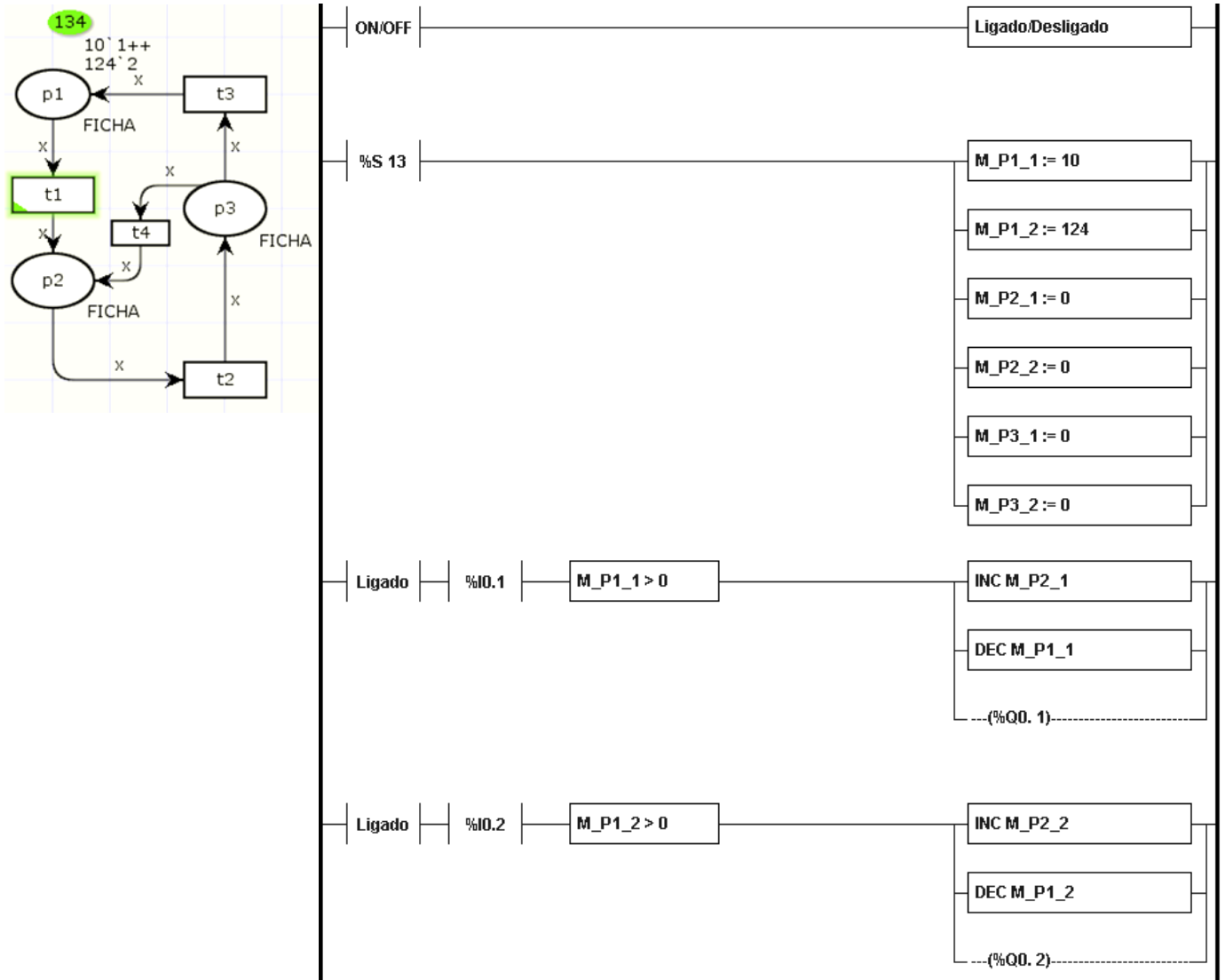


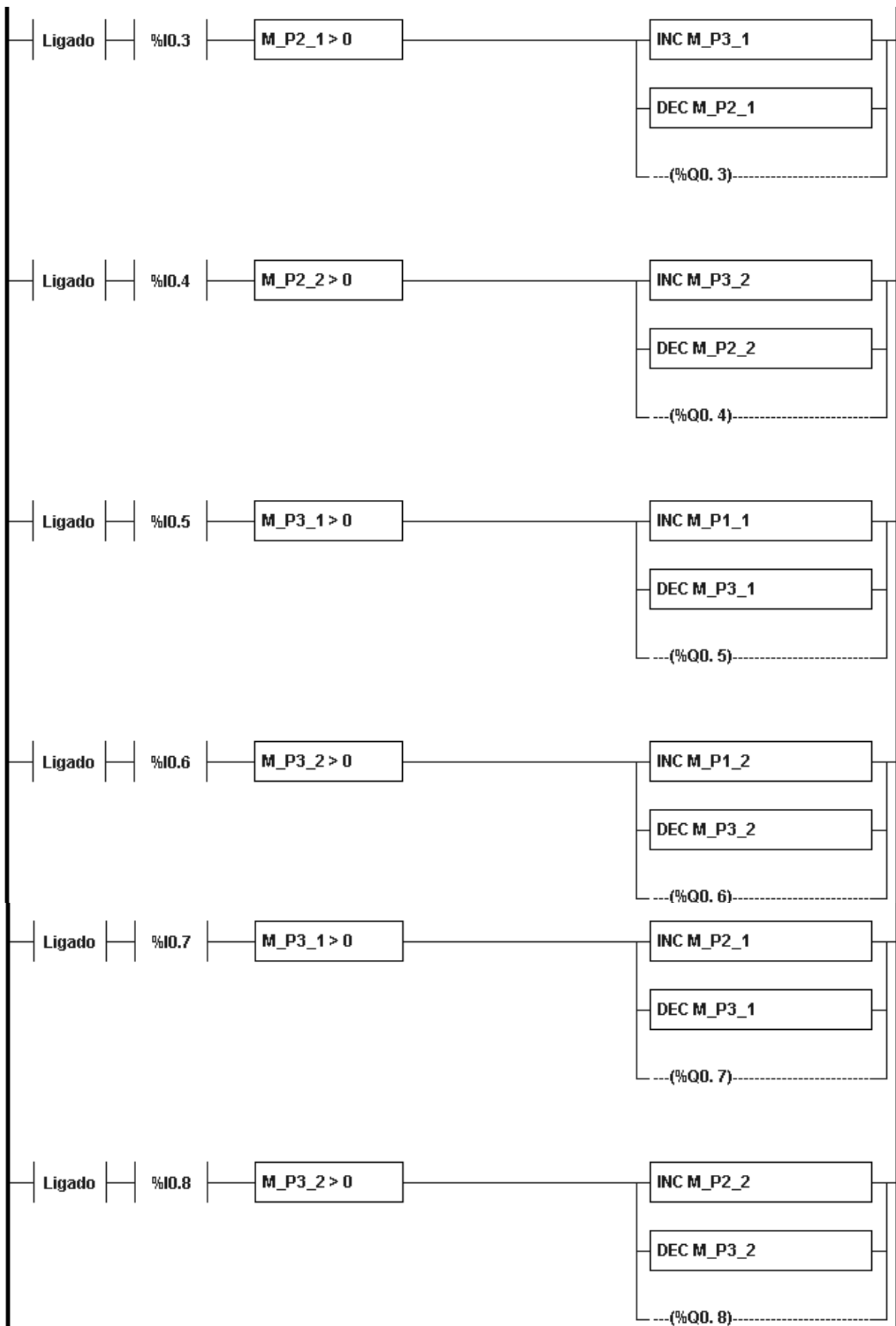
Rede 7 – RP para LLD usando o conversor automatizado (ao lado) e correspondência em LLD usando método intuitivo via Twido Suite (abaixo).

