Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

# MINERAÇÃO DE PADRÕES TEMPORAIS HÍBRIDOS ESPECIFICADOS NA LÓGICA TEMPORAL DE INTERVALOS

Waldecir Pereira Junior

Uberlândia - MG
Novembro de 2007

**Universidade Federal de Uberlândia**

**Faculdade de Computação**

**Programa de Pós-Graduação em Ciência da Computação**

Waldecir Pereira Junior

# Mineração de Padrões Temporais Híbridos Especificados na Lógica Temporal de Intervalos

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.
Área de concentração: Banco de Dados.

Orientadora:
Prof[a]. Dr[a]. Sandra de Amo

Uberlândia, MG
2007

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Mineração de Padrões Temporais Híbridos Especificados na Lógica Temporal de Intervalos**" por **Waldecir Pereira Junior** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 30 de Novembro de 2007

Orientador: _____

Profª. Drª. Sandra de Amo

Universidade Federal de Uberlândia UFU/MG

Banca Examinadora: _____

Profª. Drª. Denise Guliato

Universidade Federal de Uberlândia UFU/MG

_____

Prof. Dr. Caetano Traina Júnior

Universidade de São Paulo USP/SP

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Data: Novembro de 2007

| | |
|---|---|
| Autor: | **Waldecir Pereira Junior** |
| Título: | **Mineração de Padrões Temporais Híbridos Especificados na Lógica Temporal de Intervalos** |
| Faculdade: | **Faculdade de Computação** |
| Grau: | **Mestrado** |

Fica garantido à Universidade Federal de Uberlândia o direito de circulação e impressão de cópias deste documento para propósitos exclusivamente acadêmicos, desde que o autor seja devidamente informado.

_____

Autor

# Dedicatória

*A meus pais Waldecir e Analice, a meu irmão Jean e à minha irmã Larrissa.*

*À minha namorada Ridinely.*

# Agradecimentos

Agradeço...

A Deus, por minha vida.

A meus pais Waldecir Pereira e Analice da Silva Pereira pela dedicação, confiança, apoio, carinho e amor incondicional em todos os momentos.

A meus irmãos Jean Carlos Pereira e Larissa Mara Pereira pelos conselhos, incentivos, carinho e amor durante toda esta etapa da minha vida.

À minha namorada Ridinely Otuki por ter me apoiado tanto durante este projeto. Você me deu muita força, tranqüilidade, paz, alegria e amor.

A meus amigos do Laboratório de Banco de Dados que se mostraram companheiros e, diretamente ou indiretamente, contribuíram para a realização deste trabalho, em especial à Tarcísio Gotto Clemente pelo trabalho realizado no gerador de dados sintéticos, à Cristiano Inácio, Rafael Araújo, Diogo Nunes, Gabriel Rossi e Ricardo José da Silva pelo trabalho de construção do banco de dados real. À Crícia Felício, Marcos Ribeiro, Ricardo Soares, Juliano Dalóia, Elaine Ribeiro, Jean Carlo de Sousa, Mariangela Simedo, Tauller Matos e Vinicius Borges pelos conselhos e apoio.

A meus amigos Francisco Neto, Guilherme Azevedo e Amir Heran pelo incentivo e paciência.

Principalmente à professora Sandra de Amo pelo profissionalismo, apoio, paciência, amizade e orientação em todos os momentos da realização deste trabalho.

*"A grandeza não consiste em receber honras,*

*mas sim, em merecê-las!"*

(Aristóteles)

# Resumo

A descoberta de padrões freqüentes em bancos de dados constitui um importante problema do domínio da descoberta de conhecimentos e sua importância é justificada pela diversidade de áreas onde pode ser empregada, como no varejo, no mercado financeiro, na medicina, na agricultura, na agropecuária, em empresas de telecomunicações, etc.

O problema de mineração de padrões freqüentes em bancos de dados temporais, conhecido também como mineração de padrões temporais, tem sido amplamente estudado. Em alguns trabalhos os padrões temporais propostos são especificados por formalismos da Lógica Temporal Proposicional, em outros, eles são mais expressivos, por isso, são especificados por formalismos da Lógica Temporal de Primeira Ordem.

Os padrões temporais existentes na literatura representam o tempo em termos de *pontos*, onde, seus eventos ocorrem em determinados instantes, ou em termos de *intervalos*, onde, seus eventos ocorrem durante períodos de tempo. Esta distinção faz com que alguns fatos não sejam reconhecidos, como por exemplo fatos relacionados ao histórico clínico de um paciente, onde o paciente tomou determinado medicamento durante um *intervalo* e sofreu uma determinada cirurgia, em uma certa data, durante o tempo em que estava tomando o medicamento.

Nesta dissertação esta sendo proposto um novo padrão temporal, chamado *padrão temporal híbrido* ou simplesmente *pth*, que representa o tempo explicitamente em termos de pontos e/ou intervalos. Este padrão é bastante expressivo, por isso, a Lógica Temporal de Intervalos de Allen foi adaptada para especificá-lo. Está sendo proposto também um algoritmo, chamado MILPRIT*, para minerar os *pth's* freqüentes em bancos de dados temporais com relação a um suporte mínimo e a uma restrição especificada pelo usuário através de uma expressão regular. Isto permite ao usuário um maior controle sobre o processo de mineração. A performance e a escalabilidade do MILPRIT* foi avaliada através de um conjunto de testes em bancos de dados sintéticos e real.

**Palavras chave:** mineração de dados temporais, lógica temporal de intervalos, mineração baseada em restrições e padrões seqüenciais.

# Abstract

Discovering frequent patterns in databases is an important problem for knowledge discovery and its importance is justified by the diversity of areas where it can be used, such as retail, financial market, medicine, agriculture, farming, telecommunications, etc.

The problem of mining frequent patterns in temporal databases, also know as mining temporal patterns, has been widely studied. In some works the temporal patterns are expressed by *propositional* temporal logic, and in others, the temporal patterns are more expressive and are expressed by *first order* temporal logic.

