Franck Carlos Vélez Benito

Desdobramento para Redes de Petri k-limitadas

Curitiba - PR

30/08/2010

Franck Carlos Vélez Benito

Desdobramento para Redes de Petri k-limitadas

Dissertação apresentada como requisito parcial à obteção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador:
Luis Allan Künzle

Co-orientador:
Fabiano Silva

UNIVERSIDADE FEDERAL DO PARANÁ - UFPR

Curitiba - PR

30/08/2010

Agradecimentos

Agradeço primeiramente a Deus por me dar forças e determinação nos momentos difíceis.

A minha mãe Sra. Sabina Pillco Benito que sempre acreditou em mim, incentivando-me a cumprir meus objetivos.

A minha esposa e eterna enamorada Diana Cristina Prochnow Vélez pela paciência, atenção, amor e carinho em todos os momentos.

A minhas tias Juana e Clorinda pela apoio e moradia durante todo o primeiro ano do mestrado.

A meu orientador professor Luis Allan Künzle pelos conselhos e orientações que fizeram possível a realizaçao deste trabalho.

Ao professor Fabiano Silva pelos conselhos sempre pertinentes que atuou como meu co-orientador.

Aos Professores Marcos Sunye e Bruno Müller pela oportunidade de trabalhar no projeto GHI.

Aos meus amigos e colegas do projeto GHI, com os quais passei momentos agradáveis.

Aos professores do Dinf pelo grande conhecimento passado para mim durante o mestrado.

E a todos aqueles que contribuiram direta ou indiretamente na realização deste trabalho.

Sumário

Lista de Figuras

Resumo

Um dos problemas chave dos sistemas autômatos é o problema de alcançabilidade. A resolução deste mediante o grafo de alcançabilidade gera, sobretudo em sistemas do mundo real, o problema de explosão de estados. McMillan [12] propôs uma técnica chamada de *unfolding* – desdobramento – que gera uma nova rede, de complexidade menor que a do grafo de alcançabilidade, que contém o conjunto de estados alcançáveis, o que permite evitar a explosão de estados de sistemas modelados com redes de Petri.

Esta técnica tem várias implementações, a maioria limitada para redes de Petri seguras, sendo que no contexto dos sistemas do mundo real, geralmente trabalha-se com um número limitado de recursos, frequentemente superior a uma unidade. Por esta razão, é importante dispor-se de uma implementação da técnica de desdobramento, mas para redes de Petri k-limitadas, que permitem modelar sistemas com um número limitado de recursos.

Neste trabalho serão apresentados, além de conceitos importantes de redes de Petri e do processo de desdobramento, uma proposta de desdobramento para redes de Petri k-limitadas. Para a implementação foi escolhida uma das ferramentas de mais destaque na técnica de desdobramento. Após um estudo aprofundado desta ferramenta, ela foi modificada de forma a incorporar o desdobramento de redes k-limitadas.

A proposta e a implementação foram validadas a partir de um estudo de caso. São apresentados e discutidos os resultados obtidos, as limitações da proposta e possíveis trabalhos futuros neste campo de pesquisa.

Palavras chave: Redes de Petri, Redes de Petri k-limitadas, desdobramento.

Abstract

One of the key problems of automated systems is the reachability problem. The solution of this through the reachability graph, especially in real-world systems, generates the state explosion problem. McMillan [12] proposed a technique called unfolding which generates a new network of smaller complexity than the reachability graph, which contains the set of states reachable, thus preventing the explosion of states of systems modeled with networks Petri.

This technique has several implementations, the mostly limited for safe petri nets, being that in the context of real-world systems, typically works with a limited number of resources, often more than one unit. Therefore, it is important to have an implementation of the technique of unfolding, but for k-bounded Petri nets, which allow to model systems with limited resources.

This work presents, beyond important concepts of Petri nets and the unfolding process, a proposal of unfolding for k-bounded Petri nets. For the implementation was chosen one of the most prominent tools in the unfolding technique. After a detailed study of this tool, it was modified of way to incorporate the k-bounded nets unfolding.

The proposal and implementation has been validated from a case study. Are presented and discussed the results, the limitations of the proposal and possible future work in this field of research.

Keywords: Petri nets, k-bounded Petri nets, unfolding.

1 Introdução

O problema de alcançabilidade é um dos problemas chave dos sistemas autômatos. A resolução deste por meio do grafo de alcançabilidade gera o problema de explosão de estados. McMillan [13] propôs uma técnica chamada de *unfolding* – desdobramento – para auxiliar no problema de explosão de estados de sistemas modelados com redes de Petri finitas. O desdobramento de uma rede é outra rede finita e acíclica que preserva as propriedades da rede original.

O algoritmo do processo de desdobramento proposto inicialmente por McMillan sofreu algumas alterações. Primeiramente foi melhorado por J. Esparza [4] deixando o algoritmo melhor estruturado, diminuindo o excesso de chamadas de funções que o procedimento impunha e agrupando estas funções em uma única. Em seguida, Komenkho [8] introduziu a noção de *slices* e paralelismo para o processo de desdobramento, tornando-o mais rápido.

Com base nestas modificações, foram desenvolvidas algumas ferramentas para o processo de desdobramento. Uma destas ferramentas é o Mole [14] que utiliza o algoritmo de J. Esparza, a outra é o Punf [17] fundamentada nas modificações feitas por Komenkho. Ambas ferramentas realizam o processo de desdobramento para redes de Petri seguras.

O presente trabalho tem como objetivo apresentar uma proposta que amplia o escopo de aplicação do algoritmo de desdobramento para redes de Petri não seguras. Esta ideia não é inédita, uma vez que o próprio McMillan menciona em seu texto que o processo de desdobramento criado por ele não é limitado a redes seguras e que uma marcação com multimarcas em um determinado lugar não impossibilita a execução do processo de desdobramento. Entretanto, não há nenhuma implementação de desdobramento para redes de Petri k-limitadas e os artigos que tratam do tema não apresentam exemplos de tal processo, apesar de citarem isto como possível.

A proposta consiste em utilizar as marcações não seguras e fazer com que estas habilitem uma determinada transição a quantidade de vezes que esta possa ser disparada. Tendo em consideração que a marcação resultante do disparo de uma transição varia dependendo do

número de disparos, então cada possibilidade de disparar uma mesma transição, com um disparo ou com diferente número de disparos simultâneos, será considerada como uma transição distinta.

A presente dissertação está organizada da seguinte forma: no capítulo 2 é feita uma revisão bibliográfica dos conceitos mais relevantes sobre redes de Petri e o problema de alcançabilidade, necessários para a compreensão do trabalho. O capítulo 3 conceitua e descreve o processo de desdobramento, incluindo alguns conceitos importantes para o entendimento deste, além da apresentação do algoritmo de desdobramento desenvolvido por J. Esparza. No capítulo 4 é apresentada uma análise detalhada da ferramenta Mole, mais importante implementação de desdobramento já realizada, identificando as estruturas que esta utiliza e seu funcionamento. O capítulo 5 apresenta a nova abordagem para o processo de desdobramento de redes de Petri k-limitadas, as modificações feitas na ferramenta mole para adequá-la à nova abordagem e um estudo de caso com a utilização da nova aboradagem para verificar se os resultados obtidos são satisfatórios. Finalmente, no capítulo 6 são apresentadas as conclusões e perspectivas de trabalhos futuros.

2 *Redes de Petri*

Neste capítulo são apresentados os principais conceitos concernentes a redes de Petri (RdP), como por exemplo: representação gráfica, formalismo matemático e algumas classes e propriedades importantes para o desenvolvimento de nosso trabalho. Por último abordamos os problemas de alcançabilidade e a consequente explosão de estados que são um dos focos deste trabalho.

2.1 Conceitos Base

As redes de Petri (RdP) foram propostas por Carl Adam Petri, na sua tese de doutorado, submetida em 1962 [16], na qual apresentou um tipo de grafo bipartido com estados associados, com o objetivo de estudar a comunicação entre autômatos. Atualmente as redes de Petri são utilizadas para modelar sistemas dinâmicos (paralelos, concorrentes, assíncronos e não-determinísticos) tendo como fundamento uma forte base matemática.

A análise de uma RdP pode revelar características importantes do sistema modelado com relação a sua estrutura e/ou seu comportamento dinâmico, podendo assim modificá-lo ou melhorá-lo.

A representação gráfica de uma RdP consiste de um grafo bipartido, ponderado e dirigido contendo dois tipos de nós, chamados de lugares e transições, conectados por segmentos orientados chamados de arcos. Cada um destes arcos pode ter como rótulo um número, sendo este o peso do arco. Os lugares podem conter uma ou mais marcas [3] ,[18].

- **Lugares**: Representam condições, predicados, recursos ou uma descrição lógica de um estado do sistema. Cada lugar pode conter um número não-negativo de marcas. São representados graficamente por um círculo.
- **Transições**: Representam eventos, ações que mudam o estado do sistema, cuja ocorrência depende dos estados do sistema. Transições removem ou adicionam marcas dos lugares.

São representadas graficamente por um retângulo.

- **Marcas**: São associadas aos lugares e representam a quantidade de recursos disponíveis nos estados do sistema. São representadas graficamente por pequenos círculos pretos.
- **Arcos**: Conectam lugares com transições e transições com lugares. Cada arco tem um peso, representado por um número inteiro, determinando o número de marcas que serão adicionadas ou removidas dos lugares; os arcos com peso igual a 0 não são desenhados, e os arcos que tenham peso igual a 1 não são rotulados. São representados graficamente por setas. A figura 2.1 mostra uma rede de Petri.

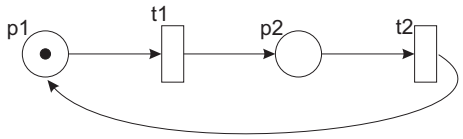

Figura 2.1: Rede de Petri.

Formalmente, uma rede é uma tripla $N = (P,T,F)$ onde [18]:

$P = \{p_1,p_2,...,p_k\}$ é um conjunto disjunto finito de lugares;

$T = \{t_1,t_2,...,t_k\}$ é um conjunto disjunto finito de transições;

F é um conjunto de arcos com um determinado peso, tal que:

$F \subseteq (P \times T) \cup (T \times P)$ (relação de fluxos) e

$F \rightarrow \mathbb{N}^+$ (função de peso).

A relação de Fluxo F é definida pelo pré-conjunto e pós-conjunto dos lugares e transições da rede. Para x ∈ N pode se definir:

${}^\bullet x = \{|y \mid (y,x) \in F|\}$, é o pré-conjunto e

$x^\bullet = \{|y \mid (x, y) \in F|\}$, é o pós-conjunto

Para $x \subseteq N$ usa-se:

${}^\bullet X := \bigcup_{x \in X} {}^\bullet x$

$X^\bullet := \bigcup_{x \in X} x^\bullet$

$\forall x, y \in N: x \subset {}^\bullet y \Leftrightarrow y \subset x^\bullet$

Para uma relação de fluxo F usamos:

- $\prec$ para o fechamento transitivo, se:

  (i) $(x,y) \in F \Rightarrow (x,y) \in \prec$;

  (ii) $((x,y) \in \prec) \wedge ((y,z) \in \prec) \Rightarrow (x,z) \in \prec$.

- $\preceq$ para o fechamento transitivo reflexivo, onde:

  (i) $\preceq = \prec \cup (x,x) \mid x \in F$

- **Marcação**: Uma marcação de N é um multiconjunto M dos lugares(P), isto é $M : P \rightarrow \mathbb{N} = \{1,2,3,...\}$, e o conjunto de todas as marcações de N serão denominadas por $\mathcal{M}(N)$ [10].

- **Sistema de Rede**: Um sistema de rede é um par $\mathcal{SN} = (N,M_0)$ que compreende uma rede finita $N = (P,T,F)$ e uma marcação inicial M_0 tal que $M_0 \in \mathcal{M}(N)$ [9].

2.2 Classes de Redes de Petri

Existem várias classes de redes de Petri, sendo duas mais importantes para o nosso estudo. Estas são apresentadas a seguir:

- **Sistema Condição/Evento**: Um sistema C/E é aquele que permite no máximo uma marca em cada lugar e o peso de todos os arcos é igual a 1. Formalmente um sistema C/E é uma dupla $\mathcal{SN}_{C/E} = (N,A_N)$ tal que:

  (i) $N = (B,E,F)$ onde os lugares são representados por condições (B) e as transições são representadas por eventos (E);

  (ii) A_N é o estado inicial, tal que $A_N \in \mathcal{M}(N)$;

  A figura 2.2 mostra um sistema C/E.

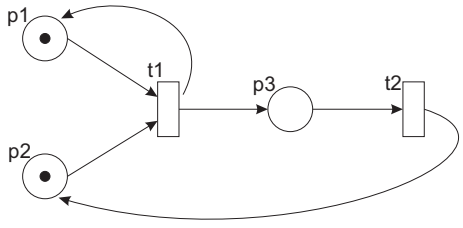

Figura 2.2: Sistema C/E.

- **Sistema Lugar/Transição**: Um sistema L/T é aquele que permite o acúmulo de marcas no mesmo lugar e o peso dos arcos é igual a *k* tal que $k \in \mathbb{N} = \{1,2,3,...\}$. Formalmente um sistema L/T é uma dupla $\mathcal{SN}_{L/T} = (N,M_0)$ tal que:

  (i) $N = (P,T,F)$;

  (ii) M_0: $P \rightarrow \mathbb{N}$ é uma marcação inicial, tal que $M_0 \in \mathcal{M}(N)$.

  A figura 2.3 mostra um sistema L/T.

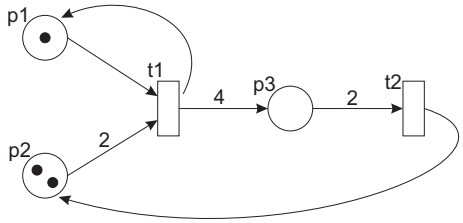

Figura 2.3: Sistema L/T.

2.3 Dinâmica da Rede

Uma transição pode ou não estar habilitada. Somente transições habilitadas podem "disparar". O disparo de uma transição habilitada muda o estado do sistema representado pela rede. Uma transição $t \in T$ está habilitada por uma marcação *M*, se e somente se $M \geqslant {}^\bullet t$, isto é, o número de marcas nos lugares de entrada de *t* é maior ou igual ao peso dos arcos que liga os lugares à *t* [1]. Se uma transição habilitada for disparada, será obtida uma nova marcação M' tal que $M' = M - {}^\bullet t + t^\bullet$ [3]. O disparo de uma transição *t*, habilitada por uma marcação *M* gera uma marcação M' que pode ser denotada por: $M\ [t\rangle\ M'$. Na figura 2.4 será apresentada a nova marcação obtida do disparo de *t1* da rede da figura 2.3.

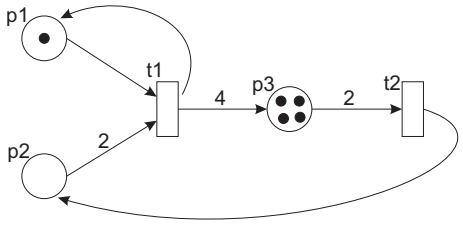

Figura 2.4: Disparo de um sistema.

2.4 Conflito e Concorrência

- **Conflito**: Dois nós (lugar ou transição), *x* e x', estão em conflito se existem transições distintas *t*, $t' \in T$, tal que ${}^\bullet t \cap {}^\bullet t' \neq 0$ e (t, x) e (t', x') são $\preceq$. Denotado por $x \# x'$ [9], [10] e

[8]. A figura 2.5 mostra o conflito entre as transiçoes *t1* e *t2*, sendo que ambas transições têm *p2* como um dos elementos do pré-conjunto.

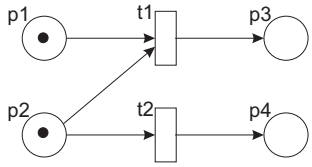

Figura 2.5: Conflito.

- **Concorrência**:Dois nós (lugar ou transição), x e $x^{'}$, são concorrentes se ${}^{\bullet}x \cap {}^{\bullet}x^{'} = 0$ e $\neg(x \preceq x^{'})$ e $\neg(x^{'} \preceq x)$. Denotado por $x\ co\ x^{'}$ [9], [10] e [8]. A figura 2.6 mostra a concorrência entre as transiçoes *t1* e *t2*.

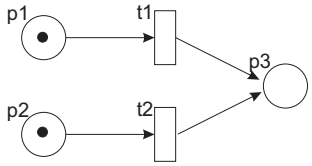

Figura 2.6: Concorrência.

2.5 Propriedades

As propriedades das redes de Petri podem ser classificadas em dois grupos: um é baseado na estrutura da rede e o outro no comportamento, que analisa a dinâmica da rede. Neste trabalho, focaremos a propriedade estrutural de aciclicidade e as propriedades comportamentais de limitabilidade e alcançabilidade.