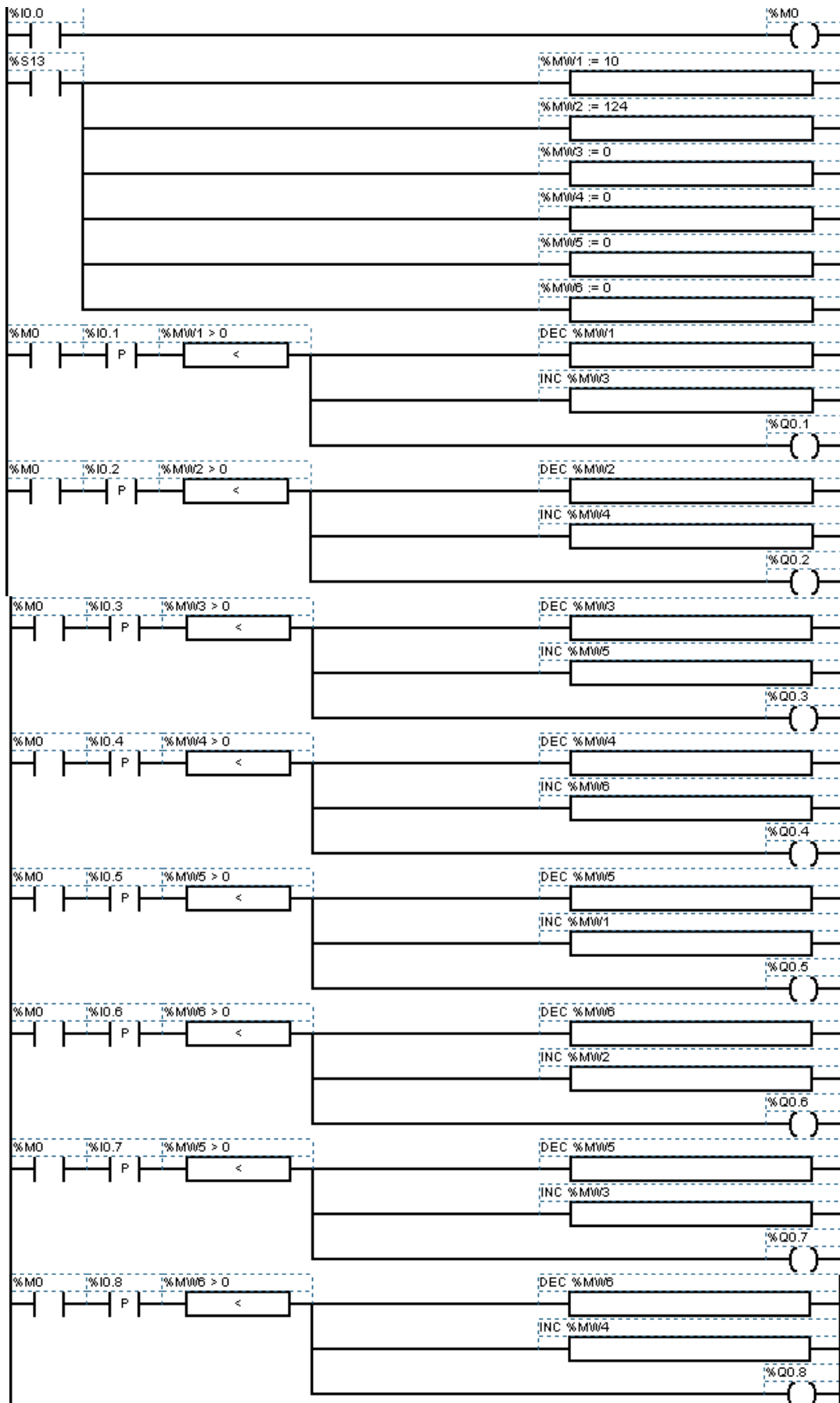




Rede 8 (Rede 7 colorida) - RPC para LLD usando o conversor automatizado (ao lado) e correspondência em LLD usando método intuitivo via Twido Suite (abaixo).