The existing temporal patterns in the literature represent the time either in terms of *points*, where event occur in determined instants, or in terms of *intervals*, where event occur during a period of time. This dichotomy implies that some facts may not be inferred. For instance, facts related to the clinical history of a patient, where, the patient took some medicine during a period of time and was submitted to a surgery on a day, during the period when he or she was taking the medicine.

In this dissertation, we propose a new temporal pattern, called the *hybrid temporal pattern* or simply *htp*, where, time is represented in terms of *points* and/or *intervals*. This pattern is very expressive. We adapted Allen's Interval Temporal Logic to specify it. We also proposed the algorithm MILPRIT* for mining the frequents *htp's* in a database with respect to a minimum support and satisfying a constraint specified by user through a regular expression. Doing so, we allow the user to control the process of *htp* discovery. The performance and scalability of MILPRIT* has been evaluated through a set of experiments over synthetic and real databases.

**Keywords:** temporal data mining, interval temporal logic, constraint-based mining and sequential patterns.

# Sumário

# Lista de Figuras

# Capítulo 1

# Introdução

Nas últimas décadas, com o grande crescimento do uso de computadores, o tamanho dos bancos de dados existentes nas mais variadas instituições aumentou consideravelmente impossibilitando a interpretação e a compreensão humana dos dados armazenados. Na maioria dos casos, depois que supostamente estes dados já serviram a seus propósitos eles ficam inutilizados, pois, as instituições não sabem o que fazer com eles. Para suprir a incapacidade humana de analisar enormes quantidades de dados e melhor aproveitá-los surgiu a *mineração de dados* (*data mining*), como uma poderosa ferramenta para a descoberta de informações importantes que não estão explícitas nos bancos de dados.

A mineração de informações interessantes ocultas em bancos de dados é uma área de pesquisa muito abrangente que engloba várias tarefas de mineração, das quais podem ser destacadas a mineração de regras de associação, classificação, agrupamentos (*clusters*) e mineração de padrões seqüenciais.

O problema de mineração de padrões seqüenciais freqüentes em bancos de dados temporais, conhecido também como mineração de padrões temporais, tem sido amplamente estudado [3, 4, 5, 6] e sua importância é justificada pela diversidade de áreas onde esse tipo de mineração pode ser empregada, tais como: no varejo (pode-se ter interesse em minerar padrões que expressam a evolução de compras de clientes), no mercado financeiro (pode-se ter interesse em minerar padrões que expressam a evolução de cotações de ações), na medicina (pode-se ter interesse em minerar padrões que expressam a evolução de sintomas de pacientes), em telecomunicações (pode-se ter interesse em minerar padrões que expressam seqüências de disparos de alarmes), etc. Já foram propostos vários tipos de

padrões [6, 7, 8, 9] juntamente com inúmeros formalismos e algoritmos que possibilitam a mineração dos mesmos [10, 11, 12, 13, 14, 15, 16]. Alguns dos padrões propostos são especificados por formalismos da Lógica Temporal Proposicional. Por exemplo, considere um padrão seqüencial clássico da forma $s =< \{a, b\}, \{c, d\} >$, onde, $\{a, b\}$ e $\{c, d\}$ são conjuntos de itens comprados por um cliente. Este padrão pode ser expresso na Lógica Temporal Proposicional pela fórmula $P_a \wedge P_b \wedge \Diamond(P_c \wedge P_d)$, onde, para cada $i \in \{a, b, c, d\}$, $P_i$ é um símbolo proposicional que significa *"clientes compram o item i"*. O símbolo $\Diamond$ é o operador temporal *"em algum momento no futuro"*.

Outros padrões propostos, como os que modelam comportamentos de usuários do sistema Unix [17], necessitam de uma maior expressividade, por isso, são especificados por formalismos da Lógica Temporal de Primeira Ordem. Por exemplo, considere a seguinte seqüência de comandos utilizados por usuários do Unix: $vi\ texto, mv\ texto, rm\ texto, ls$. Esta seqüência pode ser representada por uma seqüência de átomos de primeira ordem da forma: $vi(texto), mv(texto), rm(texto), ls$, permitindo a descoberta de padrões freqüentes da forma $vi(X)\ rm(Y)$, onde, $vi\ rm$ é uma seqüência de comandos (predicados) freqüentemente usados pelos usuários, e $X$ e $Y$ representam nomes de arquivos (parâmetros).

Os padrões temporais existentes na literatura representam o tempo em termos de *pontos*, onde os eventos ocorrem em determinados instantes, ou em termos de *intervalos*, onde os eventos ocorrem durante períodos de tempo. Esta distinção faz com que alguns fatos não sejam reconhecidos pelos processos de descoberta de padrões, como por exemplo, fatos relacionados ao histórico clínico de um paciente, onde, o paciente tomou um medicamento $X$ por um período de tempo $e$ durante um período de tempo $f$ em que apresentou um sintoma $Y$, e após ter interrompido o uso do medicamento $X$ e antes do sintoma $Y$ ter desaparecido sofreu uma cirurgia $Z$ em um instante $t$.

Nesta dissertação está sendo proposto um novo tipo de padrão temporal, chamado *padrão temporal híbrido* ou simplesmente *pth*, que representa o tempo explicitamente em termos de pontos e/ou intervalos. Este padrão é bastante expressivo, por isso ele é especificado por formalismos da Lógica Temporal de Intervalos de Allen [2] que foi adaptada para esta finalidade. Está sendo proposto também um algoritmo chamado MILPRIT* para minerar este novo padrão temporal.

## 1.1 Motivação

O padrão temporal híbrido proposto nesta dissertação pode aparecer em bancos de dados temporais relacionais onde o tempo é representado em termos de pontos e/ou intervalos, e tem como objetivo representar os relacionamentos entre eventos ocorridos em instantes ou períodos ao longo do tempo. A *mineração de padrões temporais híbridos* pode ser empregada em vários domínios de aplicações, como por exemplo na agricultura (pode-se ter interesse em descobrir se o uso de agrotóxicos durante determinados períodos de tempo tem influência na evolução de determinadas plantações), na agropecuária (pode-se ter interesse em descobrir se o uso de determinadas rações durante um determinado período de tempo e a aplicação de vacinas em alguns instantes influenciam na evolução de doenças ou no desenvolvimento de rebanhos), na medicina (pode-se ter interesse em descobrir se pacientes que tomam um medicamento X e entram na menopausa durante o tempo em que estão tomando esse medicamento irão apresentar um sintoma Y no futuro). A seguir, será apresentado um exemplo onde a mineração de padrões temporais híbridos é empregada na medicina.