- **Aciclicidade**: Uma rede é acíclica se a relação de fluxo é $\not\preceq$, isto é [15]:

$$x_1 \text{ F } x_2 \text{ F } .. \text{ F } x_n \Rightarrow x_1 \neq x_n.$$

- **Limitabilidade**: Um $\mathcal{SN}$ é limitado se para cada marcação alcançável M e cada $p \in P$, $M(p)$ é limitado. A rede é dita k-limitada se $M(p) \leqslant k$; é dita segura se é 1-limitada. Sendo que $k \in \mathbb{N}$ [3].

- **Alcançabilidade**: O conjunto de marcações alcançáveis $\mathcal{RM}$ de $\mathcal{SN}$ é o menor conjunto em que: $M_0 \subset \mathcal{RM}(\mathcal{SN})$ e tal que se $M \in \mathcal{RM}(\mathcal{SN})$ e $M[t\rangle\ M^{'}$. Para algum $t \in T$ e $M^{'} \in \mathcal{M}(N)$, então $M^{'} \in \mathcal{RM}(\mathcal{SN})$. Para uma sequência finita de transições $\sigma = t_1 \ldots t_k$,

escrevemos $M[\sigma\rangle\ M'$ se existe uma marcação M_1 ... M_{k+1}, tal que $M_1 = M$, $M_{k+1} = M'$, e $M_i[t_i\rangle\ M_{i+1}$, para algum i = 1, ..., k [3].

O conjunto de marcações alcançáveis finito pode ser representado graficamente por meio de um grafo, denominado grafo de alcançabilidade. Neste, os nós são as marcações e os arcos são as transições disparadas que geram uma marcação alcançável.

A figura 2.7 apresenta o grafo de alcançabilidade do sistema da figura 1.3.

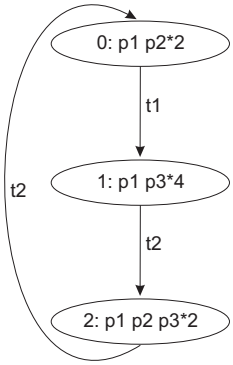

Figura 2.7: Grafo de alcançabilidade.

2.6 Equação Fundamental

A equação fundamental é definida por:

$$M' = M_0 + \mathrm{C}.\ \omega$$

Onde:

C é a matriz de incidência que é definida pela diferença do pós-conjunto e pré-conjunto C = $x^{\bullet}$ - $^{\bullet}x$. A seguir é exemplificada a matriz de incidência do sistema da figura 2.3.

$$Pre = \begin{array}{c} \begin{matrix} t_1 & t_2 \end{matrix} \\ \begin{bmatrix} 1 & 0 \\ 2 & 0 \\ 0 & 2 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix} \qquad Pos = \begin{array}{c} \begin{matrix} t_1 & t_2 \end{matrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 4 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix} \qquad C = \begin{array}{c} \begin{matrix} t_1 & t_2 \end{matrix} \\ \begin{bmatrix} 0 & 0 \\ -2 & 1 \\ 4 & -2 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}$$

ω é o vetor de parikh [3] que descreve a sequência de disparos de transições, isto é, seja $\sigma = t_1...t_k$ uma sequência de disparos de transições então $\omega = (\#_{t_1}\sigma, ..., \#_{t_k}\sigma)$ sendo que $\#t_i$ é o número de ocorrências de t_i em t_1 ... t_k.

A equação fundamental permite a análise da acessibilidade das marcações e a representação dos aspectos comportamentais da rede, já que esta descreve a dinâmica da inserção e remoção de marcas nos lugares e a sequência de disparos de transições.

2.7 Problema de Alcançabilidade

A equação fundamental pode ser utilizada para determinar uma sequência de transições σ, tal que $M_0[\sigma\rangle M$. Porém, a existência de um vetor que atenda a equação não é uma condição necessária para que a marcação M seja realmente alcançada a partir da marcação inicial, uma vez que a ordem de disparo é perdida e a solução encontrada pode trazer vetores σ que não correspondem a sequências possíveis de disparos na rede.

O problema de alcançabilidade pode ser definido como o problema de verificar se uma dada marcação M é alcançável a partir da marcação M_0, ou seja, se:

$$M \in \mathcal{RM}(\mathcal{SN})$$

É preciso destacar que muitas vezes essa exata marcação que está sendo buscada não será encontrada e sim uma marcação maior que irá contê-la. Chamamos M_σ a marcação que está contida em uma marcação M ($M_\sigma \subseteq M$), ou ainda, que $M_\sigma \leqslant M$. Temos então o problema de alcançabilidade de sub-marcação, definido como o problema de verificar se existe um M tal que:

$$M \in \mathcal{RM}(\mathcal{SN}) \text{ sendo que } M_\sigma \subseteq M$$

O problema de alcançabilidade de sub-marcação é teoricamente equivalente ao problema de alcançabilidade, que se sabe ser um problema decidível usando espaço exponencial [11]. Em relação à complexidade computacional, sabe-se que para resolver alcançabilidade em redes de Petri acíclicas é NP-Completo [21] e, em redes limitadas ou k-limitadas, é PSPACE-Completo [20].

Desta forma, o grafo de alcançabilidade pode ser utilizado para resolver problemas de alcançabilidade, envolvendo métodos de busca e técnicas heurísticas, mas apenas para redes pequenas devido à explosão do espaço de estados.

3 *Desdobramento*

Desdobramento é um método introduzido por McMillan [13] para evitar o problema de explosão de estados mediante a criação de uma estrutura que preserva todos os lugares alcançáveis da rede original.

Este processo é feito mediante o mapeamento de uma rede de Petri Lugar-Transição para uma rede de ocorrência levando em consideração alguns fundamentos teóricos para a criação e delimitação desta, formando assim o desdobramento da rede.

Neste capítulo serão apresentadas definições relacionadas ao processo do desdobramento, o processo de ramificação, configurações e cortes e o algoritmo de desdobramento com algumas ferramentas criadas para o desenvolvimento deste.

3.1 Definições Relacionadas

- **Rede de Ocorrência:** Uma rede Condição/Evento $N_{E/C} = \{B,E,F\}$ é uma rede de ocorrência ON se:

  (i) $\forall b \in B, |{}^\bullet b| \leqslant 1$;

  (ii) ON é acíclica;

  (iii) $\forall x \in ON$, o conjunto $\{y \mid y \prec x\}$ é finito;

  (iv) Para cada $y \in (B \cup E)$, $\neg(y\#y)$;

  (v) denota-se $Min(ON)$ o conjunto de elementos mínimos de $B \cup E$ com respeito a uma relação causal, isto é, os elementos que tem um pré-conjunto vazio.

- **Homomorfismo:** É uma aplicação que preserva uma estrutura dada. Sejam $(D,\cdot)$ e $(E,*)$ dois grupos e seja h uma função de D em E. Diz-se que h é um homomorfismo se:

$$(\forall x,y \in D)\colon h(x \cdot y) = h(x) * h(y)$$

3.2 Processo de Ramificação

Um processo de ramificação de um $\mathcal{SN}$ é uma dupla $\pi = (ON, h)$ onde: ON é uma rede de ocorrência e h é um homomorfismo de uma rede de ocorrência ON = (B,E,F) para um sistema de rede $\mathcal{SN}$ ($h : B \cup E \rightarrow P \cup T$) tal que [19]:

(i) $h(B) \subseteq P$ e $h(E) \subseteq T$ (condições são mapeadas para lugares, e eventos para transições);

(ii) Para cada $e \in E$, $h\{|^\bullet e|\} = {}^\bullet h(e)$ e $h\{|e^\bullet|\} = h(e)^\bullet$ (preserva o ambiente das transições);

(iii) $h\{|Min(ON)|\} = M_0$ (o conjunto de condições mínimas correspondem a marcação inicial);

(iv) $\forall e, e' \in E$, se ${}^\bullet e = {}^\bullet e'$ e $h(e) = h(e')$ então $e = e'$(Não existe redundância nas transições).

O processo de ramificação pode ser interminável, uma vez que ele pode dar início a vários processos de ramificação, sendo estes chamados de prefixos [4]. Um processo de ramificação $\pi' = (ON', h')$ de $\mathcal{SN}$, é um prefixo do processo de ramificação $\pi = (ON, h)$; denota-se $\pi' \sqsubseteq \pi$, se $ON' = (B', E', F')$ é um subconjunto de $ON = (B, E, F)$ tal que:

(i) $Min(ON) \in ON'$;

(ii) Se $e \in E'$ e $(b, e) \in F$ ou $(e, b) \in F$ então $b \in B'$;

(iii) Se $b \in B'$ e $(e, b) \in F$ então $e \in E'$;

(iv) h' é uma restrição de h para $B' \cup E'$.

A figura 3.1 mostra um exemplo de um sistema de rede segura e dois processos de ramificação, onde o homomorfismo h é indicado pelos rótulos nos nós. O processo na figura 3.1(b) é um prefixo da figura 3.1(c).

Para cada $\mathcal{SN}$ existe um único processo de ramificação máximo, que possui todas as marcações alcançáveis e preserva a concorrência e conflito do $\mathcal{SN}$; este processo de ramificação é chamado de prefixo finito completo e representa o desdobramento da rede. Para uma melhor definição deste, necessitaremos de alguns conceitos sobre configuração e corte.

3.3 Configuração e Corte

Uma configuração C representa uma possível execução parcial da rede, isto é, um conjunto de transições que satisfazem as seguintes condições([12]):

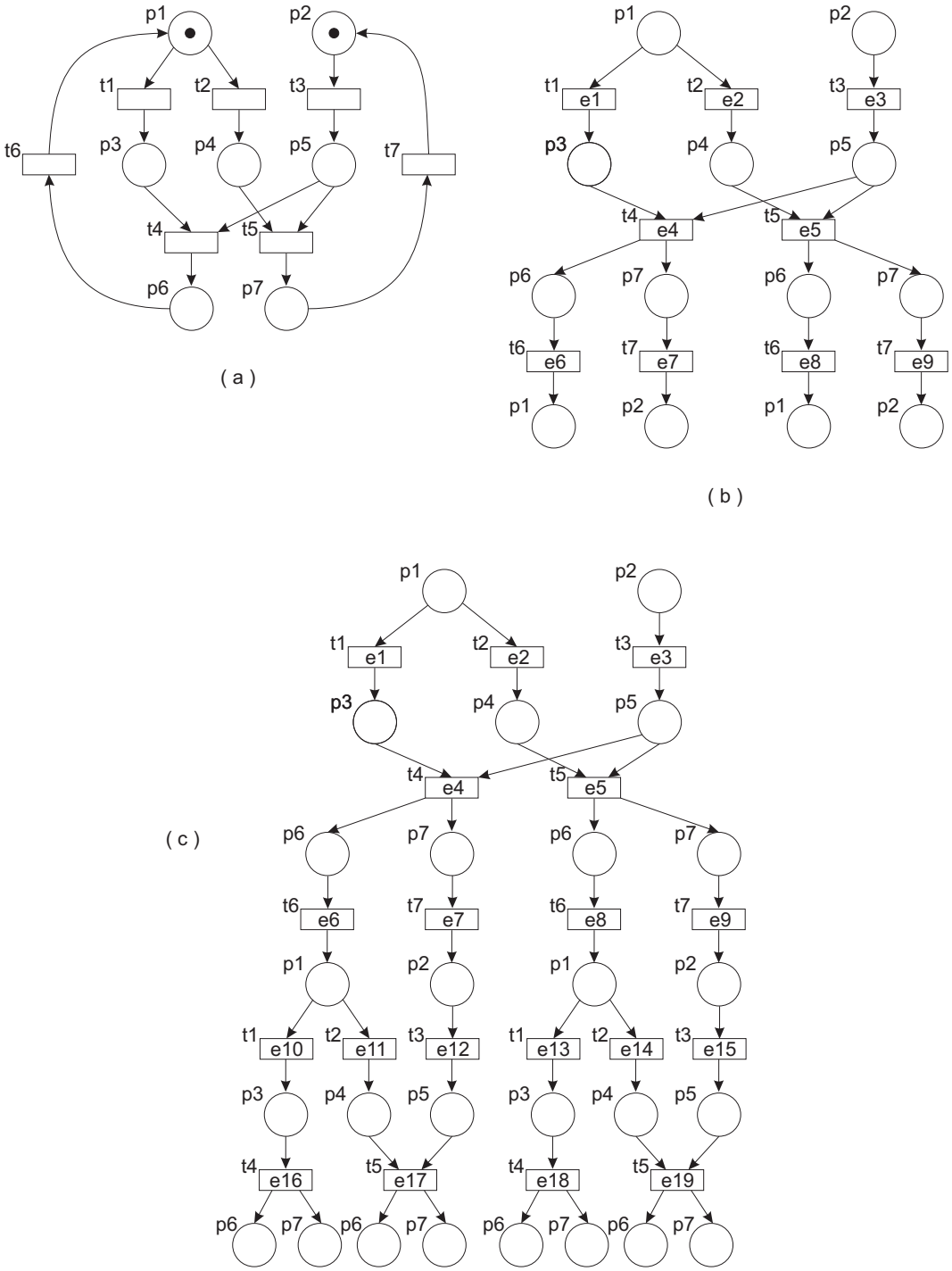

Figura 3.1: Um sistema de rede (a) e dois processos de ramificação (b,c).

(i) $\forall e' \preceq e, e \in C \Rightarrow e' \in C$;

(ii) $\forall e , e' \in C$: $\neg(e \# e')$.

Na figura 3.1(b) podemos visualizar o conjunto de eventos {*e1,e3,e4,e6*}, que formam uma configuração.

Para cada evento $e \in E$, a configuração $[e] = \{e' \mid e' \preceq e\}$ é chamada de configuração local de *e*, e $\langle e \rangle = [e] \setminus \{e\}$ denota o conjunto de predecessores causais de *e*.

Na figura 3.1(b,c) podemos observar a configuração local do evento [*e9*] = {*e2,e3,e5,e9*}.

Um conjunto de condições B' tal que para todos $b, b' \in B'$ distintos, $bcob'$, é chamado de co-conjunto. Um corte é um co-conjunto máximo. Cada marcação alcançável de *Min*(*ON*) é um corte.

Uma configuração pode ser associada com uma marcação Mark(*C*) que corresponde a uma marcação alcançável a partir de M_0 após todas as transições de *C* terem sido disparadas. $\text{Mark}(C) = h((Min(ON) \cup C^{\bullet}) \setminus {}^{\bullet}C)$.

Na figura 3.1 a sequencia de eventos {*e2,e3,e5,e9*} é uma configuração e a marcação alcançável por esta configuração é *p2* e *p6*.

3.4 Algoritmo

A construção do prefixo finito completo é feita seguindo o algoritmo melhorado por Esparza, Römer e Vogler [4], também chamado de algoritmo ERV Unfolding apresentado na figura 3.2.

O algoritmo ERV Unfolding apresentado neste capítulo foi utilizado como base para a implementação de diversas ferramentas que geram o desdobramento de redes de Petri. Uma dessas ferramentas, denominada *Mole*, foi desenvolvida por Stefan Römer e Stefan Schwoon [14]. Outra que podemos destacar é a ferramenta *PUnf*, desenvolvida por Khomenko [17], que é uma versão melhorada para a realização do processo de desdobramento, adicionando conceitos de *slice* e paralelismo.

```
   Input: Um SN = {N,M0}, sendo que M0 = {p1 , . . . , pk}
   Output: Unfolding Unf de SN
1  Unf ← lugares de M0;
2  pe ← transições habilitadas por M0;
3  cut-off ← Ø;
4  while pe ≠ Ø do
5  |  escolha um evento e = (t, X) de pe tal que [e] seja mínimo;
6  |  if [e] ∩ cut-off = Ø then
7  |  |  adicione e e novas instancias dos lugares de h(e) em Unf;
8  |  |  pe ← PE(Unf) {Atualiza as transições habilitadas};
9  |  |  if e é um evento de corte then
10 |  |  |  cut-off ← cut-off ∪ e;
11 |  |  end
12 |  else
13 |  |  pe ← pe \ {e}
14 |  end
15 end
```

Figura 3.2: Algoritmo ERV

4 Análise da Ferramenta Mole

Mole é uma ferramenta que serve para gerar o desdobramento de redes de Petri L/T 1-limitada, desenvolvida por Stefan Romer e Stefan Schwoon e que tem como fundamento o algoritmo ERV Unfolding. Está disponibilizada sob licença pública geral (GNU GPL) [5].

Esta ferramenta foi construída para ser compatível com o ambiente do projeto PEP (Programming Environment based on Petri Nets)[1], mantido pelo grupo de Sistemas Paralelos do Departamento de Ciência da Computação da Universidade de Oldenburg na Alemanha.

Neste capítulo é apresentada a análise da Ferramenta *Mole*, descrevendo a estrutura do arquivo de entrada, as estruturas de dados utilizadas pela ferramenta e o processo da criação do desdobramento.

A descrição da ferramenta *Mole*, realizada a seguir, será exemplificada com a rede de Petri da figura 4.1.