APÊNDICE C – ARTIGO PUBLICADO NO CONGRESSO SBAI 2013

USO DE RESTRIÇÕES SOBRE CORES DECOMPOSTAS COMO ALGORITMO DE CONTROLE NUMA CÉLULA PRODUTIVA ROBOTIZADA

DYONATHA R. COSTA¹, FRANCISCO A. A. MENEZES¹, GIOVANNI C. BARROSO²

1. *Departamento de Engenharia de Teleinformática, Universidade Federal do Ceará*
Av. Benjamin Brasil 1100, bloco 01, apto. 306, Maraponga, Fortaleza -CE
E-mails: drcosta@live.com, alencarmenezes@gmail.com

2. *Departamento de Física, Universidade Federal do Ceará*
Campus do Pici S/N, bloco 928, sala 39
E-mails: gcb@fisica.ufc.br

Abstract— This paper proposes conversion for flexible manufacturing system (FMS) in Colored Petri Nets (CPN) for Ladder Diagram (LD). For this we use the methodology of supervisory control for CPN nominated Control Restrictions on decomposed colors (RCCD). It emphasizes the transformation CPN controlled to LD and ease of inclusion of control. The results described here demonstrate that the methodology facilitating the work of a PLC programmer minimizing the possible errors during the programming phase. To exemplify the methodology we use a vertical articulated robot controlled via PLC, which operates compared with and without supervision.

Keywords— Supervisory control, Coloured Petri nets, discrete event systems.

Resumo — Este trabalho apresenta uma proposta de conversão de modelos de células de manufatura (FMS) em Redes de Petri Coloridas (RPC) para *Ladder Diagram* (LD). Para tanto utiliza-se a metodologia de controle supervisorio para RPC denominada Restrições de Controle sobre Cores Decompostas (RCCD). Enfatiza-se a transformação de RPC não controlada para LD e a facilidade de inclusão do controle. Os resultados aqui descritos demonstram que a metodologia facilita o trabalho de um programador de CLP minimizando os possíveis erros durante a fase de programação. Para exemplificar a metodologia é usado um robô articulado vertical controlado via CLP, que opera comparativamente com e sem supervisão.

Palavras-chave— Controle supervisorio, redes de Petri coloridas, sistema a eventos discretos.

1. Introdução

O uso de robôs como elementos-chave para maximizar o processo produtivo industrial apresenta-se como solução quando se fala em segurança, qualidade, repetibilidade, acurácia e disponibilidade. Entretanto, na perspectiva de eficiência, a ausência de técnicas de controle pode sugerir a aquisição desnecessária de tais máquinas. Em essência, deseja-se que o custo industrial de tal máquina seja absorvido pela multifuncionalidade desta sem bloqueios, conflitos de operação e com reduzida complexidade operacional.

Nesse sentido, tal objetivo poderá ser alcançado pelo uso da teoria do controle supervisorio (TCS) (RAMADGE e WONHAM, 1987) e (Ramadge & Wonham, 1989), a qual surge da necessidade de uma ferramenta sistemática (Hopcroft, Motwani, & Ullman, 2006) para especificação e implementação do controle dos sistemas a eventos discretos (SED). Nessa teoria, um SED é modelado por um autômato cujo comportamento é descrito por linguagens formais. Sob esta perspectiva, a questão chave do controle é encontrar o supervisor, tal que, associado ao sistema, gere a linguagem especificada. Três fundamentos compõem a TCS: controlabilidade de

uma linguagem, existência do supervisor e existência de máxima linguagem controlável. Entretanto, alguns inconvenientes são evidenciados ao se usar autômatos como ferramenta para modelagem de SED, tais como o crescimento exponencial do número de estados, o que limita seu uso a sistemas de menor tamanho. Diante disso, propõe-se uma abordagem da TCS utilizando como ferramenta as redes de Petri coloridas (RPC), apropriadas à modelagem de sistemas mais robustos, sem prejuízo de visualização gráfica e, com o mesmo poder de análise das Redes de Petri (RP). (Gomaa, 2011) apresenta uma ferramenta em Java para a conversão de um modelo de SED em RP para a linguagem *Ladder Diagram* (LD). Esta automatização representa uma vantagem para a programação, embora restrita às RP lugar-transição. (Barghash, Abuzeid, Al-Rabadi, & Jaradat, 2011) estudam o problema dos SED semiautomáticos em 3 níveis de funcionalidades a partir das RP e, realizando implementações destes em CLP via LD. O trabalho não explora, entretanto, as estratégias de conversão de RP em LD, fator de elevada importância para o projetista. (Morais & Castrucci, 2007) apresentam as técnicas *top-down* e *bottom-up* para modelagem de um SED em RP e posterior

conversão para LD apresentando uma tabela de correlação entre esses. Tais técnicas, embora mantenham uma programação bem estruturada, não permitem ao programador aplicar controle sem grandes alterações na RP e no programa.

O objetivo deste trabalho é a utilização do método de controle de SED denominado Restrições de Controle sobre Cores Decompostas (RCCD) para a implementação de um algoritmo de controle numa célula de produção robotizada. Neste faz-se uso do método Fusão de Controladores e das RPC (Prata, Barroso, & Arruda, 2008).

A estrutura deste trabalho compreende uma pequena introdução da TCS e das RP na Seção 2; a apresentação do método RCCD e do método de fusão de controladores, na Seção 3; a metodologia proposta para transformação de um modelo RPC em LD, com e sem controle, bem como um exemplo de aplicação, são apresentados na Seção 4; as discussões sobre a operação do robô via CLP, na Seção 5; e as conclusões, na Seção 6.

2. A modelagem de sistemas

2.1 Conceitos de redes de Petri

As RP são uma ferramenta matemática e gráfica para modelagem de SED. Estas consistem de um grafo direcionado com dois tipos de nós: lugares e transições, e arcos direcionados e ponderados conectando lugares a transições e transições a lugares, bem como fichas (inteiros positivos) associadas aos lugares, conforme (Murata, 1989).