**Exemplo 1.1.1** Suponha que o objetivo seja descobrir como o estilo de vida e casos clínicos de pacientes do sexo feminino influencia ou não no surgimento de lesões, malignas ou benignas, na mama. Considere o banco de dados simplificado de uma clínica que realiza exames mamográficos mostrado na figura 1.1. A tabela *Paciente* armazena os nomes de todas as pacientes que realizaram exames mamográficos. A tabela *Birads* armazena os resultados dos exames mamográficos de cada paciente. Na classificação Bi-Rads [18] os laudos mamográficos são divididos em sete categorias, quanto maior a categoria maior é a chance da paciente ter uma lesão maligna. As tabelas *Gravidez*, *Fumo* e *Hormônio* armazenam os instantes de cada gravidez, períodos em que fumou e períodos em que usou hormônios, respectivamente.

Os atributos $T_{it}$ e $T_{pt}$ representam respectivamente o período (intervalo) e o instante (ponto) no qual determinado evento aconteceu. Os números que aparecem nestas colunas são mapeados em datas de acordo com a linha do tempo, portanto, neste banco de dados os eventos aconteceram em um ponto (uma data) ou em um intervalo (durante o período entre duas datas).

*Paciente*

| *IdPac* | *Nome* |
|---|---|
| 01 | Maria |
| 02 | Priscila |
| 03 | Carla |
| 04 | Paula |
| 05 | Cintia |

*Birads*

| *Paciente* | *Categoria* | $T_{pt}$ |
|---|---|---|
| Maria | 3 | 10 |
| Paula | 3 | 9 |
| Aline | 2 | 2 |
| Cintia | 4 | 5 |

*Gravidez*

| *Paciente* | *Status* | $T_{pt}$ |
|---|---|---|
| Maria | sim | 3 |
| Carla | sim | 2 |
| Paula | sim | 1 |

*Fumo*

| *Paciente* | *Qtde* | $T_{it}$ |
|---|---|---|
| Maria | 5 | [4,10] |
| Priscila | 10 | [2,4] |
| Paula | 10 | [2,5] |
| Cintia | 20 | [3,4] |

*Hormônio*

| *Paciente* | *NomeHorm* | $T_{it}$ |
|---|---|---|
| Maria | estradiol | [5,8] |
| Paula | progesterona | [6,8] |
| Priscila | progesterona | [5,9] |
| Carla | noretisterona | [4,5] |

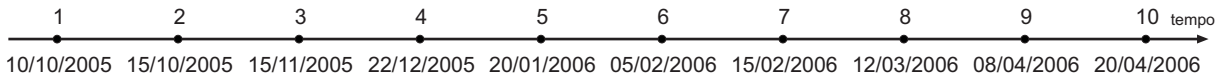



Figura 1.1: Exemplo de um banco de dados de uma clínica de mamografia

Considere os seguintes comportamentos:

1. "Pacientes tiveram filho em um instante e futuramente apresentaram Bi-Rads 3".
2. "Pacientes fumaram 10 cigarros por dia e após terem parado de fumar iniciaram o uso do hormônio *progesterona*.
3. "Pacientes fumaram uma quantidade de cigarros por dia e após terem parado de fumar apresentaram alguma categoria de Bi-Rads.

Note que o comportamento 1 é compatível com o das pacientes Maria e Paula e envolve somente tabelas onde o tempo é representado por pontos (*Gravidez* e *Birads*). Note também que o comportamento 2 é compatível com o das pacientes Priscila e Paula e envolve somente tabelas onde o tempo é representado por intervalos (*Fumo* e *Hormônio*). Já, o comportamento 3 é compatível com o das pacientes Paula e Cintia e envolve uma tabela que representa o tempo por intervalos (*Fumo*) e uma tabela que representa o tempo por pontos (*Birads*). É importante enfatizar que este último comportamento não pode ser expresso diretamente por nenhum dos padrões existentes na literatura.

No entanto, padrões temporais híbridos são especificados por um conjunto de fórmulas atômicas onde o tempo é expresso por variáveis que representam pontos ou intervalos

e estas variáveis são relacionadas pelos predicados temporais $before$, $meets$, $overlaps$, $starts$, $during$ ou $finishes$. Assim, o comportamento 1 pode ser expresso por um padrão temporal híbrido que é uma tripla ($K$,$\mathcal{E}$,$\mathcal{T}$), onde, $K = Paciente(x)$ (significa que $x$ é uma paciente registrada no banco de dados), $\mathcal{E} = \{Gravidez(x, sim, e), Birads(x, 3, f)\}$ (representa os eventos do padrão) e $\mathcal{T} = \{before(e, f)\}$ (representa os relacionamentos que são verificados pelos parâmetros temporais associados aos eventos do padrão), o comportamento 2 pode ser expressado por $K = Paciente(x)$, $\mathcal{E} = \{Fumo(x, 10, e), Hormônio(x, progesterona, f)\}$ e $\mathcal{T} = \{before(e, f)\}$ e o comportamento 3 pode ser expressado por $K = Paciente(x)$, $\mathcal{E} = \{Fumo(x, y, e), Birads(x, z, f)\}$ e $\mathcal{T} = \{before(e, f)\}$.

## 1.2 Contribuições

As principais contribuições desta dissertação são:

- A proposta de um novo padrão temporal, chamado *padrão temporal híbrido*, que representa o tempo explicitamente em termos de pontos e/ou intervalos e é especificado pelos formalismos da Lógica Temporal de Primeira Ordem.

- O desenvolvimento do algoritmo MILPRIT*[1], que é o resultado de otimizações e extensões de um protótipo, chamado MILPRIT [16], desenvolvido para mineração de padrões temporais onde o tempo é representado unicamente em termos de intervalos.