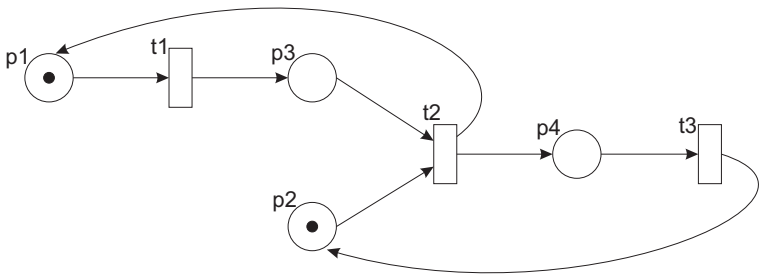

Figura 4.1: Rede exemplo.

4.1 Estrutura do Arquivo de Entrada

A entrada do programa é um arquivo com a descrição da rede no formato próprio do ambiente e a saída um arquivo com o desdobramento da rede. A ferramenta também dispõe de um aplicativo que lê o arquivo de saída, e imprime os dados contidos neste, em um formato possível

[1] http://theoretica.informatik.uni-oldenburg.de/ pep/

para gerar um gráfico, utilizando a ferramenta Dot [2].

O arquivo de entrada que descreve a rede deve conter em ordem os seguintes elementos:

1. Cabeçalho;
2. Lista dos lugares;
3. Lista das transições;
4. Lista dos arcos de transições para lugar;
5. Lista dos arcos de lugar para transições.

O cabeçalho deve iniciar com a palavra-chave PEP. A segunda linha deve especificar o tipo de rede, sendo que PTNet que é a abreviação para uma rede de Petri Lugar/Transição.
A terceira e última linha do cabeçalho conterá a palavra-chave que define o formato da rede utilizada, neste caso FORMAT_N.

```
PEP
PTNet
FORMAT_N
%Comentários podem ser feitos somente depois do cabeçalho
PL
%lista dos lugares em ordem crescente
TR
%lista das transições em ordem crescente
TP
%lista dos arcos de transição para lugar
PT
%lista dos arcos de lugar para transição
```

As listas de lugares, transições, arcos de transição para lugar e arcos de lugar para transição devem vir logo abaixo do cabeçalho, indicados pelas respectivas palavras-chave PL, TR, TP e PT. Todas devem estar presentes no arquivo mesmo no caso de a lista a ser descrita estar vazia.

Por ser um formato de arquivo desenvolvido para um ambiente gráfico, a posição dos lugares e das transições precisa ser especificada através de coordenadas num@num, sendo que num

[2] http://www.graphviz.org/

é um número inteiro, positivo ou negativo. É preciso especificar também um nome e um identificador único para os lugares e transições. Para os lugares é permitido definir a marcação *M* num m num, em que *M* é relativo à marcação inicial, m à marcação corrente e num ao número de marcas.

Para especificar os arcos, deve-se seguir a sequência identificador_da_transição < identificador_do_lugar para os arcos de transição para lugar (TP) e a sequência identificador_ do_lugar > identificador_da_transição para os arcos de lugar para transição (PT). A extensão do arquivo de entrada será .ll_net.

A figura 4.2 apresenta o arquivo de entrada da rede de Petri da figura 4.1.

```
PEP
PTNet
FORMAT_N
PL
1"p1"0@0M1m1
2"p2"0@0M1m1
3"p3"0@0M0m0
4"p4"0@0M0m0
TR
1"t1"0@0
2"t2"0@0
3"t3"0@0
TP
1<3
2<1
2<4
3<2
PT
1>1
2>2
3>2
4>3
```

Figura 4.2: Arquivo de entrada.

Para executar o *Mole* deve-se digitar o comando ./mole <nome_do_arquivo>.ll_net no shell. Além da execução normal, o *Mole* tem também outras opções como:

- -T <nome_transição> que serve para parar o processo de geração do desdobramento na transição indicada pela variável <nome_transição>;
- -d<profundidade> que serve para parar o processo de geração do desdobramento no nível de profundidade indicada pela variável <profundidade>;
- -i que oferece a opção de escolher a sequência da criação dos eventos no processo do desdobramento;
- -m<nome_arquivo> que serve para escolher o nome do arquivo onde será armazenado o

desdobramento da rede. Sendo que por padrão o nome do arquivo de saída é o mesmo que o arquivo de entrada, mas com a extensão .mci.

4.2 Estruturas de Dados

Para uma melhor análise do *Mole* serão mencionadas as principais estruturas e variáveis utilizadas, sendo algumas para a criação da estrutura de dados da rede de entrada (place_t, trant_t, net_t e nodelist_t) e da rede de ocorrência (cond_t, event_t, unf_t e nodelist_t), e outras para estruturas auxiliares que servem para o processo de desdobramento.

nodelist_t: Estrutura encarregada de fazer as conexões dos lugares com as transições e vice-versa (arcos), do homomorfismo e de outras conexões importantes para a criação do desdobramento. Está composta pelos seguintes campos:

- node: ponteiro para um lugar, transição, condição ou evento;
- next: ponteiro para outra estrutura *nodelist_t*, que serve para fazer conexões entre: lugares (que formam uma marcação, sendo sempre feita em ordem decrescente dos identificadores dos lugares); eventos (que formam a lista de eventos de corte); um lugar com duas ou mais transições, uma transição com dois ou mais lugares e uma condição com dois ou mais eventos (que demostram o paralelismo).

A figura 4.3 ilustra a estrutura nodelist_t.

Figura 4.3: Estrutura nodelist_t.

contingent_t: Armazena as estruturas encarregadas de fazer as conexões. Está composta pelos seguintes campos:

- nodes: vetor de 1024 estruturas *nodelist_t*, sendo estas estruturas utilizadas em forma decrescente pelo programa;
- next: ponteiro que aponta para outra estrutura *contigent_t*, se a quantidade de estruturas *nodelist_t* for totalmente utilizada será criada outra estrutura *contingent_t*.

place_t: Armazena as informações do lugar e está composta pelos seguintes campos:

- name: nome do lugar(ex: *p1*);
- num: identificador do lugar(ex: 1);
- marked: indica se o lugar possui ou não uma marca;
- conds: ponteiro que auxilia no homomorfismo do lugar, aponta para a condição ou condições derivadas deste lugar;
- preset: ponteiro para uma estrutura *nodelist_t*, que faz a conexão do lugar com o seu pré-conjunto (transição);
- postset: ponteiro para uma estrutura *nodelist_t*, que faz a conexão do lugar com o seu pós-conjunto (transição);
- next: ponteiro para outra estrutura *place_t*, serve para associar os lugares da rede.

A figura 4.4 ilustra a estrutura *place_t*.

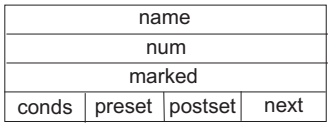

Figura 4.4: Estrutura place_t.

trans_t: Armazena as informações da transição e está composta pelos seguintes campos:

- name: nome da transição (ex: *t1*);
- num: identificador da transição (ex:1);
- preset_size e postset_size: quantidade de lugares do pré-conjunto e pós-conjunto da transição;
- preset: ponteiro para uma estrutura *nodelist_t*, que faz a conexão da transição com o seu pré-conjunto (lugar);
- postset: ponteiro para uma estrutura *nodelist_t*, que faz a conexão da transição com o seu pós-conjunto (lugar);
- next: ponteiro para outra estrutura *trans_t*; serve para associar as transições da rede.

A figura 4.5 ilustra a estrutura *trans_t*.

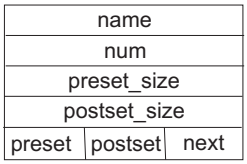

Figura 4.5: Estrutura trans_t.

net_t: Armazena as informações gerais da rede L/T do arquivo de entrada e está composta pelos seguintes campos:

- numpl e numtr: quantidade de lugares e transições;
- maxpre e maxpost: armazena o maior valor da quantidade de elementos dos pré-conjuntos e pós-conjuntos das transições;
- places: ponteiro para os lugares (*place_t*);
- transitions: ponteiro para as transições(*transt_t*).

A figura 4.6 ilustra a estrutura *net_t*.

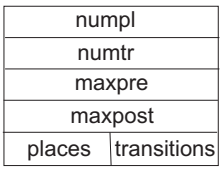

Figura 4.6: Estrutura net_t.

cond_t: Armazena as informações das condições e está composta pelos seguintes campos:

- mark: identificador da marcação que auxiliará no disparo das condições;
- num: identificador da condição;
- coarray_common: vetor que armazena as condições que estão em curso;
- coarray_private: vetor que armazena todas as condições que não pertencem ao conjunto de condições alcançáveis deste;
- origin: ponteiro para o homomorfismo da condição;
- preset: ponteiro para uma estrutura *event_t*, que faz a conexão da condição e seu pré-conjunto (evento);

- postset: ponteiro para uma estrutura *nodelist_t*, que faz a conexão da condição com seu pós-conjunto (evento);
- next: ponteiro para outra estrutura *cond_t*, serve para associar as condições.

A figura 4.7 ilustra a estrutura *cond_t*.

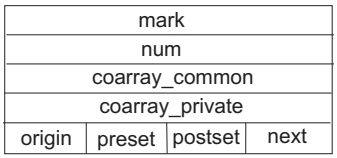

Figura 4.7: Estrutura cond_t.

event_t: Armazena as informações dos eventos e está composta pelos seguintes campos:

- mark: identificador da marcação que serve para ver se o evento está habilitado;
- coarray: ponteiro para as condições que não pertencem ao pré-conjunto do evento;
- id: identificador do evento;
- foata_level: quantidade de eventos da configuração do evento (|[evento]|);
- preset_size e postset_size: quantidade de condições do pré-conjunto e pós-conjunto do evento;
- origin: ponteiro para o homomorfismo do evento;
- preset: vetor de ponteiros que apontam para estruturas *cond_t*; faz a conexão do evento e seu pré-conjunto (condição);
- postset: vetor de ponteiros que apontam para estruturas *cond_t*; faz a conexão do evento e seu pós-conjunto (condição);
- next: ponteiro para outra estrutura *event_t*; serve para associar os eventos.

A figura 4.8 ilustra a estrutura *event_t*.

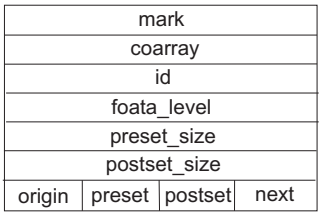

Figura 4.8: Estrutura event_t.

unf_t: Armazena as informações gerais da rede de ocorrência e está composta pelos seguintes campos:

- numco e numev: quantidade de condições e eventos;
- m0: ponteiro para uma estrutura *nodelist_t* que apontará para a M_0;
- conditions: ponteiro para as condições (*cond_t*);
- events: ponteiro para os eventos (*event_t*).

A figura 4.9 ilustra a estrutura unf_t.

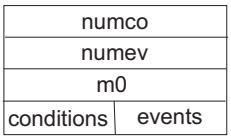

Figura 4.9: Estrutura unf_t.

events: É um vetor de ponteiros com 2000 posições em que cada posição aponta para uma estrutura *event_t*; serve para auxiliar o disparo de um evento.

hashcell_t: Armazena as informações de uma marcação e está composta pelos seguintes campos:

- marking: ponteiro que indica a marcação;
- event: ponteiro que indica o evento que gerou esta marcação;
- next: ponteiro para outra estrutura *hashcell_t*, serve para associar marcações com diferentes configurações.

A figura 4.10 ilustra a estrutura *hashcell_t*.

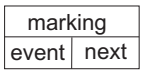

Figura 4.10: Estrutura hashccell_t.

hash: É um vetor de ponteiros onde cada posição aponta para uma estrutura *hashcell_t*, o número de posições é igual ao número de lugares da rede L/T multiplicado por quatro[3], podendo incrementar o número de posições se for necessário. Serve para auxiliar o evento de corte, armazenando as marcações da rede.

parikh_t: Armazena as informações de uma transição pertencente a uma configuração e está composta pelos seguintes campos:

- tr_num: identificador da transição;
- appearances: quantidade de vezes que esta transição foi disparada numa mesma configuração.

A figura 4.11 ilustra a estrutura *parikh_t*.

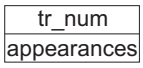

Figura 4.11: Estrutua parikh_t.

parikh: É um vetor de estruturas *parikh_t* cujo número de posições é igual ao número de transições da rede adicionado a dois[4], podendo incrementar o número de posições se for necessário. Serve para armazenar a configuração mínima do evento ($[e]$), sendo esta sempre em ordem crescente.

pe_queue_t: Armazena as informações das transições habilitadas por uma determinada marcação e está composto pelos seguintes campos:

- lc_size: quantidade de eventos da configuração do evento($|[e]|$);
- id: identificador;
- p_vector: aponta para o vetor de *parikh* da transição habilitada;

[3]Não é especificado a origem deste valor

[4]Não há justificativa para esse incremento de duas unidades no número de posições do vetor parikh_t

- trans: ponteiro que indica a transição;
- conds: vetor de ponteiros que indica as condições que habilitaram este evento;
- marking: ponteiro que indica a nova marcação alcançada pelo disparo da transição.

A figura 4.12 ilustra a estrutura *pe_queue_t*.

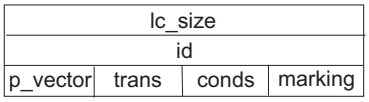

Figura 4.12: Estrutura pe_queue_t.

pe_queue: É um vetor de ponteiros com 1024 posições, que apontam para as estruturas *pe_queue_t*. Serve para armazenar as transições habilitadas.

pe_conds: É um vetor de ponteiros auxiliar, que armazenam as condições que são pré-conjunto de uma determinada transição.

colists: Vetor de ponteiros, que armazenam as configurações da rede.

ev_mark: Variável que auxilia no controle das condições e eventos habilitados.

pe_qsize: Variável que armazena a quantidade de transições habilitadas presentes na lista *pe_queue*.

cutoff: Variável que controla a identificação da existência de um evento de corte.

cutoff_list: Ponteiro para os eventos que produzem o cut-off, formando uma lista encadeada com estes.

4.3 Algoritmo

Para um melhor entendimento da ferramenta *Mole*, será sub-dividido o algoritmo E.R.V. em processos apresentado na figura 4.13, o qual servirá para relacionar as ações do algoritmo com os processos feitos pela ferramenta *Mole*.

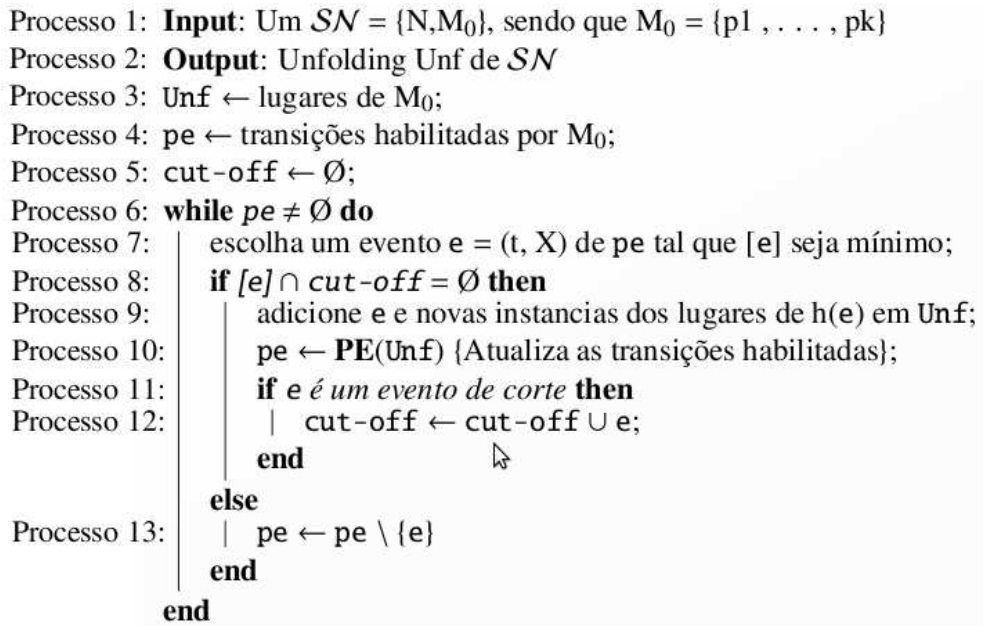

Figura 4.13: Algoritmo ERV sub-divido em processos.