As redes de Petri coloridas (RPC) (Jensen & Kristensen, 2009) são uma extensão às RP que combinam a estrutura de uma RP com uma linguagem de programação. Assim, pode haver diferenciação de fichas e associação de variáveis e funções aos arcos da rede em substituição aos pesos dos arcos das RP. Sendo assim, as RPC são capazes de modelar sistemas mais complexos de forma mais compacta e de fácil visualização.

2.1 Controle supervisorio

O desafio de encontrar o controle supervisorio numa rede é, de um modo geral, resolvido pela TCS, que separa o sistema a ser controlado (*open loop dynamics*) do controlador (*feedback control*) e envolve a modelagem, especificação do comportamento e síntese do supervisor. No processo, a síntese de um supervisor é realizada para um dado modelo com o objetivo de satisfazer uma especificação de comportamento desejada. Dessa forma, o supervisor é um agente externo que possui habilidade de observar os eventos gerados pelo sistema e influenciar no seu comportamento através de entradas de controle. Em malha fechada, a ação de controle do supervisor garante que o funcionamento do sistema esteja de acordo com uma especificação dada.

Na TCS, segundo (Barkaoui, Chaoui, & Zouari, 1997) e (Cury, 2001), um SED é modelado por um par de linguagens, L e L_m , bem definidas, mais o conjunto de eventos Σ . L representa o conjunto de todas as sequências de eventos que o sistema pode gerar a partir de seu estado inicial e L_m é a linguagem marcada que representa as tarefas completadas do sistema. Seja:

$$G = (Q, \Sigma^*, \delta, \Sigma, q_0, Q_m) \quad (1)$$

o gerador associado a estas duas linguagens. Por definição, Σ^* representa o conjunto de todas as sequências finitas em Σ , incluindo a sequência nula ϵ ; Q é o conjunto de estados de G ; $\delta: \Sigma \times Q \rightarrow Q$ é a função de transição de estados, tal que, $\delta(\epsilon, q) = q$ e $\delta(\sigma, q) = q'$ onde $q, q' \in Q$ e $\sigma \in \Sigma$; $q_0 \in Q$ é o estado inicial de G e $Q_m \subseteq Q$ é o conjunto de estados marcados. $\Sigma: Q \rightarrow 2^\Sigma$ denota o conjunto ativo de eventos num estado, tal que, $\delta(q, s)$ é definida.

Segue que:

$$L = \mathcal{L}(G) = \{s \in \Sigma^* : \delta(q_0, s) \text{ é definida}\}, \quad (2)$$

$$L_m = \mathcal{L}_m(G) = \{s \in \mathcal{L}(G) : \delta(q_0, s) \in Q_m\}, \quad (3)$$

G é dito não-bloqueante (trim) quando

$$\mathcal{L}(G) = \overline{\mathcal{L}_m(G)}, \quad (4)$$

caso contrário é dito bloqueante.

Quando o autômato G tem um “comportamento proibido” (gera tarefa não especificada), o objetivo do controle é então restringir o comportamento do sistema não controlado, descrito por G , dentro dos limites do “comportamento não-proibido”, dado como um subconjunto de $\mathcal{L}(G)$; esse controle pode ser alcançado conectando o sistema em malha com um supervisor S (um controlador externo), sob a restrição que S nunca desabilite um evento não controlável. O sistema resultante em malha fechada denotado por S/G , a linguagem gerada $\mathcal{L}(S/G) \subseteq \mathcal{L}(G)$ e sua linguagem marcada consiste exatamente das sequências marcadas de G que estão sob o controle de S . O “comportamento não-proibido” é dado como um subconjunto de $\mathcal{L}_m(G)$. Assim, uma propriedade importante que S deve satisfazer é ser um supervisor próprio, dessa forma, o sistema sob supervisão S/G é não-bloqueante, ou seja,

$$\mathcal{L}(S/G) = \overline{\mathcal{L}_m(S/G)}, \quad (5)$$

3. O método de Restrições de Controle Sobre Cores Decompostas (RCCD)

3.1 RCCD

Aproveitando-se das características das RPC, o método restrições de controle sobre cores decompostas (RCCD), para a síntese de

supervisores, desenvolve-se a partir da análise da matriz de incidência da RPC, com a decomposição dessa matriz em outras matrizes de coeficientes de uma mesma variável (Menezes, Barroso, & Prata, 2012).

Dessa forma é proposta uma solução para síntese de supervisores em RPC, utilizando a ferramenta CPN Tools (Jensen & Kristensen, 2009).

Observe que um lugar em uma RPC pode conter fichas de diferentes cores, as quais podem ou não sofrer restrições (controle), independentemente. Nesse caso, definem-se dois conjuntos de fichas:

- fichas de restrições, que são as fichas que estão submetidas a alguma restrição (controle).
- fichas complementares de fluxo, que são as fichas do conjunto que não sofrem restrições.

Dado um SED não controlado, uma especificação de controle, o modelo em RPC e a marcação inicial da rede, pode-se obter um supervisor seguindo os passos apresentados por (Menezes, Barroso, & Prata, 2012).

3.2 A fusão de controladores

No caso em que o RCCD é aplicado a uma rede que apresente diferentes conjuntos de cores associados a diferentes lugares, e expressões de arcos compostas por variáveis distintas, podem-se definir as especificações como um só conjunto, sem prejuízo para o sistema modelado, o que leva à Fusão de Controladores (FC). Esse controlador resultante agrega toda ação dos lugares de controle (*Controladores*).

Dada uma RPC com conjuntos de cores representados por variáveis distintas, sejam B_i e M_0 , respectivamente, a matriz das restrições com as relativas fichas complementares de fluxo e a marcação inicial da RPC controlada, então a marcação inicial e a matriz de incidência de fusão do supervisor são, respectivamente,

$$M_0 = \sum_{i=1}^r B_i + L_i M_0, \quad (6)$$

$$\mathcal{H} = -(\sum_{i=1}^r L_i)C. \quad (7)$$

A prova matemática deste teorema pode ser vista em (Menezes, Barroso, & Prata, 2012).