    1. Otimizações: Uma das otimizações foi realizada com o objetivo de aperfeiçoar as relações entre a etapa de geração dos padrões, a etapa de poda dos padrões e o consumo de memória, uma vez que os padrões gerados pelo MILPRIT preenchem rapidamente a memória principal. Outra otimização foi realizada com o objetivo de corrigir uma falha na etapa da geração, uma vez que o MILPRIT não gera determinados padrões ou gera padrões que não interessam ao usuário. Um trabalho abordando estas otimizações foi publicado em [19].

[1] O algoritmo *MILPRIT** está disponível em www.lsi.ufu.br

2. Extensão: A extensão foi realizada com o intuito de permitir que o MILPRIT* além de minerar padrões temporais que representam o tempo por intervalos minere também padrões temporais que representam o tempo por pontos. Um trabalho abordando esta extensão foi publicado em [20].

- Extensão dos fundamentos teóricos do protótipo MILPRIT [16] para o contexto do algoritmo MILPRIT*.

  1. Durante o processo de mineração, o protótipo MILPRIT gera somente padrões *completos* e *consistentes*, assim, eles podem ser vistos como seqüências, o que possibilita que sejam especializados de modo que satisfaçam uma expressão regular.

  2. O protótipo MILPRIT utiliza dois operadores (instanciação e extensão) para a especialização dos padrões. Esses operadores são *completos* e *optimais*. Assim, a eficiência do método é garantida, pois, são gerados todos os padrões possíveis a partir de um dado padrão e não são gerados padrões duplicados.

     Todos esses fundamentos foram estendidos para o problema de mineração proposto e implementados no algoritmo MILPRIT*.

- A incorporação de restrições especificadas através de expressões regulares como uma ferramenta para incorporar ao processo de mineração o foco do usuário, promovendo assim a redução do espaço de busca dos padrões.

- O desenvolvimento de um gerador de dados sintéticos[2], que permite que diferentes tipos de bancos de dados contendo informações temporais representadas por pontos e/ou intervalos sejam criados de acordo com as necessidades do usuário.

- A construção de um banco de dados real[3], cuja modelagem foi baseada na classificação Bi-Rads, padrão publicado pela ACR (Colégio Americano de Radiologia) [18]. O banco de dados real é um banco de dados temporal que contém informações sobre pacientes do sexo feminino, como estilo de vida e histórico clinico.

[2] O *gerador de dados sintéticos* está disponível em www.lsi.ufu.br
[3] O *banco de dados real* está disponível em www.lsi.ufu.br

## 1.3 Organização da Dissertação

Esta dissertação está dividida como segue.

No capítulo 2 serão apresentados alguns conceitos básicos de mineração de dados e tarefas de mineração, juntamente com os principais trabalhos relacionados à mineração de padrões temporais. O problema de mineração proposto será apresentado no capítulo 3. No capítulo 4 serão apresentados os conceitos teóricos relativos à classe de restrições sobre os padrões temporais que é incorporada no processo de mineração. O algoritmo MILPRIT* juntamente com suas principais sub-rotinas será apresentado no capítulo 5. Os resultados dos experimentos realizados com o MILPRIT* serão apresentados no capítulo 6. Finalmente, no capítulo 7 serão apresentadas as conclusões e perspectivas para trabalhos futuros.

# Capítulo 2

# Mineração de Dados

Como o aumento dos bancos de dados surgiu o interesse e a necessidade de estudar maneiras de extrair conhecimentos automaticamente dos mesmos [1, 21]. Com a extração de conhecimentos, os grandes bancos de dados passaram a ser vistos como valiosas fontes de informações que podem auxiliar em tomadas de decisões, marketing, finanças e em muitas outras tarefas.

A **Mineração de Dados** (*Data Mining*) é apenas uma etapa, entre várias, do processo de descoberta (extração) de conhecimentos em grandes bancos de dados. De maneira superficial, mineração de dados corresponde em aplicar a uma base de dados pré-processada de forma adequada algoritmos de técnicas de descoberta de informações interessantes.

## 2.1 O Processo de Descoberta de Conhecimento

Atualmente existe uma necessidade urgente de se criar novas teorias e ferramentas para auxiliar os seres humanos na extração de informações importantes dos crescentes volumes de dados armazenados em todos os tipos de bancos de dados. O desenvolvimento dessas teorias e ferramentas é o objetivo da **Descoberta de Conhecimento em Bancos de Dados** (*Knowledge Discovery in Databases* - KDD) [1].

KDD é definido em [1] como o processo de extração de padrões interessantes, não triviais, implícitos, de antemão desconhecidos e potencialmente úteis em grandes bancos de dados. Os *dados* são um conjunto de fatos (casos nas bases de dados), e o *padrão* é uma

expressão em alguma linguagem descrevendo um subconjunto de dados. *Processo* informa que KDD é constituído de várias etapas, envolvendo a preparação dos dados, a busca por padrões, a avaliação do conhecimento, e o refinamento, e todas essas etapas podem ser repetidas em múltiplas iterações. *Não trivial* indica que o processo não é direto e pode envolver o uso de técnicas de busca e algoritmos.

O processo KDD é divido em etapas bem definidas: (1) seleção dos dados, (2) pré-processamento e transformação, (3) mineração de dados e (4) interpretação e avaliação dos resultados. A figura 2.1 ilustra os passos do processo KDD.