O ferramenta *Mole* opera da mesma forma que o algoritmo E.R.V. como mostrado na figura 4.13, com algumas diferenças na ordem da execução de alguns processos. Alguns processos podem ser parcialmente executados, isto é, um processo pode começar e logo executar outro processo e depois terminar o processo anteriormente começado. A seguir serão mostrados e explicados detalhadamente os processos na ordem em que a ferramenta *Mole* os executa:

Processo 1: Neste processo será feita a leitura do arquivo de entrada verificando se este está sintaticamente correto com a estrutura anteriormente mencionada e criando estruturas para armazenar as informações deste. Isto é feito nos seguintes passos:

- Passo 1: Cria a estrutura *net_t* que será o nó principal;
- Passo 2: Lê sequencialmente todos os elementos da lista de lugares e, para cada lugar lido, cria estruturas *place_t*, armazenando o nome do lugar (ex: *p1*), o identificador do lugar, sendo este número atribuído de acordo com a criação da estrutura (ex: primeiro lugar criado recebe 1) e se este contém uma marca. Estas estruturas são encadeadas por meio do campo next formando uma lista encadeada. A última estrutura criada será apontada pelo campo places e a quantidade de estruturas *places_t* criadas serão armazenadas no campo numpl da estrutura *net_t*;
- Passo 3: Lê sequencialmente todos os elementos da lista de transições e, para cada transição lida, cria estruturas *trans_t*, armazenando o nome da transição (ex: *t1*), o identificador da transição, sendo este número atribuído de acordo com a criação da estrutura

(ex: primeira transição criada recebe 1). Estas estruturas são encadeadas por meio do campo next formando uma lista encadeada. A última estrutura criada será apontada pelo campo transitions e a quantidade de estruturas *trans_t* serão armazenadas no campo numtr da estrutura *net_t*;

- Passo 4: Cria a estrutura *contigent_t*;

- Passo 5: Lê sequencialmente todos os elementos da lista de arcos transição-lugar e, para cada arco lido, faz a conexão entre as estruturas da transição com o lugar, referentes a seus identificadores. Isto é feito por meio de duas estruturas *nodelist_t*: a primeira faz a conexão entre a transição e o lugar e a segunda entre o lugar e a transição. Desta maneira é armazenado o pós-conjunto da transição e o pré-conjunto do lugar;

- Passo 6: Lê sequencialmente todos os elementos da lista de arcos lugar-transição e, para cada arco lido, faz a conexão entre as estruturas do lugar com a transição, referentes a seus identificadores. Isto é feito por meio de duas estruturas *nodelist_t*: a primeira faz a conexão entre o lugar e a transição e a segunda entre a transição e o lugar. Desta maneira é armazenado o pós-conjunto do lugar e o pré-conjunto da transição;

- Passo 7: Analisa a estrutura de dados criada e obtém a quantidade de elementos do pré-conjunto e pós-conjunto de cada transição, armazenando estas informações na sua respectiva transição. Em seguida é armazenado o maior valor destas na estrutura *net_t*.

A figura 4.14 apresenta os passos 1, 2, 3 e 7

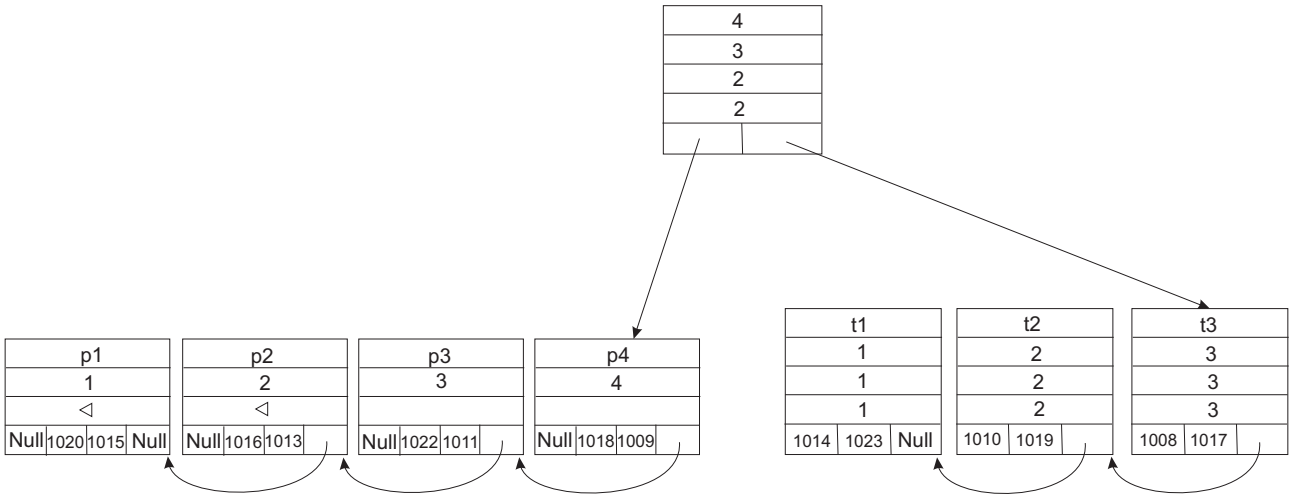

Figura 4.14: Representação dos lugares e transições na ferramenta Mole.

A figura 4.15 mostra os passos 4, 5 e 6.

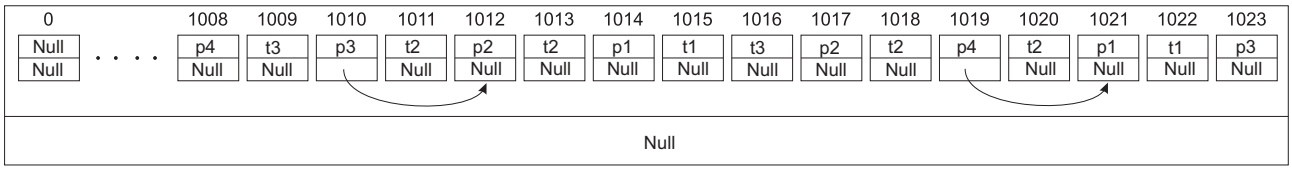

Figura 4.15: Representação dos arcos na ferramenta Mole.

A figura 4.16 mostra uma comparação da representação da rede no programa com a rede inserida.

Processos 3 e 5: A ferramenta *Mole* executa estes dois processos em conjunto sendo feito primeiramente a criação da estrutura principal da rede de ocorrência do desdobramento, na qual serão associadas as condições, e a inicialização do conjunto dos eventos de corte. Logo são identificados os lugares da marcação inicial da rede original e criadas as estruturas que iram armazenar os dados destas na rede de ocorrência. Estes 2 processos são decompostos em 5 sub-processos:

1. Criação das principais estruturas para a o processo de desdobramento e inicialização do cutoff

   - Passo 1: Cria a estrutura *unf* que será o nó principal da rede ocorrência;
   - Passo 2: Cria as estruturas *events* e *hash*;
   - Passo 3: O ponteiro *cutoff_list* recebe o valor null.

2. Identificação de M_0 na rede L/T

   - Passo 1: Percorre os lugares da rede L/T, buscando lugares marcados. Neste caso, o campo node de uma estrutura *nodelist_t* apontará para este lugar. Caso mais de um lugar estiver marcado o campo next apontará para o *nodelist_t* do lugar anterior. A conexão destes lugares dão a idéia de uma marcação, que neste caso é a M_0.

   A figura 4.17 apresenta o passo 1.

3. Inserção da M_0 na estrutura hash

   - Passo 1: Percorre as estruturas *nodelist_t* referentes à M_0. O algoritmo realiza um cálculo para obter a posição na estrutura hash, que apontará para a estrutura *hash-cell_t*; este cálculo é mostrado a seguir:

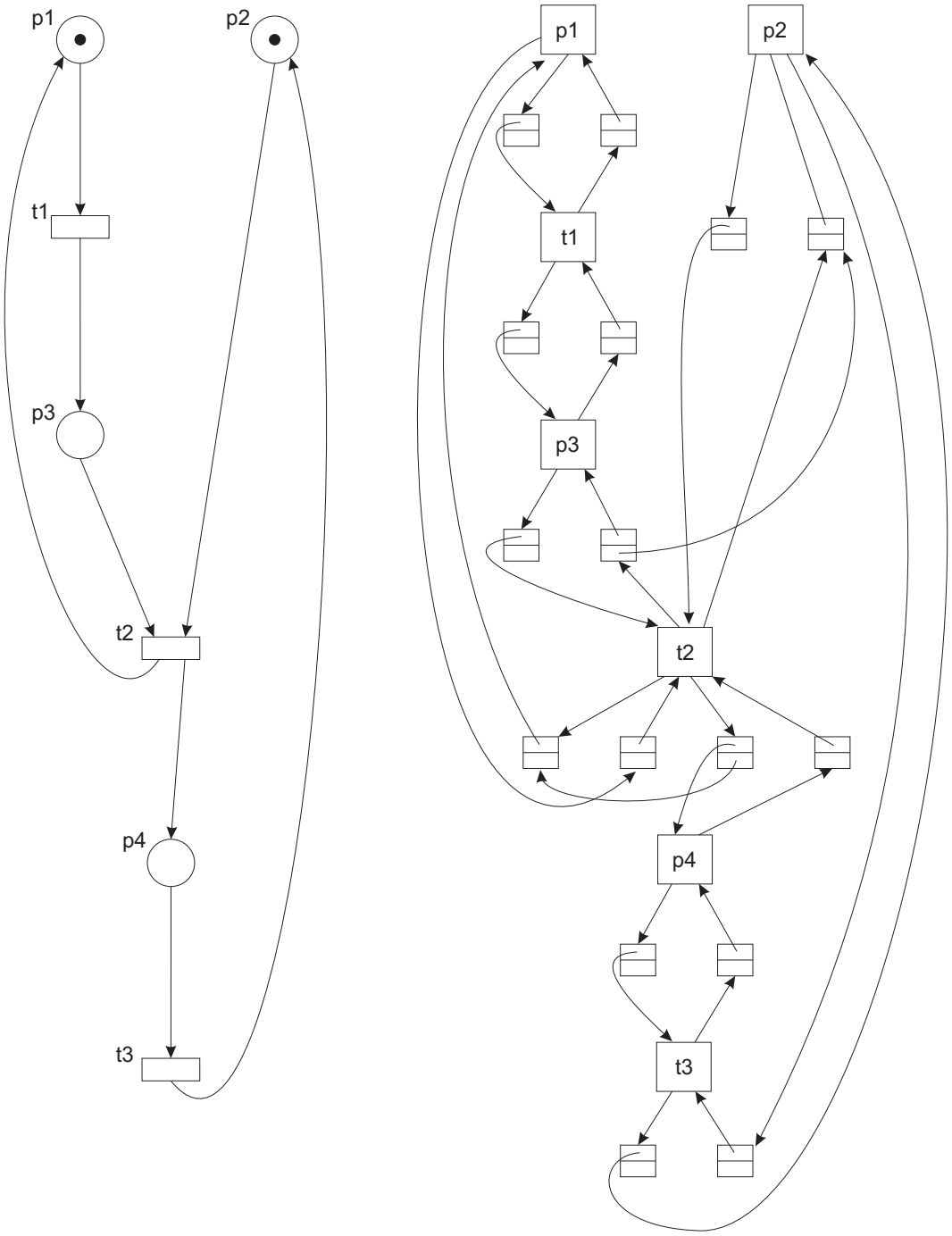

Figura 4.16: Rede de exemplo e sua representação na ferramenta Mole.

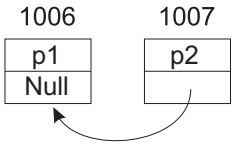

Figura 4.17: Representação da M_0 na ferramenta Mole.

$$\sum_{i=1}^{k} p_i * i.$$ sendo que: $p_i \in M$, $i \in \mathbb{N} = \{1,2,3,...\}$.e $k = |M|$

- Passo 2: Com a posição obtida do calculo anterior; cria-se uma estrutura *hashcell_t*, armazenando a marcação.

A figura 4.18 mostra os passos 1 e 2.

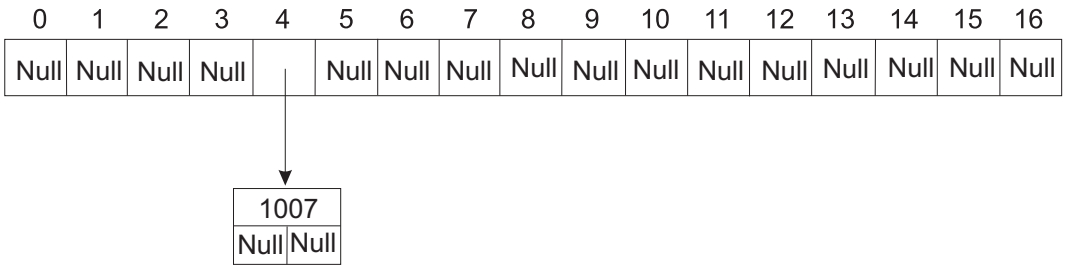

Figura 4.18: Estrutura Hash armazenando a M_0.

4. Criação de estruturas de auxílio

   - Passo 1: Cria as estruturas *pe_queue*, *pe_conds*, *pe_combs*, *pe0_conflicts*.
   - Passo 2: Cria a estrutura *parikh*.

5. Criação do homomorfismo de M_0 na rede ocorrência

   Neste sub-processo serão criadas as condições referentes a M_0; serão percorridas as respectivas estruturas *nodelist_t* e executados os seguintes processos em cada uma delas:

   - Passo 1: Cria a estrutura *cond_t* armazenando o identificador da condição (ex: 0), o campo origin aponta para o lugar concernente ao *nodelist_t* e o campo conditions da estrutura unf aponta para a condição criada. Se haver mais de uma condição, o campo next aponta para a condição que estiver no campo conditions e o campo conditions aponta para a nova condição, formando assim uma lista encadeada. Incrementa o valor no campo numco da estrutura *unf_t*, que representa o número de condições.
   - Passo 2: Cria os vetores *coarray_common* e *coarray_private* da estrutura *cond_t*.
   - Passo 3: Identifica a M_0 da rede de ocorrência (desdobramento), por meio de uma conexão entre o campo m0 da estrutura *unf_t* com a condição criada; isto é feito por uma estrutura *nodelist_t*. Se houver mais de uma condição, o campo next da estrutura *nodelist_t* apontará para o *nodelist_t* da condição anteriormente criada e o m0 apontará para a estrutura *nodelist_t* atual.

- Passo 4: Armazena todas as condições concorrentes no campo *coarray_private* da condição.

A figura 4.19 apresenta este sub-processo.

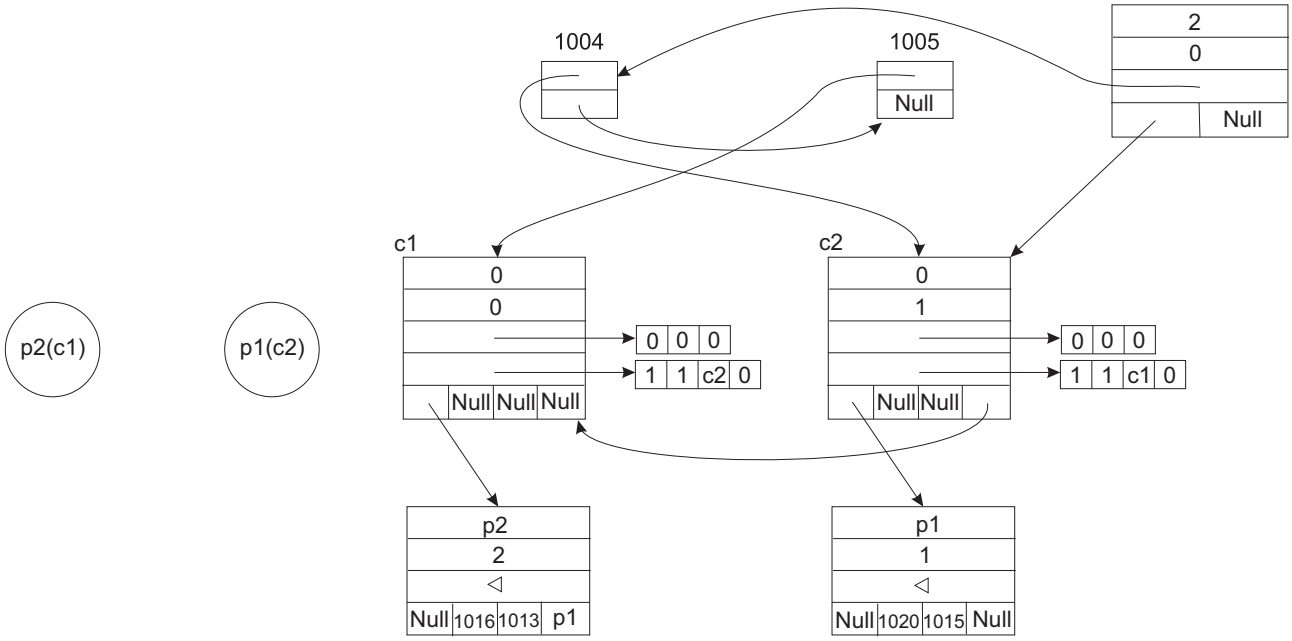

Figura 4.19: Representação das condições pertencentes a M_0 na ferramenta Mole.

Processo 4: Neste processo será feita a análise das condições pertencentes a marcação inicial e a criação das estruturas que armazenaram as transições habilitadas pelo homomorfismo destas condições. Este processo está decomposto em 2 sub-processos que são:

1. Análise das condições

   - Passo 1: Faz a conexão entre o campo conds da estrutura *place_t* (lugar) armazenado no campo origin da condição e a condição (homomorfismo), mediante uma estrutura *nodelist_t*;
   - Passo 2: Analisa as transições pertencentes ao pós-conjunto do lugar conectado à condição, verificando se os lugares correspondentes às condições armazenadas no vetor *coarray_common* e *coarray_private* da condição pertencem ao pré-conjunto da transição. Isto é feito para verificar se a transição está habilitada. Se estiver, analisa se há conflito. Caso exista conflito, analisa outra transição do pós-conjunto do lugar senão existir cria a estrutura para armazenar a transição habilitada.