4. Metodologia para a conversão de uma RPC para LD

4.1 Contexto industrial

A programação de um robô numa célula de manufatura pode ser simplificada pela inserção de um CLP, o qual tratando as informações de uma rede, envia ao robô somente os pulsos de ordem de operação. Desta forma, ganha-se na redução da complexidade da programação do robô, uma vez que

este será, para o sistema, um simples atuador (*slave*) de manufatura flexível (FMS).

Portanto, a metodologia consiste em:

- Modelar o FMS em RPC sem controle;
- Implementar um algoritmo de trabalho no FMS via CLP, baseado no modelo RPC, usando-se a metodologia de conversão;
- Baseado nas restrições, aplicar o RCCD para encontrar o(s) supervisor(es), conforme 3.1;
- Realizar a fusão dos supervisores, conforme 3.2;
- Incluir o(s) lugar(es) de controle no modelo RPC do FMS com a fusão de supervisores conforme passo anterior;
- Incluir o controle no algoritmo de trabalho no FMS via CLP usando a metodologia de conversão;
- Programar o robô como *slave* do CLP.

A ação acima pode ser usada no contexto industrial onde há algoritmos de controle já implementados nos CLP's de FMS, ou ainda pode ser usada para o projeto de FMS. Desta forma, explora-se o quão baixo é o esforço de implementação e/ou alteração na programação do CLP para implementar o controle supervisorio por RCCD.

4.2 Descrição de célula de manufatura flexível (FMS)

Esta secção apresenta a aplicação de RCCD na implantação do controle supervisorio em uma célula de manufatura robotizada, onde foram usados: a) 1 CLP TWDLCAE40DRF; b) 1 Robô Mitsubishi modelo RV-A2J; c) 1 conjunto de 8 botoeiras sem identificação; d) 1 conjunto de peças de duas cores distintas.

Seja a célula de manufatura apresentada na Figura 1.

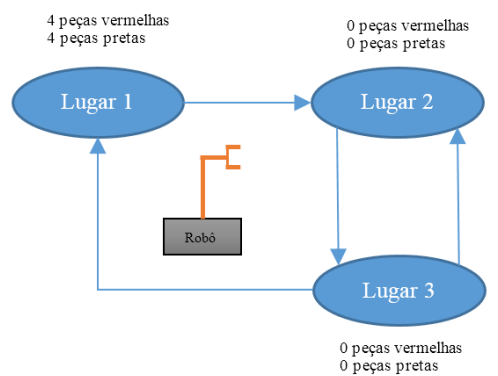


Figura 1 - Célula de manufatura robotizada

Nela, um robô precisa posicionar peças de diferentes cores entre 3 lugares e com um *loop* de processo em caso de falha de controle de

qualidade, representado pela seta que liga o lugar 3 ao lugar 2.

Da forma em que se encontra o sistema, o robô pode pegar peças **aleatoriamente**, e sem controle da quantidade de peças, entre quaisquer lugares. Ocorre que a capacidade real de processamento e de reproprocessamento nos lugares 2 e 3 é limitada a uma única peça de cada cor.

Esta situação caracteriza a ausência de um controle e, portanto, torna oportuna a aplicação do RCCD com as restrições de número de peças nos lugares supracitados.

4.3 Modelagem do FMS sem controle

Na rede da Figura 2 é apresentado o modelo do comportamento não controlado do robô. Observe que enquanto houver fichas em *P1* (peças disponíveis para a execução de operações fabris), a transição *t1* estará habilitada e, assim, pode-se colocar até 8 fichas no lugar *P2*. Mas, se *P2* for um lugar limitado, é necessário exercer controle para limitar o número de fichas (peças) nesse lugar. O mesmo ocorre com *P3*.

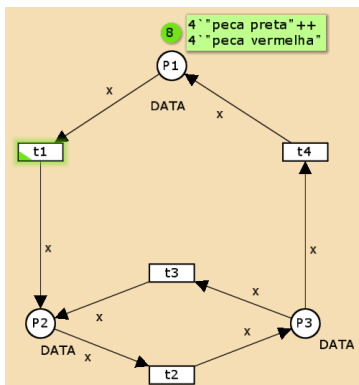


Figura 2 - Modelagem do FMS em RPC

4.4 Conversão do modelo sem controle no programa diagrama Ladder para CLP

A primeira ação é converter o modelo sem controle em um programa de CLP. Nesta ação, como *t1* possui somente um lugar de entrada e um de saída, então o disparo de *t1*, para cada cor, é convertido no trecho de programa apresentado nas figuras 3 e 4.

Se no lugar *P1* houvesse mais fichas de outras cores, bastaria acrescentar uma linha de programa com uma entrada externa para cada cor.

Assim ‘*n*’ cores corresponderiam a ‘*n*’ linhas de programa.

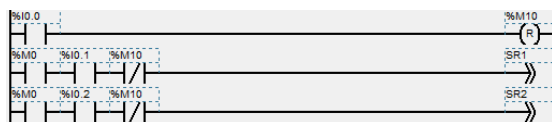


Figura 3 - linha de programa do CLP chamando respectiva sub-rotina

Na Figura 3 são mostradas duas rotinas de varredura, realizadas quando um sensor de proximidade ativa uma das oito entradas do CLP.

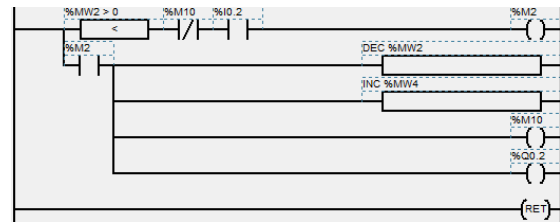


Figura 4 - sub-rotina do CLP condição de saída de P1

A Figura 4 apresenta a ação executada quando blocos de memórias (“%MW2” e “%MW4”) recebem respectivamente decremento e incremento representando uma ficha que sai do lugar *P1* (*P4*) para o lugar *P2* (*P3*), conforme Figura 2.