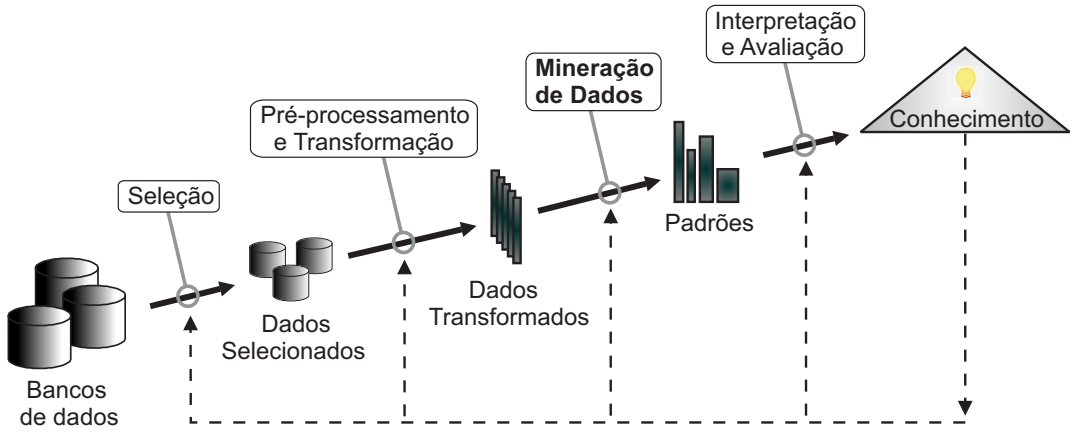



Figura 2.1: Passos do KDD (*adaptação de [1]*)

Na *seleção*, são selecionados os dados que interessam ao usuário, sobre os quais o processo de descoberta de conhecimentos será executado. Depois que os dados foram selecionados, a etapa de *pré-processamento e transformação* é executada, nessa etapa é realizada a limpeza dos dados, ou seja, ruídos são removidos dos mesmos, um exemplo de limpeza pode ser descartar registros contendo atributos incompletos ou nulos. Ainda nesta fase os dados são transformados em um formato apropriado para a aplicação de algoritmos de mineração. O objetivo principal do processo KDD é realizado na terceira etapa, a ***mineração de dados***, onde os algoritmos de mineração como os de regras de associação, classificação, clusterização, padrões seqüenciais, entre outros, são executados. A última etapa é a de *interpretação e validação dos resultados*, nesta etapa os padrões extraídos são mostrados de maneira compreensível ao usuário.

## 2.2 Tarefas de Mineração de Dados

Resumidamente, uma tarefa de mineração de dados consiste na especificação do que esta sendo procurado, que tipo de regularidades ou padrões devem ser encontrados, por exemplo, comportamentos dos clientes de um supermercado, sintomas apresentados por pacientes antes ou após o uso de algum medicamento, gastos exagerados por um cliente de um banco, etc. Até hoje foram propostas várias tarefas de mineração de dados [3, 1, 22, 23, 24], e para cada tipo de tarefa vários algoritmos e formalismos foram criados para possibilitar a mineração. As principais tarefas de mineração são:

1. **Mineração de Regras de Associação:** Corresponde à descoberta de associações entre objetos. Dado um banco de dados de transações onde cada transação é um conjunto de objetos, uma regra de associação é uma expressão da forma $X \rightarrow Y$, onde, X e Y também são conjuntos de objetos. De um modo intuitivo, essa regra significa que transações do banco de dados que contém X tendem a conter Y. Por exemplo, é possível descobrir um conjunto de produtos de um supermercado que é freqüentemente vendido com outro conjunto de produtos, assim, uma regra de associação que reflete o comportamento dos clientes de um supermercado poderia ser: "Clientes que compram pão também compram leite". As informações que as regras de associação transmitem podem ser utilizadas, por exemplo, para melhorar a distribuição dos produtos nas prateleiras [22, 25, 26, 27].

2. **Classificação:** Consiste em encontrar modelos de classificação (regras) que separam os dados em classes distintas, os modelos de classificação são criados com base nas características dos dados de um banco de dados de treinamento, cujos elementos são chamados de amostras, onde se conhece as classes de cada objeto. Um conjunto de regras é gerado a partir do modelo de classificação com o propósito de classificar objetos que ainda não foram classificados. Por exemplo, se o objetivo fosse descobrir se um cliente é bom ou mau pagador, um modelo de classificação poderia conter a regra: "Clientes que pertencem a classe econômica B, com menos de 3 filhos são bons pagadores", esta conclusão é obtida com base em casos já conhecidos no banco de dados de treinamento e generalizada para novos casos [24, 28, 29].

3. **Agrupamento (*Cluster*):** Consiste em agrupar em classes objetos que possuem características semelhantes. A diferença deste tipo de mineração com a classificação é que na classificação os dados (objetos) de treinamento estão devidamente classificados, ou seja, distribuídos em classes pré-definidas. No agrupamento, as classes são criadas no decorrer do processo, objetos com maior similaridade são agrupados em uma mesma classe, enquanto que objetos com pouca similaridade são agrupados em classes distintas. Por exemplo, o agrupamento pode ser aplicado a um banco de dados de um supermercado para identificar grupos de clientes com algumas características em comum, como por exemplo clientes de determinada região da cidade vão ao supermercado uma vez por semana, enquanto que clientes de outra região vão ao supermercado duas ou mais vezes por semana [23, 30].

4. **Mineração de Padrões Seqüenciais:** Tem como principal objetivo encontrar seqüências de eventos (fatos) que ocorrem freqüentemente em bancos de dados. Por exemplo, em um banco de dados de transações de clientes, onde cada transação é constituída pelo identificador do cliente, data da transação e objetos envolvidos na transação, um padrão seqüencial é uma expressão da forma $< s_1, s_2, ..., s_n >$, onde, cada $s_i$ é um conjunto de objetos e a ordem dos conjuntos de objetos deve refletir a ordem cronológica em que os fatos ocorreram. Assim, a seqüência $< \{TV\}, \{DVD\ Player,\ Home\ Theater\} >$ significa que clientes que compram $TV$ em uma transação, posteriormente compram $DVDPlayer$ e $HomeTheater$ em outra transação [3, 11, 12, 31, 4, 32].

A mineração de padrões seqüenciais sobre bancos de dados temporais, conhecida também como *mineração de padrões temporais*, é a tarefa de mineração destacada como foco desta dissertação.