2. Criação da estrutura para armazenar a transição habilitada

- Passo 1: Incrementa 1 na variável *ev_mark*;
- Passo 2: limpa as estruturas *parikh_t* da estrutura *parikh* e armazena o identificador da transição na estrutura *parikh_t*;
- Passo 3: armazena o valor da variável *ev_mark* nos campos *mark* das condições pertencentes ao pré-conjunto da transição. Identificando as condições pertencentes ao pré-conjunto da transição habilitada;
- Passo 4: Cria uma estrutura *pe_queue_t* para armazenar a transição habilitada. O campo *trans* aponta para a transição. O campo conds cria um vetor de ponteiros com o número de posições igual à quantidade de elementos do pré-conjunto e armazena dentro destas as condições pertencentes ao pré-conjunto. O campo *p_vector* armazena a estrutura parikh. O campo *lc_size* armazena o número de elementos do parikh;
- Passo 5: Incrementa 1 na variável *ev_mark*;
- Passo 6: Faz a conexão entre o campo *marking* da estrutura *pe_queue_t* com os lugares pertencentes ao pós-conjunto da transição e com os lugares das condições da marcação inicial que não foram consumidas por algum disparo. Isto é feito, mediante uma estrutura *nodelist_t*;
- Passo 7: A variável *pe_qsize* é incrementada em 1. A estrutura *pe_queue_t* criada anteriormente é inserida na estrutura *pe_queue* (lista de transições habilitadas) seguindo a ordem adequada. Em primeiro lugar são comparados os campos lc_size; se iguais, compara os valores da estrutura parikh_t da estrutura *parikh* um por um, se forem iguais compara o campo appearances e senão aquele que tenha o número de transição menor.

A figura 4.20 apresenta este sub-processo.

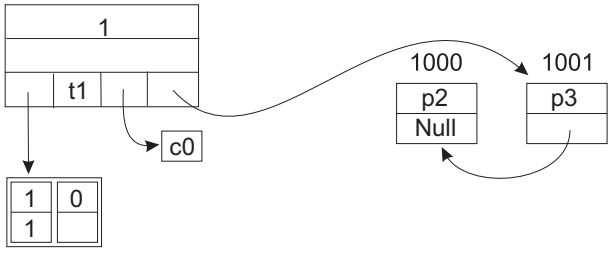

Figura 4.20: Representação de uma transição habilitada na ferramenta Mole - estrutura pe_queue_t.

Processo 6: Este processo é responsável pelo laço de repetição executando os seguintes 6 processos até que não existam elementos na lista de transições habilitadas. Isto é feito mediante avaliação da variável *pe_qsize* que armazena a quantidade de transições existentes na lista de transições habilitadas.

Processo 7: Neste processo é feita a escolha e remoção da transição habilitada, sendo esta a primeira da lista de transições habilitadas. Este processo é descrito nos seguintes passos:

- Passo 1: Sempre é escolhida a primeira *pe_queue_t* da estrutura *pe_queue*. Sendo esta retirada da lista, deslocando todas as estruturas *pe_queue_t*, uma posição a menos, preenchendo assim o campo vazio resultante. Verifica que as trasições habilitadas estejam ordenadas seguindo a ordem adequada.
- Passo 2: Decrementa a variável *pe_qsize* em 1.

Processo 9: Este processo é responsável pela criação do evento da transição escolhida, sendo feita nos seguintes passos:

- Passo 1: Com auxílio da estrutura *pe_queue_t* escolhida anteriormente, será criada uma estrutura *event_t*. Serão armazenados no campo origin a transição da *pe_queue_t*, no campo *preset_size* e *postset_size* o preset_size e postset_size da transição, no campo foata_level a |[e]|, no campo preset o conds da pe_queue_t;
- Passo 2: Faz a conexão das condições que estão armazenados no campo preset com o evento, mediante uma estrutura *nodelist_t*.

Processos 11 e 12: A ferramenta *Mole* executa estes dois processos em conjunto, sendo feita primeiramente a inserção da marcação gerada pelo disparo da transição anteriormente escolhida, dentro da estrutura hash. Logo é feito a verificação se esta marcação já existe, identificando assim um evento de corte.

- Passo 1: Percorre a marcação identificada no campo *mark* da estrutura *pe_queue_t*, que auxília na criação do evento anteriormente mencionado, também é feito o cálculo anteriormente mencionado para obter a posição na estrutura hash que apontará para a estrutura *hashcell_t*.
- Passo 2: Verifica se a posição obtida está vazia:

- Passo 2.1: Se estiver, cria uma estrutura *hashcell_t*, armazenando a marcação e o evento que gerou esta marcação;

- Passo 2.2: Se não estiver, compara os lugares da marcação contida nesta posição com os lugares da marcação que está sendo inserida:

- Passo 2.2.1: Se forem iguais as estruturas *nodelist_t* da marcação inserida serão reutilizadas para fazer outras conexões. O campo node de uma estrutura *nodelist_t* apontará para o evento da marcação inserida, a variável *cutoff_list* apontará para esta estrutura *nodelist_t* e o campo next apontará para outra estrutura *nodelist_t*, caso tenha mais um evento de corte. Desta maneira são identificados os eventos de corte;

- Passo 2.2.2: Se não forem iguais é criada uma estrutura *hashcell_t*, armazenando a marcação, o evento que gerou esta marcação caso exista e o campo next apontará para a estrutura hashcell_t que estava contida na posição.

A figura 4.21 apresenta este sub-processo.

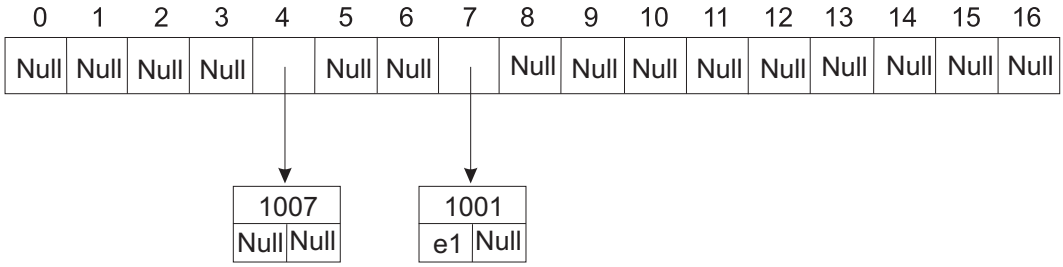

Figura 4.21: Estrutura Hash armazenando uma marcação.

Processos 8 e 13: Nestes processos é feita a análise da variável cutoff, sendo que se for igual a 0 então o campo *events* da estrutura *unf* aponta para a seguinte estrutura *event_t*.

Processo 9: Neste processo é feita a criação das condições do pós-conjunto do evento criado anteriormente. Este processo será decomposto em 2 sub-processos , que são:

1. Correlação dos eventos e condições

    - Passo 1: Analisa os vetores *coarray_commom* e *coarray_private* do pré-conjunto do evento para identificar quais condições são concorrentes a este evento;
    - Passo 2: Armazena as condições concorrentes no campo coarray da estrutura *event_t*.

2. Criação do pós-conjunto do evento

   - Passo 1: Analisa o pós-conjunto da transição pertencente ao campo origin do evento e cria estruturas *condition_t* para armazenar os dados deste;
   - Passo 2: Copia o vetor coarray do evento para o campo *coarray_commom* das condições criadas e limpa esta do evento;
   - Passo 3: Armazena todas as condições que são concorrentes no campo *coarray_private* da condição.

A figura 4.22 apresenta o processo 9. A criação do evento e seu pós-conjunto.

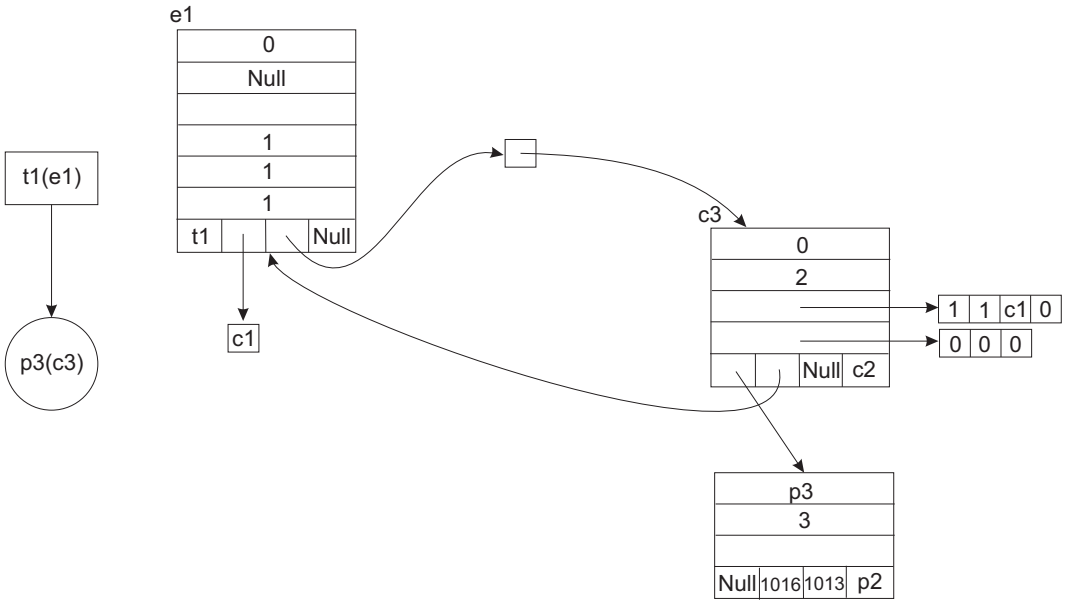

Figura 4.22: Representação de um evento e seu pós-conjunto na ferramenta Mole.

Processo 10: Neste processo será feita a análise das condições pertencentes a marcação gerada pelo disparo do homomorfismo do evento criado anteriormente e a criação das estruturas que armazenaram as transições habilitadas pelo homomorfismo destas condições. Este processo está decomposto em 2 sub-processos que são:

1. Análise das condições

   - Passo 1: Faz a conexão entre o campo conds da estrutura *place_t* (lugar) armazenado no campo origin da condição e a condição (homomorfismo), mediante uma estrutura *nodelist_t*;
   - Passo 2: Analisa as transições pertencentes ao pós-conjunto do lugar conectado à condição, verificando se os lugares correspondentes as condições armazenadas no

vetor *coarray_commom* e *coarray_private* da condição pertencem ao pré-conjunto da transição. Isto é feito para verificar se a transição está habilitada. Se estiver, analisa se tem conflito, caso exista conflito analisa outra transição do pós-conjunto do lugar.

2. Criação da estrutura para armazenar a transição habilitada

   - Passo 1: Incrementa 1 na variável *ev_mark*;
   - Passo 2: Limpa as estruturas *parikh_t* da estrutura *parikh* e armazena o identificador da transição na estrutura *parikh_t*;
   - Passo 3: Armazena o valor da variável *ev_mark* nos campos *mark* das condições pertencentes ao pré-conjunto da transição. Analisa os pré-eventos da condição, caso existam. Se o campo *mark* for diferente da variável *ev_mark*, então este receberá o valor da variável *ev_mark* e a transição do evento será inserida no *parikh_t*; esta inserção é feita em ordem crescente e o campo *mark* das condições do pré-conjunto deste evento receberam a variável *ev_mark*;
   - Passo 4: Cria uma estrutura *pe_queue_t* para armazenar a transição habilitada. O campo *trans* aponta para a transição. O campo *conds* cria um vetor de ponteiros com o número de posições igual a quantidade de elementos do pré-conjunto e armazena dentro destas as condições pertencentes ao pré-conjunto. O campo *p_vector* armazena a estrutura parikh. O campo *lc_size* armazena o número de elementos do parikh. Logo, com ajuda do campo *mark* das condições, analisa que condições pertencem parcialmente a configuração local do homomorfismo da transição, isto é ${}^\bullet b \in [e] \wedge b^\bullet \notin [e]$, fazendo uma conexão entre o campo *marking* com o lugar respectivo à condição, depois é feito este processo de novo com pós-conjunto da transição e com as condições da marcação inicial que não foram consumidas por algum disparo;
   - Passo 5: Logo a variável *pe_qsize* é incrementada em 1. A estrutura *pe_queue_t* criada anteriormente é inserida na estrutura *pe_queue* (lista de transições habilitadas) em ordem crescente. Sendo comparado em primeiro lugar os campos *lc_size*. Se acontecer que são iguais, compara os valores da estrutura *parikh_t* da estrutura *parikh* um por um, se acontecer que forem iguais compara o campo appearances, e senão aquele que tenha o número de transição menor.

A figura 4.23 apresenta este sub-processo.

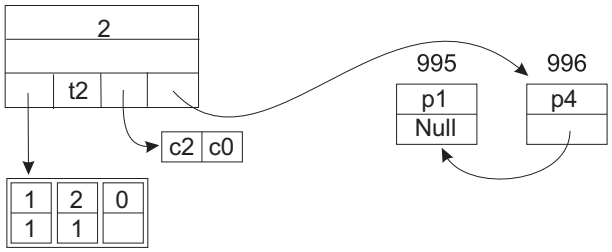

Figura 4.23: Representação de uma nova transição habilitada na ferramenta Mole - estrutura pe_queue_t.

Processo 9: Neste processo é feita a criação do pós-conjunto dos eventos de corte, percorrendo a lista de estruturas que a variável *cutoff_list* aponta.

Processo 2: Neste processo é feita a leitura da estrutura resultante do processo de desdobramento da rede original e armazenada no arquivo de saída. São armazenadas as condições e eventos com seus respectivos homomorfismos. A figura 4.24 mostra o unlfoding da rede gerado pelo programa e a representação estrutural do desdobramento da rede dentro do programa.

4.4 Conclusão

Com esta análise podemos concluir que o *Mole* é uma ferramenta muito eficiente e rápida, mesmo tendo que percorrer de forma recorrente a estrutura que armazena a rede original para identificar as transições habilitadas e os lugares pertencentes ao pós-conjunto da transição, além da necessidade de utilizar um grande conjunto estruturas para poder implementar o algoritmo de desdobramento.

A principal limitação da ferramenta *Mole* é sua restrição a redes de Petri 1-limitadas, não possibilitando o processo de desdobramento para redes de Petri k-limitadas. Esta restrição já é imposta na leitura da rede de entrada, uma vez que a execução do programa é concluída caso o número de marcas de qualquer lugar seja maior que 1. Mesmo que essa restrição na leitura seja suprimida, o processo de desdobramento é realizado pelo *Mole*, mas este desconsidera o número de marcas dos lugares, tratando estas marcações como seguras e desconsidera totalmente o peso dos arcos caso existam.

Um fator interessante é a leitura que o parser da ferramenta *Mole* realiza. Ela permite a leitura de vários tipos de descrição de rede de Petri, como por exemplo: peso nos arcos, temporização associada às transições, entre outros. Estas informações não são utilizadas na implementação da ferramenta, o que nos induz a pensar que futuras implementações superariam

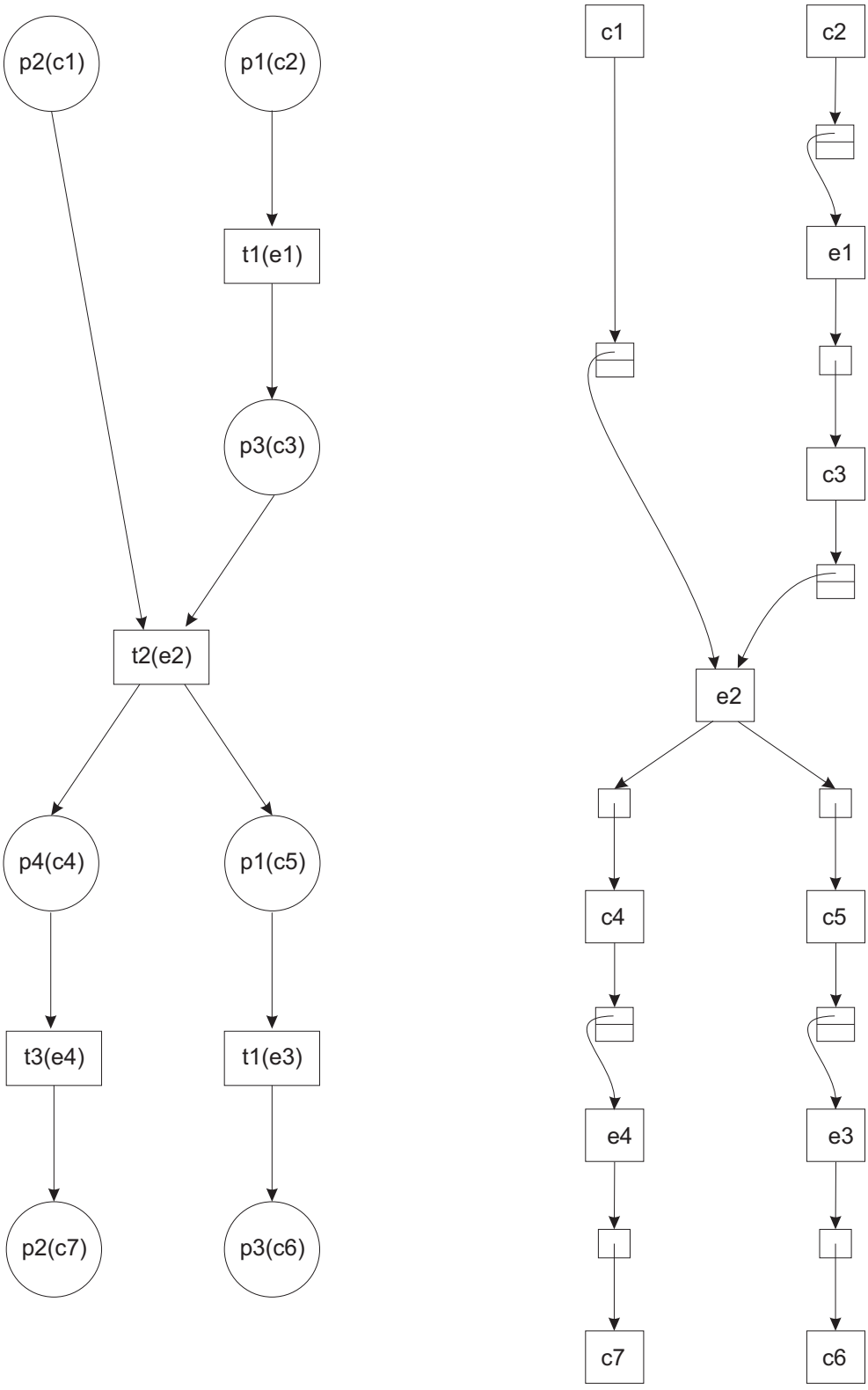

Figura 4.24: Desdobramento da rede exemplo e sua representação na ferramenta Mole.

as limitações atuais.