4.5 Aplicação de RCCD

Seja a célula manufatura apresentada na Figura 1. Seja a Figura 2 a modelagem desta em RPC. Observa-se que esta possui apenas um conjunto de cores (DATA) relativo a todos os lugares da rede e que a todos os arcos é associada a variável *x*. Nesse caso, fazendo uso do algoritmo RCCD (MENEZES, BARROSO e PRATA, 2012) temos que: $D = D_x$ e $C = C_x$.

Impondo as especificações de restrições descritas em 4.2, $M(p2) \leq l'$ “peça preta” e $M(p3) \leq l'$ “peça vermelha” tem-se, da RPC não controlada, a matriz de incidência relativa à variável *x* em (10).

$$D^x = \begin{pmatrix} -1(x) & 0 & 0 & 1(x) \\ 1(x) & 1(x) & -1(x) & 0 \\ 0 & -1(x) & 1(x) & -1(x) \end{pmatrix} \quad (8)$$

o que implica na matriz de referência de incidência, da RPC não controlada, relativa à variável *x* em (11):

$$C^x = \begin{pmatrix} -1 & 0 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 0 & -1 & 1 & -1 \end{pmatrix} \quad (9)$$

O vetor *L* indica os lugares que sofrem as restrições:

$$L = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (10)$$

Assim, a matriz C_c , do supervisor:

$$C_c = -LC = \begin{pmatrix} -1 & -1 & 1 & 0 \\ 0 & 1 & -1 & 1 \end{pmatrix} \quad (11)$$

mostra que o supervisor possui dois lugares de controle, *PC1* e *PC2*.

A primeira linha da matriz refere-se à restrição $M(p2) \leq l'$ “peça preta” e diz que o lugar de

controle *PC1* é entrada para as transições *t1* e *t2* e saída para *t3*. A segunda linha refere-se à restrição $M(p3) \leq (1' \text{ "peça vermelha"})$ e diz que o lugar de controle *PC2* é entrada de *t3* e saída de *t2* e *t4*. A seguir, as marcações iniciais dos lugares de controle *PC1* e *PC2* são adquiridas em (14), sabendo-se que a matriz *B* que apresenta a soma das restrições com as respectivas fichas complementares de fluxo é:

$$B = \begin{pmatrix} 1''a_1'' + 2''a_2'' \\ 1''a_2'' + 2''a_1'' \end{pmatrix} \quad (12)$$

e a marcação inicial da RPC não controlada é

$$M_0 = \begin{pmatrix} 2''a_1'' + 2''a_2'' \\ 0 \\ 0 \end{pmatrix} \quad (13)$$

Dessa forma,

$$M_{c_0} = \begin{pmatrix} 1''a_1'' + 2''a_2'' \\ 1''a_2'' + 2''a_1'' \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2''a_1'' + 2''a_2'' \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1''a_1'' + 2''a_2'' \\ 1''a_2'' + 2''a_1'' \end{pmatrix} \quad (14)$$

Onde: "*a₁*" é uma peça preta e "*a₂*" uma vermelha.

As linhas da matriz *M_{c0}* são as marcações iniciais dos lugares de controle *PC1* e *PC2*.

4.6 A fusão dos controladores da RPC

Os controladores *PC1* e *PC2* satisfazem às condições de 3.2 e, portanto, a fusão destes produz o lugar de controle *Pcr1*. O qual tem a matriz de incidência de fusão:

$$H = (-1 \ 0 \ 0 \ 1) \quad (15)$$

e a marcação inicial da fusão

$$M_0 = (1''a_1'' + 1''a_2'') \quad (16)$$

O supervisor resultante da fusão (*Pcr1*), nesse caso, passa a exercer o mesmo controle que os controladores *PC1* e *PC2*.

4.7 Modelagem do controle supervisor após a fusão de controladores

Fazendo-se uso da técnica de fusão de controladores, 4.6, obtém-se uma rede controlada com um modelo mais simplificado, o qual, por esta razão, encontra elevada atratividade industrial. A Figura 5 apresenta o resultado desta rede após a fusão dos controladores com a manutenção das restrições impostas no RCCD.

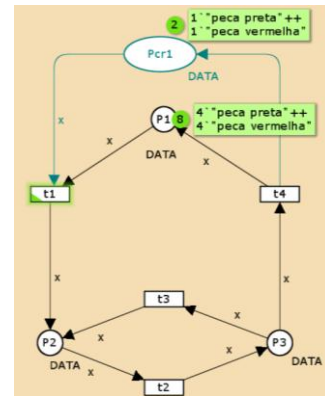


Figura 5 - RPC com fusão de controladores

4.8 Implementação do programa com controle em CLP

Uma vez concluídas as etapas anteriores, ou ainda de posse de um programa de CLP já pronto, torna-se simples a implementação de controle em CLP, pois, acrescentar um lugar de entrada a uma transição equivale a um "and" na programação do CLP. Basta, então, incluir blocos de memórias, na condição "and", nas linhas de programa que equivalem aos dois arcos da Figura 5, os quais conectam as transições "t1" e "t4" ao lugar "Pcr1". As Figuras 6, 7 e 8 mostram os impactos da inclusão do controle na programação do CLP. Percebe-se que poucas alterações no programa permitem a execução do controle.

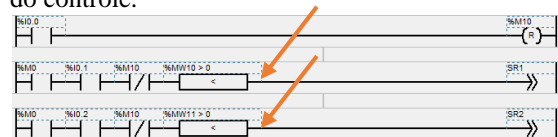


Figura 6 - Programa de CLP com supervisor

A inclusão de memórias nas linhas de programa na Figura 6, apresenta a primeira alteração no programa original do CLP (Figura 3).