Algumas das propriedades e técnicas utilizadas na mineração de padrões temporais derivaram de um trabalho apresentado em 1994 por Agrawal e Srikant [25], onde eles apresentaram o clássico algoritmo *Apriori* para descoberta de *itemsets freqüentes* (conjuntos de itens freqüentes) em bancos de dados de transações de clientes. Com o objetivo de minerar padrões freqüentes de forma eficiente eles introduziram uma propriedade *an-*

*timonotônica* chamada *propriedade Apriori.*

**Propriedade *Apriori.*** *Um conjunto de itens não é freqüente se possui pelo menos um sub-conjunto de itens que não é freqüente, ou seja, todo sub-conjunto de itens freqüentes também é freqüente.*

O algoritmo *Apriori* é um algoritmo iterativo que em cada iteração produz *itemsets* candidatos maiores a partir de *itemsets* freqüentes da iteração anterior. Em cada iteração são executadas três etapas bem definidas: *Geração*, onde são gerados todos o padrões candidatos com base na iteração anterior; *Poda*, onde são podados todos os padrões candidatos que não satisfazem a propriedade *Apriori*; e *Avaliação*, onde o banco de dados é percorrido e o suporte de todos os padrões que satisfazem a propriedade *Apriori* é calculado. Os padrões que possuem suporte superior a um suporte mínimo estipulado pelo usuário são considerados freqüentes. Como será notado neste capítulo, a técnica de mineração desenvolvida para o algoritmo *Apriori* constitui a base de muitos algoritmos de mineração de padrões temporais. Está técnica é comumente referenciada como técnica *Apriori.*

A maioria dos algoritmos de mineração de dados restringem de alguma forma seus padrões. Existem vários tipos de restrições que podem ser introduzidas no processo de mineração, como por exemplo *restrições de geração*, que são incorporadas na etapa de geração dos algoritmos visando diminuir o espaço de busca dos padrões e *restrições de validação*, que são incorporadas na etapa de avaliação dos algoritmos e os padrões que não satisfazem essas restrições são eliminados. Detalhes de algumas restrições podem ser encontrados em [11, 33]. Nesta dissertação tem destaque as restrições de expressões regulares que são incorporadas tanto na etapa de geração quanto na etapa de poda dos algoritmos de mineração [12, 34, 35, 36].

Nas próximas seções serão descritos alguns dos principais trabalhos relacionados a mineração de padrões temporais. Serão apresentados diversos enfoques referentes a diferentes técnicas de mineração de padrões temporais e para cada um desses enfoques serão introduzidos os conceitos necessários, de acordo com os autores, para seu entendimento. É importante salientar que às vezes conceitos são definidos de forma diferente em cada enfoque, portanto, para não deixar o leitor confuso foram introduzidas referências em

todas as definições de outros autores apresentadas nesta dissertação.

Para facilitar o entendimento, os padrões temporais foram divididos em dois grupos principais, os que representam o tempo por pontos (seção 2.3) e os que representam o tempo por intervalos (seção 2.4).

## 2.3 Padrões Temporais que Representam o Tempo por Pontos

Padrões temporais que representam o tempo por pontos são aqueles em que os eventos ocorrem em determinados instantes (uma hora, uma data, etc.) e podem ser vistos como uma seqüência de eventos respeitando uma ordem cronológica. Esses padrões são conhecidos historicamente como *padrões seqüenciais*.

Existem padrões seqüenciais especificados pela Lógica Temporal Proposicional (LTP), chamados de padrões seqüenciais proposicionais, e padrões seqüenciais mais expressivos, especificados pela Lógica Temporal de Primeira Ordem (LTPO), chamados de padrões seqüenciais de primeira ordem. O problema de mineração de padrões seqüenciais proposicionais será abordado formalmente na próxima seção, já, o problema de mineração de padrões seqüenciais de primeira ordem será abordado na seção 2.3.3.

### 2.3.1 O Problema de Mineração de Padrões Seqüenciais Proposicionais

O problema de mineração de padrões seqüenciais tem sido amplamente estudado e é aplicado com sucesso em diversos setores, tal como no varejo (os padrões minerados podem representar seqüências de compras de clientes), no mercado financeiro (os padrões minerados podem representar a evolução de cotações de ações), na Internet (os padrões minerados podem representar os caminhos de páginas da web freqüentemente percorridos por usuários), etc. Um exemplo clássico da aplicação da mineração de padrões seqüenciais proposicionais pode ser dado como segue.

Suponha que o gerente de um supermercado esteja interessado em descobrir a evolução

de compras de seus clientes ao longo do tempo. Ele quer saber quais produtos são comprados por um mesmo cliente em diferentes momentos. Se ele descobre que a seqüência de produtos $< \{p_1, p_2\}\ p_3 >$ é freqüentemente comprada nessa mesma ordem, ou seja, clientes compram os produtos $p_1$ e $p_2$ em um mesmo momento e o produto $p_3$ em um momento futuro, então poderá realizar uma campanha de marketing colocando o produto $p_3$ em promoção para os clientes que comprarem os produtos $p_1$ e $p_2$. Com isso ele terá grandes chances de não desperdiçar recursos em vão, pois sabe que os clientes têm grandes chances de comprar o produto $p_3$.

O padrão seqüencial $< \{p_1, p_2\}\ p_3 >$, apresentado no parágrafo anterior, pode ser expressado na LTP pela fórmula $(C_{p_1} \wedge C_{p_2})\ \Diamond\ C_{p_3}$, onde, para cada $p_i$ com $i \in \{1, 2, 3\}$, $C_{p_i}$ é um símbolo proposicional que significa *"clientes compram o produto $p_i$"*. O símbolo $\Diamond$ é o operador temporal *"em algum momento no futuro"*.

O problema de mineração de padrões seqüenciais proposicionais foi introduzido em 1995 por Agrawal e Srikant em [3]. Nesse trabalho foram propostos vários algoritmos para a mineração de padrões seqüenciais sobre um banco de dados de transações de clientes. Serão definidos a seguir alguns conceitos fundamentais para o entendimento deste problema de mineração.