5 *Nova Abordagem*

Aplicações do mundo real trabalham com múltiplos recursos e, em geral, com um número limitado de recursos, mas frequentemente superior a uma unidade. Para tanto, consideramos que as redes de Petri k-limitadas são suficientes para representar sistemas com essas características. Sendo assim, o fato da ferramenta *Mole* trabalhar exclusivamente com redes de Petri seguras (1-limitadas), como descrito na seção 4.4, impossibilita sua utilização na análise da maior parte dos sistema do mundo real.

Por tal motivo, este trabalho tem como objetivo adaptar a ferramenta *Mole* para realizar o desdobramento de redes de Petri k-limitadas, mediante a aplicação de uma nova abordagem.

A nova abordagem consiste em utilizar as marcações não seguras que permitam mais de um disparo de uma mesma transição, isto é, $M \geqslant {}^\bullet t * 2$, fazendo com que estas transições possam ser disparadas mais de uma vez e tratando cada disparo múltiplo como se fosse uma transição diferente.

Neste capítulo serão apresentados a proposta para a construção do processo de desdobramento para redes de Petri k-limitadas, o algoritmo para esta proposta, as modificações feitas na ferramenta *Mole* para adequá-la a nova abordagem e por último um estudo de caso.

5.1 Proposta de Construção do Processo de Desdobramento para Redes de Petri k-limitadas

A proposta será detalhada através do exemplo a seguir, que consiste no sistema de rede apresentada na figura 5.1.

Primeiramente são inseridos os lugares da marcação inicial na rede de ocorrência. Em sequência, são identificadas as transições habilitadas por esta marcação, além do número de vezes que cada uma delas pode ser disparada. A lista de transições habilitadas recebe então todas as transições habilitadas para apenas um disparo. Para cada transição habilitada para *n*

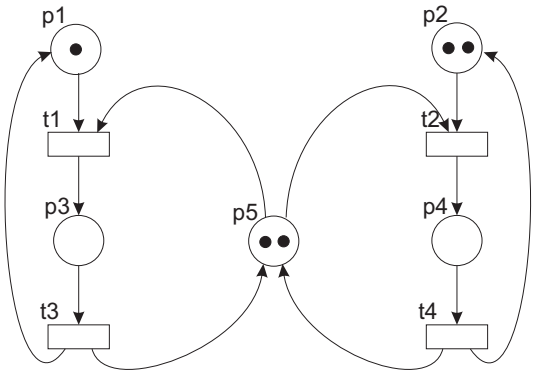

Figura 5.1: Rede exemplo para o processo Desdobramento.

múltiplos disparos (multi-habilitada), a lista recebe *n* cópias dessa transição. Cada uma das *n* cópias de uma mesma transição representa de 1 a *n* disparos simultâneos dessa transição. O processo, em seguida, escolhe uma transição da lista, gera as mudanças decorrentes de seu disparo e a insere na rede de ocorrência, assim como o pós-conjunto da transição. Uma possível consequência disso, em função do disparo de transições multi-habilitadas, é que um mesmo lugar, pertencente ao pós-conjunto da transição disparada, possua várias ocorrências. A efetiva marcação do lugar é a soma da marcação de todas as suas ocorrências.

As transições habilitadas pela marcação inicial da rede da figura 5.1 são *t1* e *t2*. a transição t1 pode ser disparada uma vez e será inserida na lista de transições habilitadas. A transição *t2* pode ser disparada duas vezes, o que implica que terá duas inserções na lista de transições habilitadas: uma considerando um disparo e outra considerando a ocorrência de dois disparos simultâneos, como apresentado na figura 5.2.

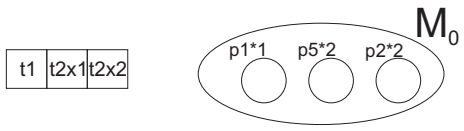

Figura 5.2: Processo de Desdobramento - fase 1.

Concluída a análise das transições habilitadas, uma das transições da lista é escolhida, seguindo a ordem adequada. No exemplo em questão é escolhida a transição *t1* com um disparo, que é portanto retirada da lista de transições. Em seguida, são adicionados na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo de *t1* a marcação *p3* com uma marca, *p5* com uma marca e *p2* com duas marcas, como mostrado na figura 5.3.

A nova marcação habilita as transições *t2* e *t3*, que por sua vez são analisadas. Ambas podem disparar apenas uma vez e são inseridas na lista de transições habilitadas. Após esta análise, é escolhida uma transição da lista, sendo esta a que tenha menor configuração local.

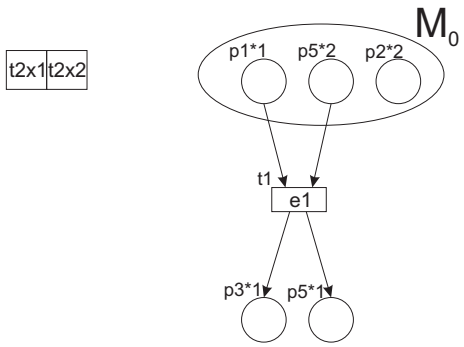

Figura 5.3: Processo de Desdobramento - fase 2.

A transição t2 será disparada e retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo de *t2* a marcação *p1*, *p2*, *p4* e *p5*, todas com uma marca, como apresentado na figura 5.4.

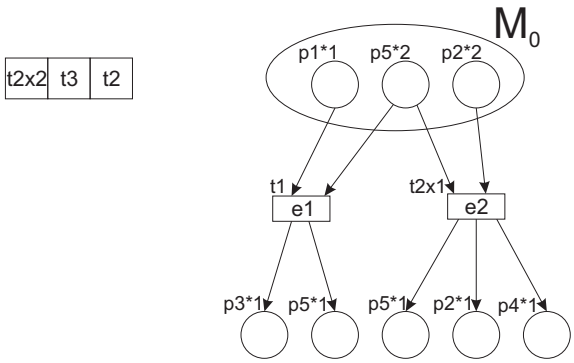

Figura 5.4: Processo de Desdobramento - fase 3.

Esta nova marcação habilita as transições *t1*, *t2* e *t4*. Elas são analisadas, sendo que todas só podem ser disparadas um vez, sendo inserindo na lista de transições habilitadas. Terminada a análise, novamente é escolhida uma transição da lista, sendo esta a que tenha menor configuração local. É escolhida a transição *t2* com dois disparos e retirada da lista de transições. Em seguida, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo duplo de *t2* a marcação *p1* com uma marca e *p4* duas marcas, como apresentado na figura 5.5.

A nova marcação habilita a transição *t4*. Sua análise mostra que ela pode ser disparada duas vezes. São então duas entradas na lista de transições habilitadas, uma com um disparo e outra com dois disparos. Em seguida, utilizando a menor configuração local, uma transição da lista é escolhida, no caso *t3*. Esta transição é então retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo de *t3* a seguinte marcação: *p1* com uma marca,

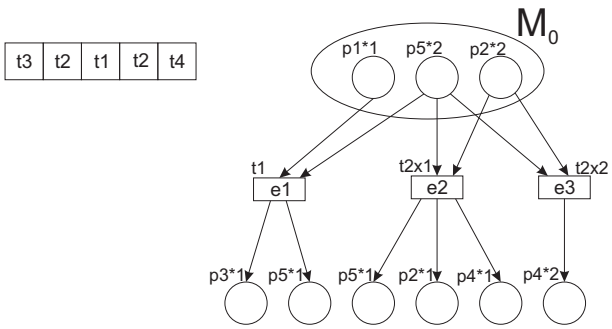

Figura 5.5: Processo de Desdobramento - fase 4.

p2 com duas marcas, *p5* com uma marca e *p5* com uma marca, como pode ser visto na figura 5.6.

Esta transição é um evento de corte. Como explicado anteriormente, quando um lugar aparece duas ou mais vezes em uma marcação, em função da ocorrência de disparos simultâneos, deve ser feita a soma das marcas das diversas "cópias" do mesmo lugar e o resultado desta soma é o número efetivo de marcas deste lugar. Portanto, na marcação anterior, o lugar *p5* possui efetivamente duas marcas.

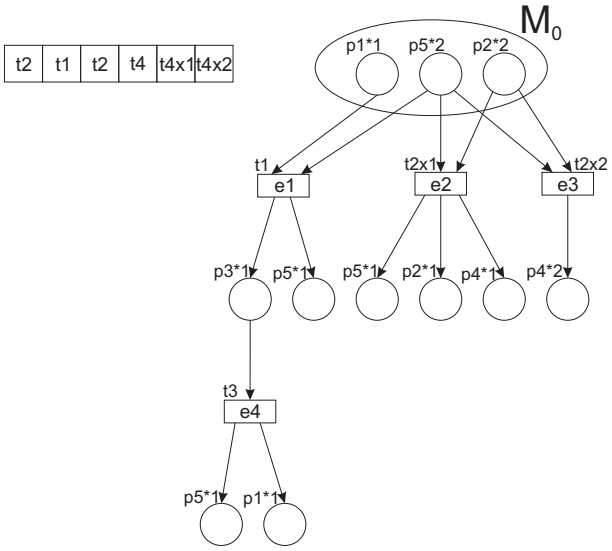

Figura 5.6: Processo de Desdobramento - fase 5.

Novamente, uma nova transição é escolhid, sendo esta a de menor configuração local. Neste momento, é escolhida a transição *t2* com um disparo e retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo de *t2* a marcação *p2*, *p3* e *p4* todas com uma marca, como apresentado na figura 5.7.

A nova marcação habilita a transição *t4*. Sendo esta analisada e podendo ser disparada uma

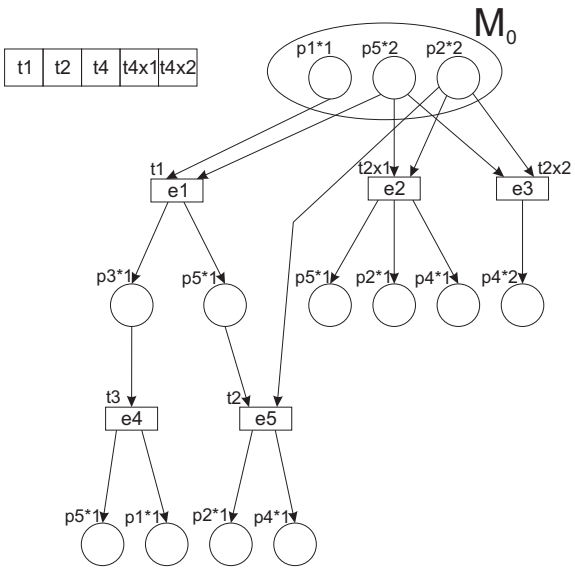

Figura 5.7: Processo de Desdobramento - fase 6.

vez, ela é inserida na lista de transições. Terminada a análise, é escolhida uma transição da lista utilizando a menor configuração local. É escolhida a transição t1 com um disparo e retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo de *t1* a marcação *p2*, *p3* e *p4* todas com uma marca, como apresentado na figura 5.8. Novamente, esta transição é um evento de corte.

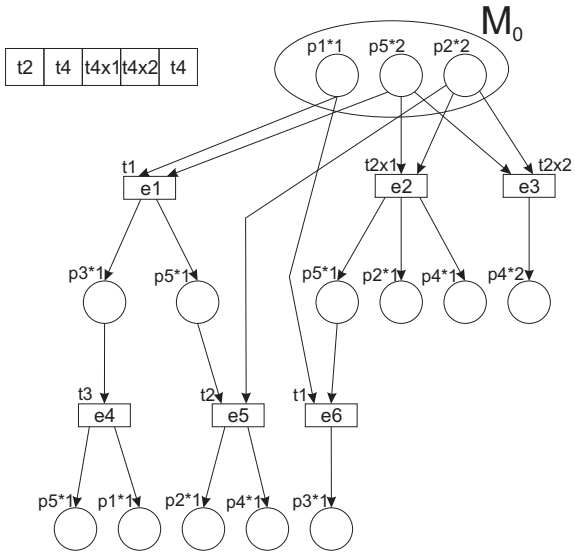

Figura 5.8: Processo de Desdobramento - fase 7.

Utilizando a menor configuração local, é escolhida a transição *t2* com um disparo e retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo de *t2* a marcação: *p1*

com uma marca, *p4* com uma marca e *p4* com uma marca, como apresentado na figura 5.9. A marcação efetiva de *p4* são duas marcas. Esta transição é um evento de corte.

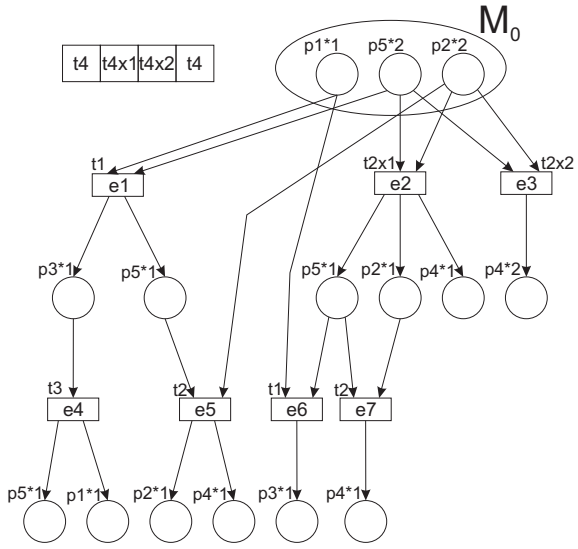

Figura 5.9: Processo de Desdobramento - fase 8.

Utilizando a menor configuração local, é escolhida a transição *t4* com um disparo e retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo de *t4* a marcação: *p1* com uma marca, *p2* com uma marca, *p2* com uma marca, *p5* com uma marca e *p5* com uma marca, como apresentado na figura 5.10. Esta transição é também um evento de corte.

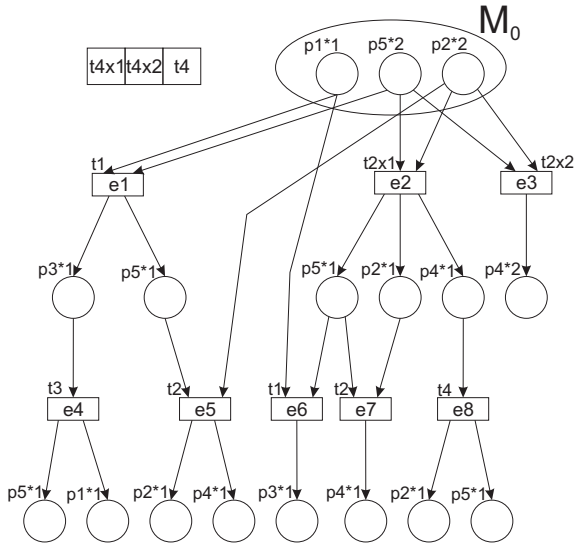

Figura 5.10: Processo de Desdobramento - fase 9.

Utilizando a menor configuração local, é escolhida a transição *t4* com um disparo e retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo de *t4* a marcação: *p1*,

p2, *p4* e *p5* todas com uma marca, como apresentado na figura 5.11. A transição *t4* é um evento de corte.