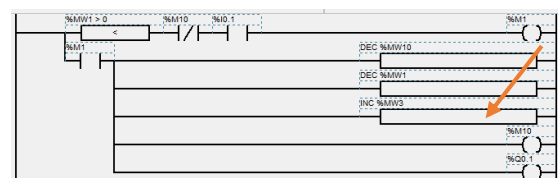


Figura 7 - Sub-rotina do CLP com alteração do controle supervisor condição de saída de *Pcr1*.

A inclusão de memória para registrar o decremento de *Pcr1*, na Figura 7, apresenta a segunda alteração na programação do CLP (Figura 4).

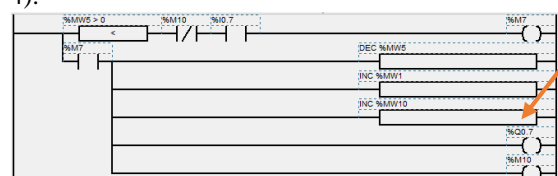


Figura 8 - Sub-rotina do CLP com alteração do controle supervisor condição de chegada de *Pcr1*.

A inclusão de memória para registrar o incremento de *Pcr1*, na Figura 8, apresenta a terceira e alteração na programação do CLP.

5. Discussões

5.1 Por que não realizar todo o controle no próprio robô?

Fundamentalmente por conta da elevada complexidade para realizar a implementação e da dificuldade de alteração das rotinas de programação, somado às limitações de memórias e de processamento dos variados modelos de robôs encontrados.

Todavia, a maioria dos robôs articulados verticais podem ser acionados via CLP. Assim, uma operação complexa pode ser simplificada de modo que o robô não precise tratar dos conflitos de chamadas de sub-rotinas, varreduras e demais, representando ganhos de energia elétrica, processamento, memória e aumento de vida útil do mesmo. Desta forma, um CLP trata as entradas da FMS e as envia ao robô como ordem de operação.

5.2 Qual a garantia de replicabilidade?

Desde que a RPC seja modelada corretamente, torna-se fácil a conversão do mesmo no algoritmo implementado no CLP, como demonstrado na Seção 4. Caberá, entretanto, ao programador atentar para a inclusão de memórias no CLP as quais representam as cores das fichas no lugar “*Pcr1*” e, na sequência, incluir estas memórias, como condição “and”, na linha de habilitação das transições que estavam livres, ou seja, sem restrição. Obviamente, um algoritmo que use lógica de acionamento “ou” usará a condição “or” em lugar da “and” descrita neste trabalho.

6. Conclusão

Neste trabalho é apresentada uma metodologia de conversão de modelos RPC para LD com e sem supervisão. Para a supervisão foi usado o método RCCD. Mostrou-se a praticidade da conversão, facilitando o trabalho do programador em LD. E, também que com poucas alterações num programa de CLP, pode-se alcançar o controle das operações de uma FMS robotizada sem comprometer as rotinas de programação já existentes no CLP, e, essencialmente garantindo o alcance do controle das operações.

A vantagem desta metodologia sobre as metodologias anteriores citadas são o uso de RPC enquanto as demais usam somente RP lugar-transição. Uma outra vantagem é que podemos converter um modelo não controlado e após incluir o controle com pequenas modificações no LD.

Atualmente estamos trabalhando na automatização desta metodologia.

Referências bibliográficas

Barghash, M. A., Abuzeid, O. M., Al-Rabadi, A. N., & Jaradat, A. M. (2011). Petri Nets and Ladder Logic for Fully-Automating and Programmable. *American J. of Engineering and Applied Sciences*, 252-264.

Barkaoui, K., Chaoui, A., & Zouari, B. (1997). Supervisory control of discrete event systems based on structure theory of petri nets. *IEEE International Conference on Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation*.

Cury, J. (2001). Teoria de controle supervisório de sistemas a eventos discretos.

Gomaa, M. M. (2011). Petri net to ladder logic diagram converter and a batch process simulation. *ARNP Journal of Engineering and Applied Sciences*.

Hopcroft, J., Motwani, R., & Ullman, J. (2006). Introduction to Automata Theory Languages and Computation.

Jensen, K., & Kristensen, L. M. (2009). Coloured Petri Nets. Em *Modelling and Validation of Concurrent Systems* (p. 8). Aarhus: University of Aarhus.

Menezes, F. A., Barroso, G. C., & Prata, B. d. (2012). Restrições de controle sobre cores decompostas: uma proposta no controle supervisório de sistemas a eventos discretos utilizando redes de petri coloridas. *Revista Controle & Automação/Vol.23 no.3/Maio e Junho*.

Moody, J., & Antsaklis, P. (1998). Supervisory Control of Discrete Event Systems Using Petri net., *Kluwer Academic Publishers*.

Morais, C. C., & Castrucci, P. d. (2007). Engenharia de automação industrial. Rio de Janeiro: LTC.

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

Prata, B. A., Barroso, G. C., & Arruda, J. B. (2008). Um novo método para controle supervisório de sistemas a eventos discretos baseado em redes de petri coloridas. *XV Simpsio Brasileiro de Pesquisa Operacional*.

Ramadge, P. J., & Wonham, W. M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization* 25(1): 206–230.