**Definição 2.3.1 (Seqüência [3])**. Uma **seqüência** $s$ é uma lista ordenada denotada por $<s_1 s_2 \ldots s_n>$, onde $s_i$ é um *itemset* definido sobre um conjunto de itens $I$. O **comprimento** de uma seqüência $s$, corresponde ao número de *itens* na seqüência. Uma seqüência de comprimento $k$ é referenciada como uma $k$-seqüência. Se um *item* ocorre múltiplas vezes em diferentes elementos de uma seqüência, cada ocorrência contribui para o valor de $k$.

**Exemplo 2.3.1** Considere o conjunto $I = \{TV, DVD, Video, HomeTheater, Ventilador, Fogão\}$ e uma seqüência $s = <\{TV, Video\}\{Ventilador\}\{TV, DVD\}>$. A seqüência $s$ possui comprimento 5, pois, possui 5 *itens*. Nesse caso os *itemsets* representam itens comprados em uma mesma transação.

**Definição 2.3.2 (Sub-seqüência [3])**. Uma seqüência $a = <a_1 a_2 \ldots a_n>$ é uma **sub-**

**seqüência** de uma seqüência $b = <b_1 b_2 \ldots b_m>$, com $m \geq n$, denotado por $a \sqsubseteq b$, se existe inteiros $i_1 < i_2 < \ldots < i_n$ tal que $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, ..., $a_n \subseteq b_{i_n}$. Também pode-se dizer que $b$ *contêm* $a$ ou que $a$ *está contida em* $b$.

**Exemplo 2.3.2** Considere as seqüências $s = <\{TV, Video\}\ \{Ventilador\}\ \{TV, DVD, Fogão\}>$, $s_1 = <\{TV\}\ \{Ventilador\}\ \{TV, DVD\}>$, $s_2 = <\{TV, Video\}\ \{TV, DVD, Fogão\}>$ e $s_3 = <\{TV\}\ \{Fogão\}\ \{DVD\}>$. As seqüências $s_1$ e $s_2$ são sub-seqüências de $s$, enquanto que, a seqüência $s_3$ não é.

**Definição 2.3.3 (Seqüência maximal [3])**. Seja $S$ um conjunto de seqüências, uma seqüência $s \in S$ é **maximal** se não existe nenhuma seqüência $s' \in S - \{s\}$, tal que, $s \sqsubseteq s'$.

**Exemplo 2.3.3** Considere o conjunto de seqüências $S = \{< \{TV\}\ \{Ventilador\}\ \{DVD, Fogão\}\ >, < \{TV, DVD\}\ \{Fogão\}\ >, < \{Ventilador\}\ \{Fogão\}\ >\}$. A seqüência $< \{TV, DVD\}\ \{Fogão\}\ >$ é maximal, pois, ela não está contida em nenhuma outra seqüência de $S$. A seqüência $< \{Ventilador\}\ \{Fogão\}\ >$ não é maximal, pois, está contida na primeira seqüência de $S$.

**Definição 2.3.4 (Banco de dados de seqüências [3])**. Um **banco de dados de seqüências** $BD$ é um conjunto de tuplas $< ids, s >$, onde $s$ é uma seqüência e $ids$ é o identificador da seqüência $s$ . Uma tupla $< ids, s >$ em $BD$ *contém* uma seqüência $a$, se $a$ é uma sub-seqüência de $s$, ou seja, $a \sqsubseteq s$.

**Exemplo 2.3.4** Considere o banco de dados $BD$ de seqüências, constituído de quatro tuplas, mostrado na tabela 2.1. Nesse caso o identificador das seqüências é o próprio código do cliente que comprou os itens pertencentes a seqüência.

| *IdCliente* | *Seqüência do cliente* |
|---|---|
| 01 | $<\{TV, DVD\}\{Fogão\}>$ |
| 02 | $<\{Ventilador\}\{Fogão\}>$ |
| 03 | $<\{TV\}\{Ventilador\}\{DVD, Fogão\}>$ |
| 04 | $<\{DVD\}\{Fogão\}>$ |

Tabela 2.1: Exemplo de um banco de dados de seqüências de clientes

**Definição 2.3.5 (Suporte [3])**. O **suporte** de uma seqüência $a$ em um banco de dados de seqüências $BD$ é denotado por $sup(a) = \frac{|\{s \mid \exists\ ids,\ <ids,s> \in BD\ e\ a \sqsubseteq s\}|}{|BD|}$, ou seja, $sup(a)$ é a porcentagem de tuplas em $BD$ que contém $a$.

**Definição 2.3.6 (Seqüência freqüente [3])**. Dado um suporte mínimo $\alpha$, uma seqüência $s$ é **freqüente** em um banco de dados $BD$ se $sup(s)$ é maior ou igual a $\alpha$, ou seja, $sup(s) \geq \alpha$. Uma seqüência freqüente é chamada de **padrão seqüencial**.

**Exemplo 2.3.5** Considere o banco de dados da figura 2.1. O suporte da seqüência $< \{TV\}\{Fogão\}>$ é 0.5 (50%), pois, ela está contida nas seqüências dos clientes 01 e 03. Se for considerado um suporte mínimo de 50% (duas tuplas), então, $<\{TV\}\{Fogão\}>$ é considerada uma seqüência freqüente.

Com os principais conceitos apresentados, a formulação geral do problema de mineração de padrões seqüênciais proposicionais é dada por:

- **Dado:** Um banco de dados de seqüências $BD$ e um suporte mínimo $\alpha$.
- **Encontrar:** Todas as seqüências freqüentes em relação a $BD$ e $\alpha$.

Vários pesquisadores contribuíram e ainda estão contribuindo para tornar a mineração de padrões seqüenciais proposicionais cada vez mais eficiente. Uma grande variedade de algoritmos já foram propostos, entre eles se destacam o AprioriAll [3], o GSP [11], o SPADE [10], o FreeSpan [32] e o PrefixSpan [31]. O GSP foi apresentado por Srikant e Agrawal em [11] e minera os mesmos tipos de padrões seqüênciais minerados pelo algoritmo AprioriAll [3], embora com performance bem superior. Por este motivo, as principais idéias do algoritmo GSP serão apresentadas a seguir.