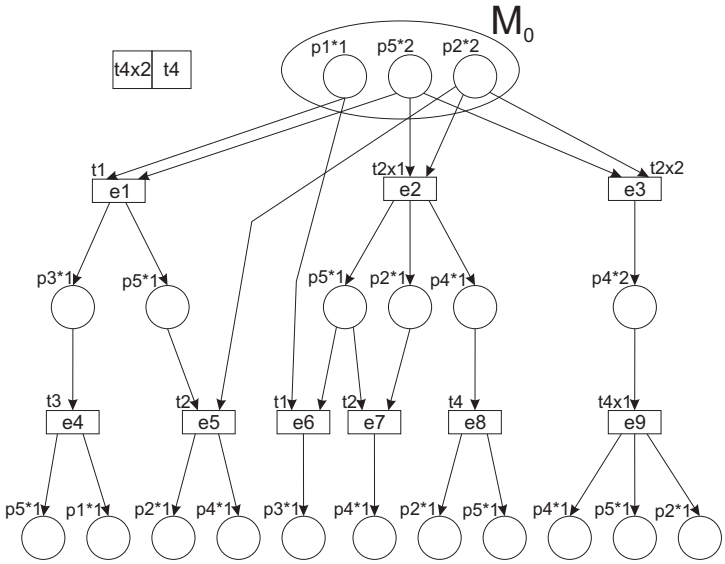

Figura 5.11: Processo de Desdobramento - fase 10.

Utilizando a menor configuração local, é escolhida a transição *t4* com dois disparos e retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição, o pós-conjunto e os lugares parcialmente consumidos desta, sendo resultante do disparo duplo de *t4* vezes a marcação: *p1* com uma marca, *p2* com duas marcas e *p5* com duas marcas, como apresentado na figura 5.12. O disparo duplo de *t4* é um evento de corte.

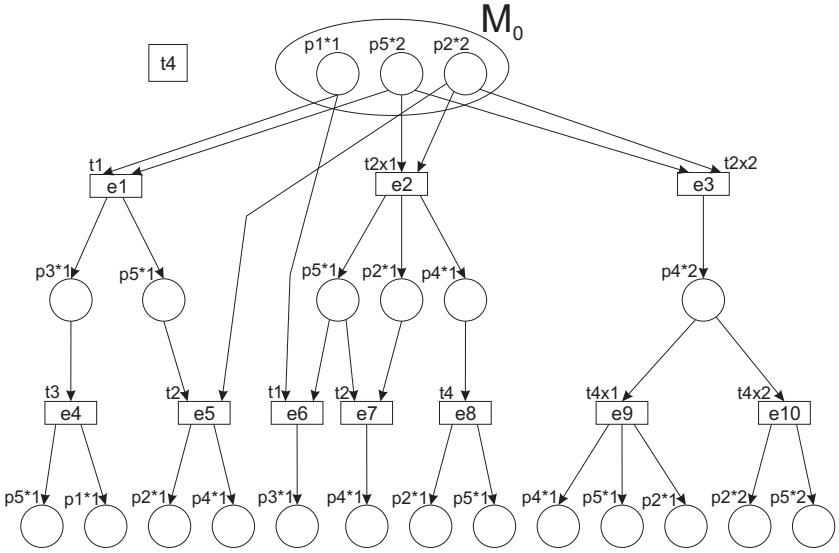

Figura 5.12: Processo de Desdobramento - fase 11.

Utilizando a menor configuração local, é escolhida a transição *t4* com um disparo e retirada da lista de transições. Logo, são adicionadas na rede de ocorrência a transição e o pós-conjunto e

os lugares parcialmente consumidos desta, sendo resultante do disparo de *t4* a marcação *p2* com uma marca, *p2* com uma marca, *p3* com uma marca e *p5* com uma marca, como apresentado na figura 5.13. Sendo esta transição um evento de corte.

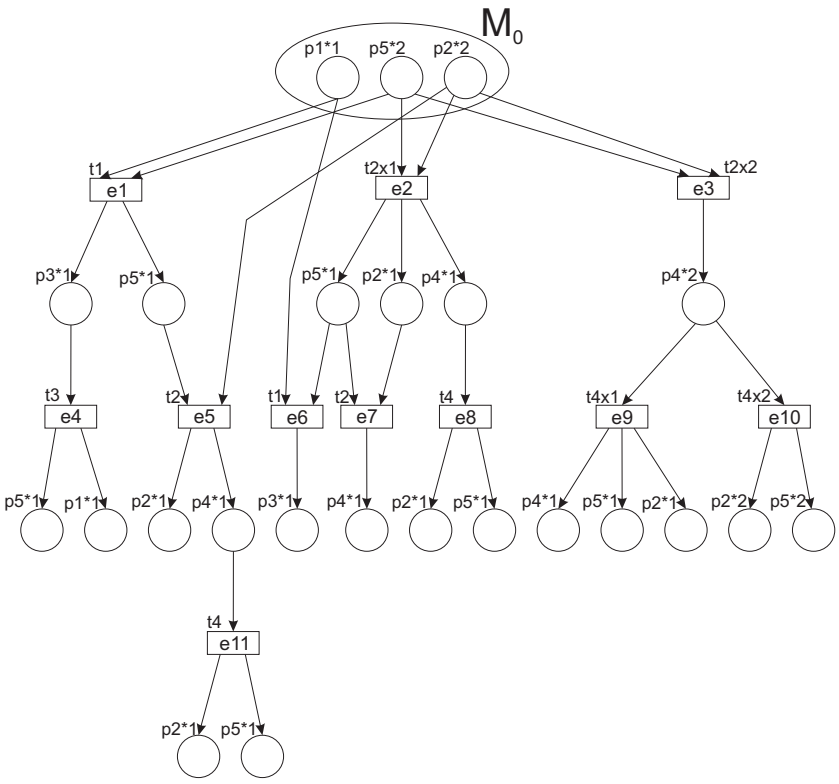

Figura 5.13: Processo de Desdobramento - fase 12.

Desta forma se conclui o processo de desdobramento da figura 5.1.

A figura 5.14 apresenta o grafo da alcançabilidade da figura 5.1, que mostra as marcações alcançáveis do sistema de rede exemplo. Comparando-se este grafo com o desdobramento obtido com a aplicação da nova abordagem, verificamos que este último contém todas as marcações alcançáveis.

5.2 Algoritmo

Para a criação do algoritmo da nova abordagem será utilizado como base o algoritmo ERV, mencionado na seção 3.4, sendo feitas algumas modificações para adequá-lo a nova abordagem.

A figura 5.15 apresenta o algoritmo da nova abordagem.

O procedimento principal do algoritmo funciona da mesma forma que o algoritmo ERV apresentado na seção 3.4. A principal diferença está na inserção das transições habilitadas na lista de transições habilitadas, uma vez que, no algoritmo ERV as transições habilitadas são

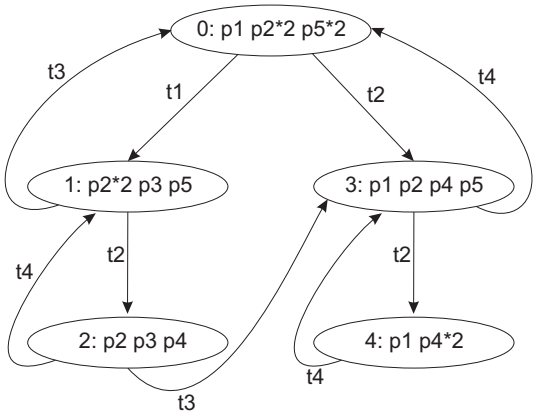

Figura 5.14: Grafo de Alcançabilidade.

inseridas em uma determinada marcação sem avaliação de múltiplas habilitações, pois este trabalha somente com redes seguras. No novo algoritmo é necessária a análise das marcações, para com isto poder identificar o número de disparos que uma determinada transição pode disparar de forma simultânea.

Esta análise é feita mediante o processo análise e uma lista auxiliar de transições habilitadas, a qual servirá para armazenar as transições habilitadas pela marcação inicial numa primeira instância e depois pelas dos homomorfismos da rede de ocorrência. Sendo estas analisadas pelo processo análise e inseridas na lista de transições habilitadas.

O processo análise verifica todas as transições habilitadas armazenadas na lista auxiliar, analisando a quantidade de marcas existentes nos lugares do pré-conjunto, identificando assim a quantidade de vezes que cada transição pode ser disparada e inserindo-la na lista de transições habilitadas. Para este processo serão utilizadas duas variáveis. Uma que auxiliara na quantidade de vezes que as marcas de um lugar podem disparar uma transição e outra que armazenara efetivamente a quantidade de vezes que a transição pode ser disparada. Logo, sendo preenchida a lista de transições habilitadas, mas tendo em consideração o número de vezes que a transição pode ser disparada.

5.3 Modificações na Estrutura da Ferranenta Mole

Para a implementação da nova abordagem será utilizada a ferramenta *Mole*, estudada no capítulo 4, sendo feitas algumas modificações nas estruturas e processos.

Estas alterações serão necessárias para adequar o *Mole* ao algoritmo da nova abordagem. A resultante destas modificações é uma ferramenta que realiza o desdobramento de redes de Petri k-limitadas.

```
1  procedure Analise(pe',pe){
2  forall t ∈ pe' \ pe do
3      fire ← ∝;
4      forall p ∈ •t do
5          cont ← 0;
6          while M(p) ⩾ •t do
7              M(p) ← M(p) - •t;
8              cont ← cont + 1;
9          end
10         if cont < fire then
11             fire ← cont;
12         end
13     end
14     for cont ← 1; cont ⩽ fire; cont ← cont + 1 do
15         pe ← pe ∪ t ∗ cont;
16     end
17 end
18 pe' ← pe;
19 }

   Input: Um SN = {N,M0}, sendo que M0 = {p1 , . . . , pk}
   Output: Unfolding Unf de SN
20 Unf ← lugares de M0;
21 pe ← Ø;
22 pe' ← transições habilitadas por M0;
23 Analise(pe',pe);
24 cut-off ← Ø;
25 while pe ≠ Ø do
26     escolha um evento e = (t, X) de pe tal que [e] seja mínimo;
27     if [e] ∩ cut-off = Ø then
28         adicione e e novas instancias dos lugares de h(e) em Unf;
29         pe' ← PE(Unf) {Atualiza as transições habilitadas};
30         Analise(pe',pe);
31         if e é um evento de corte then
32             cut-off ← cut-off ∪ e;
33         end
34     else
35         pe ← pe \ {e}
36     end
37 end
```

Figura 5.15: Algoritmo da Nova Abordagem

Para um melhor entendimento, as modificações feitas na ferramenta *Mole* serão divididas em dois tipos: as modificações feitas nas estruturas de armazenamento e as modificações feitas nos processos.

As modificações feitas nas estruturas foram necessárias para que as estruturas atuais do *Mole* possam suportar a rede de Petri L/T k-limitada de entrada e a rede de ocorrência resultante do processo de desdobramento. A ferramenta *Mole* trabalha com redes de Petri seguras, portanto não havia necessidade do armazenamento dos pesos dos arcos nem da quantidade de marcas nos lugares. Por esse motivo, foi necessário adicionar campos em algumas estruturas para poder efetuar a leitura da rede de entrada, o processo de desdobramento e a rede resultante da nova abordagem.

A seguir são mencionadas as modificações feitas nas estruturas:

Criação de um campo na estrutura *nodelist_t* apresentada na seção 4.2 para o armazenamento dos pesos dos arcos e número de marcas dos lugares de uma determinada marcação.

Criação de um campo na estrutura *cond_t* apresentada na seção 4.2, para o armazenamento das marcas dos lugares referentes ao homomorfismo da condição.

Criação de um campo na estrutura *event_t* apresentada na seção 4.2, para o armazenamento do número de disparos do homomorfismo do evento.

Criação de dois campos na estrutura *pe_queue_t* apresentada na seção 4.2, um para o armazenamento do número de disparos da transição habilitada e o outro para o armazenamento da quantidade dos lugares que foram parcialmente consumidos pelo disparo da transição habilitada que farão parte do pós-conjunto do homomorfismo da transição.

Modificadas as estruturas, para poder suportar a nova abordagem, foram feitas alterações em alguns processos. Estes serão mencionados a seguir.

Foi incrementado, no processo 1 da seção 4.3, a leitura e armazenamento do número de marcas nos lugares da marcação inicial e o peso dos arcos.

Foi adicionado, no processos 3 e 5 da seção 4.3, o armazenamento do número de marcas nas condições pertencentes aos lugares da marcação inicial.

Foi incrementado, no processo 4 e processo 10 da seção 4.3, a análise da transição habilitada, identificando a quantidade de vezes que esta pode ser disparada, a partir da criação de diferentes estruturas *pe_queue_t* para armazenar a mesma transição, mas com diferentes quantidades de disparo.

Além das alterações acima, foi modificada a criação da marcação gerada pelo disparo da transição habilitada, sendo esta diferente do processo original do *Mole*. São adicionados na marcação os lugares do pré-conjunto que foram parcialmente consumidos. Também são armazenandos em todos os lugares da marcação o número de marcas que possuem. O número

de marcas no caso da marcação inicial é igual, o número de marcas dos lugares parcialmente consumidos é igual ao número de marcas do lugar menos o peso do arco que liga o lugar com a transição multiplicado pelo número de disparos da transição. O número de marcas do pós-conjunto da transição é igual ao peso do arco que liga a transição com o lugar multiplicado pelo número de disparos da transição. Caso este lugar já esteja inserido na marcação, sua marcação efetiva será a soma de todas as marcações desse mesmo lugar.

Foi incrementado, no processo 9 da seção 4.3, o armazenamento do número de disparos do homomorfismo do evento, além de também somar o número de elementos do pós-conjunto da transição com a quantidade de lugares parcialmente consumidas pela transição sendo estas armazenadas no pós-conjunto do evento.

Foi modificado, no processo 9 da seção 4.3, a criação do pós-conjunto do evento, tendo sido criadas normalmente as condições pertencentes ao pós-conjunto como era feito no *Mole* e também as condições dos lugares parcialmente consumidos. Também é armazenado o número de marcas do homomorfismo da condição.

Foi adicionado, no processos 11 e 12 da seção 4.3, a verificação do número de marcas dos lugares pertencentes a marcação, de forma a identificar o evento de corte.

Foi incrementado, no processo 2 da seção 4.3, o armazenamento das marcas das condições e o número de disparos dos eventos no arquivo de saída. Para assim gerar a rede do processo de desdobramento com os rótulos do número de marcas nas condições e o número de disparos dos eventos.

5.4 Estudo de Caso

Para o estudo de caso da nova abordagem será utilizada a rede do artigo [6] apresentada na figura 5.16. Esta rede representa o fluxo da linha de produção de semiconductores.

Primeiramente, será feita a comparação em termos de tamanho do grafo de alcançabilidade com a rede de ocorrência que representa o processo de desdobramento, obtida com a implementação da nova abordagem. Em seguida será analisada a alcançabilidade de uma marcação, também comparando a rede de ocorrência e o grafo de alcançabilidade.

O grafo de alcançabilidade da rede da figura 5.16 possui 19040 estados e 124846 arcos e a rede de ocorrência obtida do processo de desdobramento possui 56249 condições e 48568 eventos, sendo as duas de grande tamanho e por tal motivo não apresentadas no trabalho. Comparando os dois resultados, temos que o número de estados é menor no grafo de alcançabilidade.