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

ANEXO A – CÓDIGO PRINCIPAL DO CONVERSOR RPC PARA LLD

CÓDIGO PRINCIPAL

```

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import core.ladder.Comparator;
import core.ladder.Memory;
import core.ladder.Block;
import core.ladder.Out;
import core.ladder.Switch;
import core.petri.ElementosCPN;
import core.petri.entity.Place;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.Color;
import java.util.ArrayList;
import java.util.Iterator;

import gui.util.WrapLayout;

public class GPrincipal extends JFrame {
    private ElementosCPN cpn;
    private ArrayList<Block> listaDeBlocosFonte;
    private ArrayList<Block> listaDeBlocos;
    private JPanel panel;
    private int contadorIO;
    public GPrincipal() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(200, 120, 854, 763);
        cpn = new ElementosCPN();
        init(); }
    private void init() {
        getContentPane().setLayout(new BorderLayout(0, 0));
        panel = new JPanel();
        panel.setBackground(Color.white);
        panel.setSize(getWidth(), getHeight());
        WrapLayout tela = new WrapLayout();
        tela.setVgap(0);
        panel.setLayout(tela);
        Block initMem1 = new Block(panel);
        panel.add(initMem1);
        Switch ch1 = new Switch();
        ch1.setText("ON/OFF");
        initMem1.addChave(ch1);
        initMem1.addMemoria(new Memory("Ligado/Desligado"));
        panel.setLayout(tela);
        Block initMem = new Block(panel);
        panel.add(initMem);
        Switch ch = new Switch();
        ch.setText("%S 13");
        initMem.addChave(ch);
        for (Place p : cpn.getPlaces()) {

```

```

        for (int cor : p.getFaixa()) {
            initMem.addMemoria(new Memory("M_" + p.getText() + "_" +
cor + " := " + p.getFichasDeCorX(cor + "")));}}
        listaDeBlocosFonte = new ArrayList<Block>();
        listaDeBlocos = new ArrayList<Block>();
        Block block = null;
        contadorIO = 1;
        int k1 = 1;
        for (int i = 0; i < cpn.getTransitions().size(); i++) {
            if (cpn.getListaLugaresPreTran(i).size() == 0) {
                addBlockFonte(cpn.getListaLugaresPostTran(i));}}
        for (int i = 0; i < cpn.getTransitions().size(); i++) {
            if (cpn.getListaLugaresPreTran(i).size() > 0) {
                ArrayList<ArrayList<Integer>> lista =
gerarListaCombinacoes(cpn.getListaLugaresPreTran(i));
                int tamanho = 1;
                for (Place pl : cpn.getListaLugaresPreTran(i)) {tamanho *=
pl.getFaixa().size();}
                int soma;
                for (int index = 0; index < lista.size(); index++) {
                    soma = 0;
                    block = new Block(panel);
                    ch = new Switch();
                    ch.setText("Ligado");
                    block.addChave(ch);
                    for (int s : lista.get(index)) {
                        soma += s; }
                    for (Place p :
cpn.getListaLugaresPostTran(i)) {block.addMemoria(new Memory("INC " + "M_" +
p.getText()+ "_" + p.getFaixa().get(index%p.getFaixa().size())));}
                    for (int cor = 0 ; cor < lista.get(index).size();
cor++) {block.addMemoria(new Memory("DEC " + "M_" +
cpn.getListaLugaresPreTran(i).get(cor).getText() + "_" +
lista.get(index).get(cor)));}
                    block.addComparador(new Comparator("M_" +
cpn.getListaLugaresPreTran(i).get(cor).getText() + "_" + lista.get(index).get(cor)
+ " > " + "0"));}
                    Out a = new Out("%Q0. " + k1++);
                    block.addMemoria(a);
                    boolean existeTranFonte = false;
                    for (Block bFonte : listaDeBlocosFonte) {
                        for (Memory memFonte : bFonte.getMemorias()) {
                            for (Memory mem : block.getMemorias()) {
                                if
(memFonte.getLabText().equals(mem.getLabText())) {
                                    block.remChave(1);block.addChave(new
Switch(bFonte.getChaves().get(1).getLabText()));
                                    existeTranFonte = true;}}}}
                    if (!existeTranFonte) {ch = new Switch();
                    ch.setText("%I0." + contadorIO++);
                    block.addChave(ch);
                    }listaDeBlocos.add(block);
                    for (int in = 0; in <
cpn.getListaLugaresPostTran(i).size(); in++) { if
(cpn.getListaLugaresPostTran(i).get(in).getFaixa().size() < tamanho)
{JOptionPane.showMessageDialog(null, "Erro na faixa de saída");
                    }}}}
                for (Block b : listaDeBlocosFonte) {panel.add(b);}

```

```

        for (Block b : listaDeBlocos) {panel.add(b); }
        JScrollPane jsp = new JScrollPane(panel);
        jsp.setVerticalScrollBar().setUnitIncrement(10);
        getContentPane().add(jsp);}
    private ArrayList<ArrayList<Integer>> gerarListaCombinacoes(
        ArrayList<Place> listaLugaresPreTran) {int tamanho = 1;
        ArrayList<ArrayList<Integer>> res = new
ArrayList<ArrayList<Integer>>(); ArrayList<Integer> ea;
        for (Place pl : listaLugaresPreTran) { tamanho *=
pl.getFaixa().size();}
        for (int m = 0; m < tamanho; m++) {ea = new ArrayList<Integer>();
            for (int l = 0 ; l < listaLugaresPreTran.size(); l++)
{ea.add(0); }res.add(ea); }
        int contador = 0;
        for (int q = 0; q < listaLugaresPreTran.size(); q++) {
            contador = 0;
            while (contador < tamanho) {
                for (int f : listaLugaresPreTran.get(q).getFaixa()) {
                    for (int w = 0; w <
Math.pow(tamanho/listaLugaresPreTran.get(q).getFaixa().size(),
listaLugaresPreTran.size()-1-q); w++) {
                        res.get(contador).set(q, f);
                        contador++;
                    }}}}
            return res; }
    public void addBlockFonte(ArrayList<Place> lista) {int soma;
        Block block = null;
        Switch ch = null;
        for (Place p : lista) {for (int val = 0; val < p.getFaixa().size();
val++) {
            soma = 0;
            block = new Block(panel);
            ch = new Switch();
            ch.setText("Ligado");
            block.addChave(ch);
            block.addMemoria(new Memory("INC " + "M_" + p.getText()+ "_" +
p.getFaixa().get(val)));
            ch = new Switch();
            ch.setText("%I0." + contadorIO++);
            block.addChave(ch);
            listaDeBlocosFonte.add(block);
        }}
    public static void main(String[] args) {
        new GPrincipal().setVisible(true);
    }
}

```