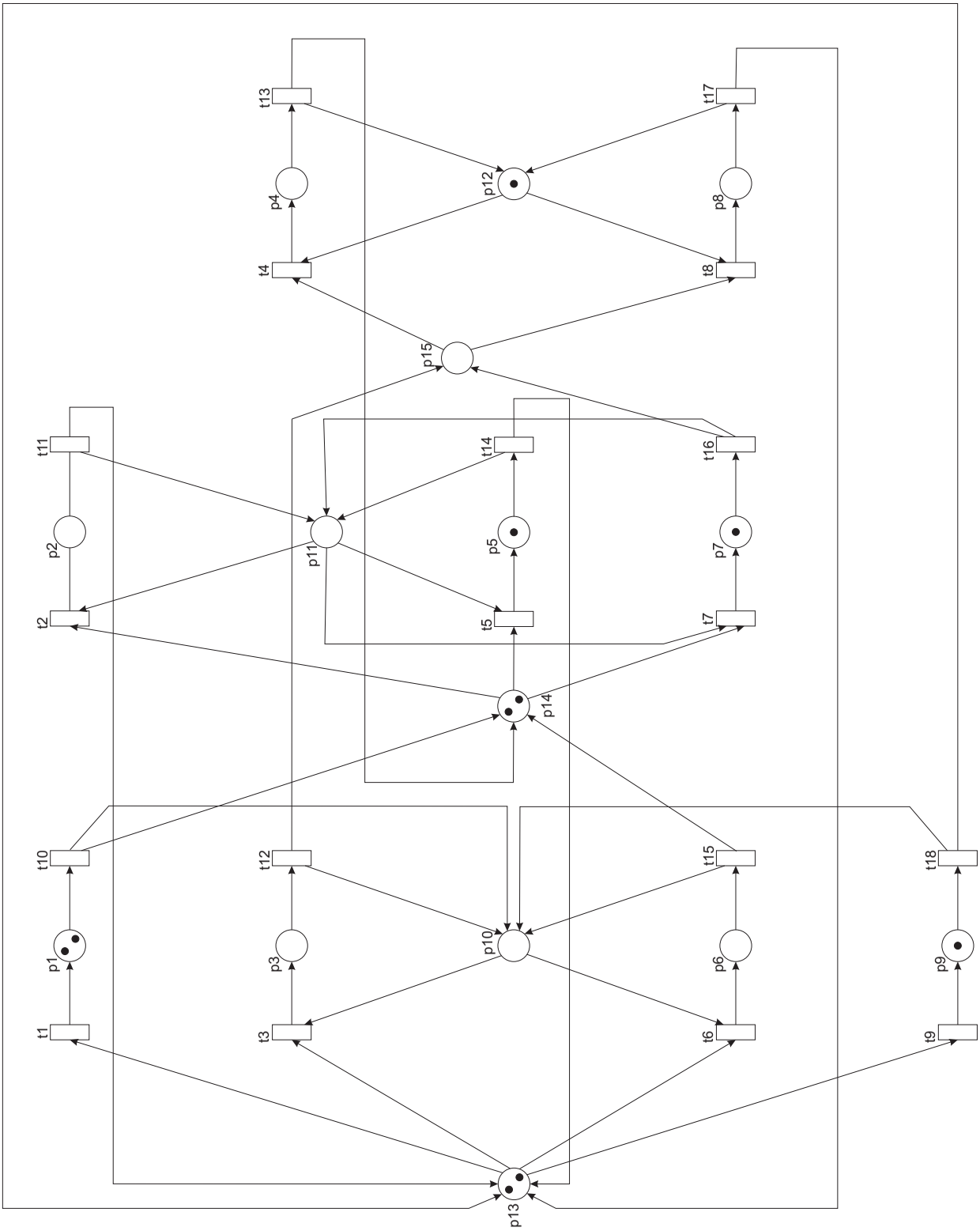

Figura 5.16: Rede exemplo.

Entretanto, no grafo, cada estado contém pelo menos quatro lugares marcados. No que se refere ao número de arcos este é superior ao número de eventos. Assim, podemos dizer que a rede de ocorrência é de menor tamanho que o grafo de alcançabilidade, sendo esta de mais fácil análise.

Nas figuras 5.17 e 5.18 são apresentadas, respectivamente, partes do grafo de alcançabilidade e da rede de ocorrência do processo de desdobramento da figura 5.16, sendo ambas equivalentes em termos de número de marcações. Comparando os grafos podemos visualizar que é possível alcançar uma mesma marcação por ambos. Por exemplo a marcação *p3*, *p5*, *p7*, *p9*, *p10*, *p12*, *p13* com uma marca e *p14* com quatro marcas é gerada pela sequência de disparos das transições *t10*, *t10* e *t3* no grafo de alcançabilidade, já na rede de ocorrência esta é gerada pelo disparo dos eventos *e1*(*t10x2*) e *e10*(*t3x1*).

5.5 Considerações

Os resultados obtidos do estudo de caso através da aplicação da nova abordagem constatou que é possivel chegar a todas as marcações do grafo de alcançabilidade mediante a rede de ocorrência gerada pelo processo de desdobramento. A implementação desta foi realizada mediante a modificação da ferramenta *Mole*, criando novos campos para o armazenamento de dados relevantes e modificando alguns processos, com isto conseguindo os resultados esperados.

Identificou-se também que quanto maior a rede original, a diferença de tamanho entre o grafo de alcançabilidade e a rede de ocorrência do processo de desdobramento é mais relevante, sendo a rede de ocorrência de menor tamanho que o grafo de alcançabilidade, tornando mais viável o uso da rede de ocorrência do processo de desdobramento para a análise da rede original.

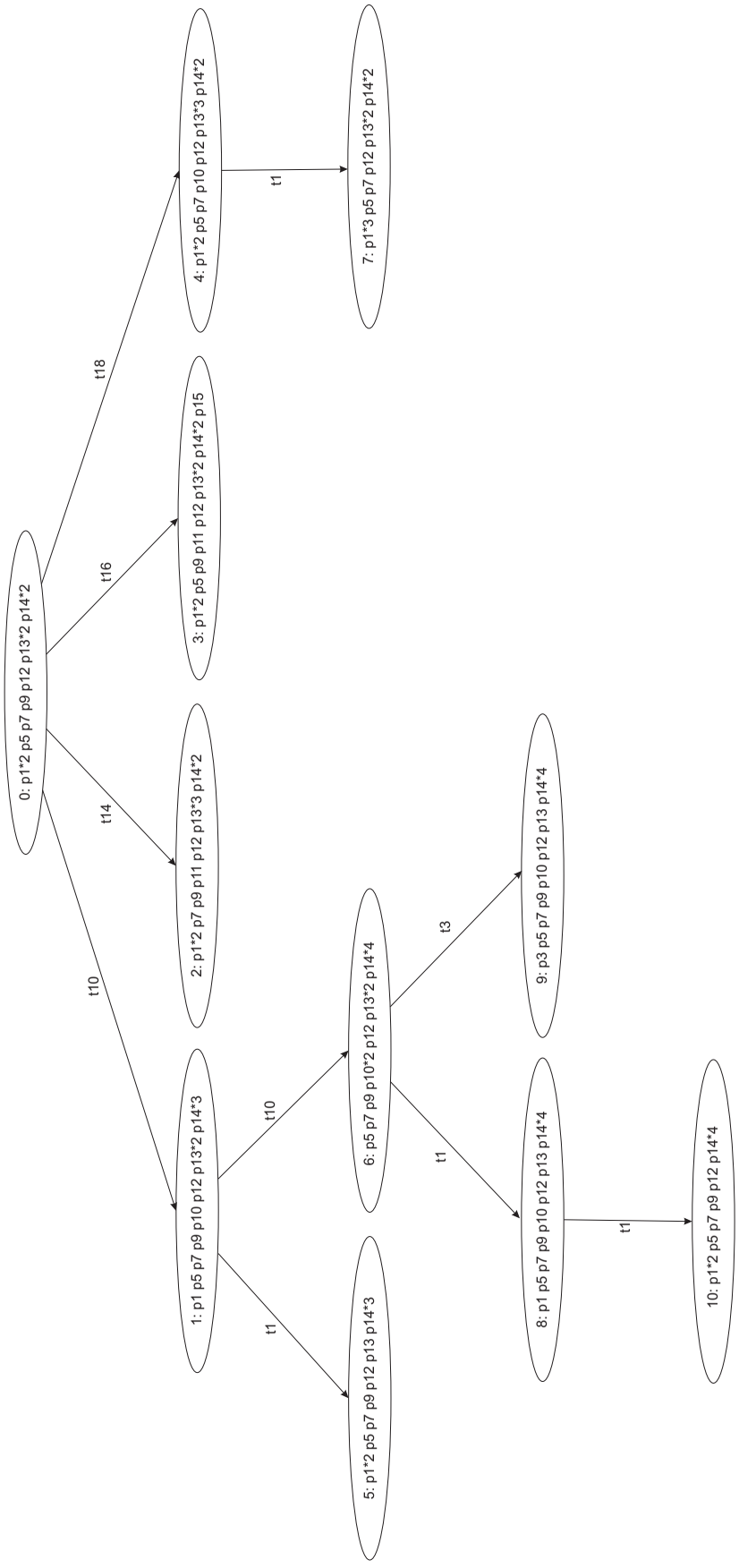

Figura 5.17: Grafo de Alcançabilidade.

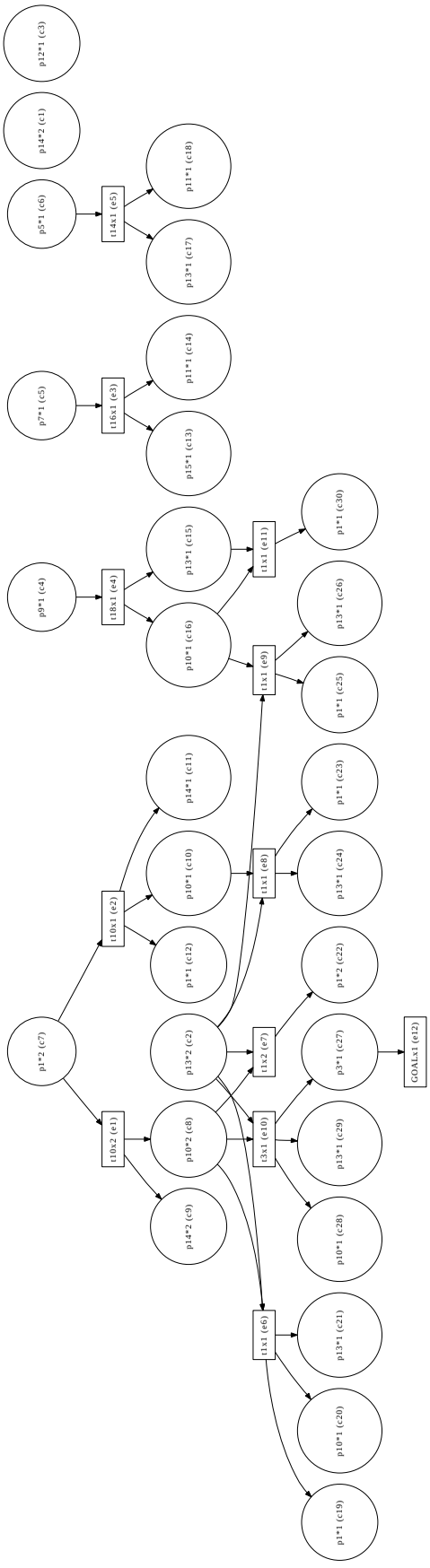

Figura 5.18: Rede de Ocorrência - Desdobramento.

6 Conclusão

O Laboratório de Inteligência Artificial e Métodos Formais, do Departamento de Informática da UFPR tem realizado diversas pesquisas buscando associar a solução para problemas de Planejamento em Inteligência Artificial com técnicas de alcançabilidade em Redes de Petri. A técnica de desdobramento é a melhor contribuição para problemas de alcançabilidade, mas o fato de existirem apenas implementações para redes seguras limita excessivamente seu uso, já que a maior parte dos problemas reais trabalha com múltiplos recursos, geralmente limitados. Desta forma, a falta de uma ferramenta que realize o processo de desdobramento de redes de Petri k-limitadas motivaram esta pesquisa. Neste trabalho buscou-se a criação e implementação de uma nova abordagem, sendo esta feita mediante modificações da ferramenta *Mole*.

Para alcançar os objetivos deste trabalho, foram estudados os conceitos básicos de redes de Petri e desdobramento. Também foram feitas pesquisas sobre algoritmos e processos de desdobramento para redes de Petri k-limitadas, sendo encontrados apenas algoritmos e ferramentas para redes de Petri 1-limitadas e algumas abordagens para o desdobramento de redes de Petri coloridas. Por tal motivo, foi necessário criar uma nova abordagem tendo em consideração vários artigos de como seria possível fazer o desdobramento de redes de Petri k-limitadas. Uma das abordagens estudadas consistiu em converter uma rede de Petri k-limitada para uma rede de Petri 1-limitada, para em seguida fazer o processo de desdobramento desta. Esta alternativa, como descrito em [2], não é recomendada, uma vez que neste processo a rede pode perder algumas propriedades. Desta forma, a solução adotada teve como ação buscar construir o processo de desdobramento da rede k-limitada.

A estratégia de abordagem consistiu em trabalhar a partir da ferramenta *Mole*, considerada como uma das melhores ferramentas no processo de desdobramento. Esta ferramenta foi profundamente estudada e em seguida modificada, de forma a que atendesse a nova abordagem proposta para o desdobramento de redes k-limitadas. Foram criadas novas estruturas e alterados alguns processos. O resultado final foi a obtenção de uma ferramenta com capacidade de realizar o processo de desdobramento para redes de Petri k-limitadas.

Para validar a implementação, a principal dificuldade consistiu na não existência de ferramentas que realizem o processo de desdobramento de redes de Petri k-limitadas. Desta forma, não foi possível realizar a comparação dos resultados obtidos com outras implementações. Desta forma, foi escolhida uma rede de teste para estudo de caso e os resultados obtidos foram comparados com o grafo de alcançabilidade desta mesma rede. Foi possível verificar, a partir da rede de ocorrência resultante do processo de desdobramento, que é possível chegar a todas as marcações existentes no grafo de alcançabilidade.

Foi possível também constatar um resultado já esperado. A rede de ocorrência obtida pela nova abordagem tem tamanho menor que o grafo de alcançabilidade, o que facilita a análise a partir desta, já que a estrutura gerada pelo processo de desdobramento auxilia no problema de explosão de estados. Estima-se que, tal como ocorre com as redes seguras, quanto maiores forem as redes k-limitadas analisadas, maior será a diferença entre a rede de ocorrência e o grafo de alcançabilidade.

A nova abordagem apresenta uma limitação em relação as redes com peso nos arcos, sendo que em algumas ocasiões a marcação efetiva de um lugar é a soma das marcas nas ocorrências desta numa marcação. Isto pode acarretar que, quando seja feita a identificação das transições habilitadas pela rede de ocorrência possa acontecer que uma condição não habilite a transição pelo peso do arco desta, mesmo que a marcação efetiva do homomorfismo da condição seja suficiente para habilitar a transição, mas como o processo de habilitação das transições é feita pelas condições concorrentes isto pode produzir que a transição seja desabilitada.

Por exemplo, dada a marcação *p1* com uma marca, *p2* com três marcas e *p1* com duas marcas, a marcação efetiva de *p1* são três marcas e uma transição *t3* com peso três pertencente ao pós-conjunto de *p1* pode ser habilitada, mas como a condição concorrente é *p1* com uma marca, isto faz com que a transição não seja habilitada pelo peso do arco.

Uma possível solução é fazer uma análise da existência de ocorrências do homomorfismo da condição que faz parte do pré-conjunto da transição habilitada, verificando assim se a marcação efetiva habilita a transição. Em seguida, serão adicionadas a condição ou condições ao pré-conjunto do evento desta.

Este trabalho abre diversas possibilidades de trabalhos futuros, como por exemplo a aplicação de paralelismo como descrito em [8], tornando o processo de desdobramento mais rápido, além da aplicação de heurísticas como mencionado em [7], [22] e [1], para problemas de planejamento com multirecursos.

Referências Bibliográficas

[1] Juliana H. Q. Benaccio. Planejamento em inteligência artificial utilizando redes de petri cíclicas. Master's thesis, Universidade Federal do Paraná, 2008.

[2] Eike Best and Harro Wimmel. Reducing k-safe petri nets to pomset-equivalent 1-safe petri nets. pages 63–82, 2000.

[3] Janette Cardoso and Robert Valette. *Redes de Petri*. Editora da UFSC, 1st edition, 1997.

[4] Javier Esparza, Stefan Romer, and Walter Vogler. An improvement of mcmillan's unfolding algorithm. 1992.

[5] Gnu gpl. `http://www.gnu.org/licenses/gpl.html`, acessado em 01/04/2010.

[6] Jonathan D. Green and Nicholas G. Odrey. Petri net models for analysis and control of re-entrant flow semiconductor wafer fabrication. Technical report.

[7] Sarah L. Hickmott. *Directed Unfolding - Reachability Analysis of Concurrent Systems & Applications to Automated Planning*. PhD thesis, University of Adelaide, Bonn, 2008.

[8] Victor Khomenko. *Model checking based on prefixes of petri net unfoldings*. PhD thesis, University of the Newcastle upon Tyne, Bonn, 2003.

[9] Victor Khomenko, Keijo Heljanko, and Maciej Koutny. Towards an efficient algorithm for unfolding petri nets. 2001.

[10] Victor Khomenko, Keijo Heljanko, and Maciej Koutny. Parallelisation of the petri net unfolding algorithm. 2002.

[11] Richard Lipton. The reachability problem requires exponential space. Technical report, 1976.

[12] Kenneth L. McMillan. *Symbolic Model Checking - An Approach to the state explosion problem*. PhD thesis, Carnegie Mellon University, Bonn, 1992.

[13] Kenneth L. McMillan. A technique of state space search based on unfolding. 1992.

[14] Mole. `http://www.fmi.uni-stuttgart.de/szs/tools/mole/`, acessado em 01/09/2008.

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

[16] Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[17] Punf. http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf/, acessado em 01/10/2008.

[18] Wolfgang Reisig. *Petri Nets - An Introduction*. Springer-Verlag, 1985.

[19] Claus Schroter and Javier Esparza. Unfolding based algorithms for the reachability problem. 2001.

[20] Fabiano Silva. *Rede de Planos: Uma Proposta para a Solução de Problemas de Planejamento em Inteligencia Artificial Usando Redes de Petri*. PhD thesis, Centro Federal de Educação Tecnológica do Paraná, 2005.

[21] Iain A. Stewart. On the reachability problem for some classes of petri nets. Technical report, 1992.

[22] Guilherme S. Töws. Petrigraph : um algoritmo para planejamento por desdobramento de redes de petri. Master's thesis, Universidade Federal do Paraná, 2008.