**O Algoritmo GSP.** O algoritmo GSP é baseado na técnica *Apriori*, portanto, trabalha em iterações e em cada iteração $k$ gera $k$-seqüências freqüentes. Em cada iteração são executadas três etapas: *geração*, *poda* e *avaliação* (cálculo do suporte). Na primeira iteração são encontrados todos os padrões freqüentes contendo apenas um item (1-seqüência). Estes padrões são utilizados na segunda iteração para o gerar as 2-seqüências candidatas. Se $<\ \{\ TV\ \}\ >$ e $<\ \{\ DVD\ \}\ >$ são 1-seqüências freqüentes, então as 2-seqüências

$<\{TV, DVD\}>$, $<\{TV\}\{DVD\}>$ e $<\{DVD\}\{TV\}>$ são seqüências candidatas. A seguir, o banco de dados é percorrido e o suporte de todas as seqüências candidatas é calculado. Todas as seqüências com suporte inferior a um suporte mínimo estipulado pelo usuário são podadas e as seqüências restantes são consideradas freqüentes. A partir da segunda iteração as seqüências são geradas pela combinação das seqüências freqüentes da iteração anterior, em seguida as seqüências que possuem sub-seqüências que não são freqüentes são podadas (propriedade *Apriori*), então, o suporte das seqüências que restaram é calculado. Resumidamente, cada etapa do algoritmo GSP funciona assim:

1. **Geração**: Considere os conjuntos $F_k$ e $C_k$, que denotam o conjunto das $k$-seqüências freqüentes e o conjunto das $k$-seqüências candidatas, respectivamente.

   Dado um conjunto $F_{k-1}$ constituído por duas ($k$-1)-seqüências $(s_1, s_2)$, se essas duas seqüências podem ser *combinadas* para dar origem a uma $k$-seqüência, então a $k$-seqüência resultante da combinação são inseridas em $C_k$. As duas seqüências podem ser *combinadas* se a retirada do primeiro item de $s_1$ e do último item de $s_2$ resulta em seqüências iguais. Neste caso, o último item de $s_2$ é acrescentado em $s_1$ formando uma seqüência em $C_k$. Se o último item de $s_2$ estava em um *itemset* separado, ele será inserido em um *itemset* separado em $s_1$, caso contrário, ele fará parte do último *itemset* de $s_1$. Por exemplo, seja $F_3 = \{<\{TV, Video\}\{DVD\}>, <\{Video\}\{DVD, Fogão\}>\}$, as duas seqüências de $F_3$ podem ser combinadas, pois, retirando o item $TV$ da primeira e $Fogão$ da segunda seqüência, as seqüências resultantes são iguais. Assim, a seqüência obtida é a 4-seqüência $<\{TV, Video\}\{DVD, Fogão\}>$ que é inserida em $C_4$.

2. **Poda**: São podados todas as seqüências de $C_k$ que possuem pelo menos uma subseqüência não pertence a $F_{k-1}$, ou seja, que não é freqüente, restando somente seqüências potencialmente freqüentes em $C_k$. Continuando o exemplo apresentado na Geração, a seqüência $<\{TV, Video\}\{DVD, Fogão\}>$ será podada, pois, a subseqüência $<\{TV\}\{DVD, Fogão\}>$ não pertence a $F_3$.

3. **Avaliação**: O suporte das seqüências que restaram em $C_k$ após a poda é calculado percorrendo o banco de dados uma vez. As seqüências candidatas que possuírem

suporte maior ou igual a um suporte $\alpha$, definido pelo usuário, são inseridas em $F_k$. Nos algoritmos de mineração esta é a etapa mais cara computacionalmente, por isso, deve ser otimizada. O GSP utiliza recursos como uma árvore *hash* para otimizá-la. Detalhes dessa árvore *hash* podem ser encontrados em [11].

### 2.3.2 O Problema de Mineração de Padrões Seqüenciais Proposicionais com Restrições de Expressões Regulares

Grande parte dos algoritmos de mineração de padrões seqüenciais utilizam um *suporte mínimo* definido pelo usuário como única forma de restringir o espaço de busca dos padrões. Estes algoritmos não permitem que os tipos dos padrões sejam especificados, assim, muitos dos padrões minerados são irrelevantes para o usuário.

As restrições de expressões regulares são introduzidas no processo de mineração com o objetivo de reduzir o espaço de busca dos padrões e servir como ferramenta para que o usuário possa informar os tipos de padrões que ele deseja obter, diminuindo assim o tempo gasto na mineração de padrões que não o interessam.

Suponha que o usuário esteja interessado somente em padrões que começam com o item $TV$ e terminam com o item $DVD$. Para isso, ele pode definir a expressão regular $(TV)\ (s)^*\ (DVD)$, onde, $s$ representa uma seqüência qualquer de itens e $^*$ é um símbolo que indica que a seqüência $s$ pode não aparecer ou aparecer uma ou mais vezes.

A maioria dos algoritmos para mineração de padrões seqüenciais com restrições de expressões regulares também utilizam a técnica *Apriori*. Em uma iteração $k$, é obtido um conjunto $C_k$ constituído por $k$-seqüências candidatas a partir das ($k$-1)-seqüências do conjunto $F_{k-1}$, obtido na iteração anterior. Em seguida, são podadas de $C_k$ as $k$-seqüências que possuem pelo menos uma ($k$-1)-seqüência não pertencente a $F_{k-1}$. A diferença entre o problema clássico de mineração de seqüências e o problema de mineração em questão, é que neste são geradas somente $k$-seqüências candidatas que satisfazem uma expressão regular $\mathcal{R}$. Na etapa de poda, não são podadas diretamente de $C_k$ as $k$-seqüências que possuem pelo menos uma ($k$-1)-seqüência não pertencente a $F_{k-1}$, pois, o conjunto $F_{k-1}$ contém **seqüências freqüentes** e que **satisfazem a expressão regular** $\mathcal{R}$. Esta restrição é

considerada porque a **condição de ser freqüente é antimonotônica**, e a **condição de satisfazer a expressão regular $\mathcal{R}$ não é antimonotônica**. Por exemplo, a seqüência $<\{TV\}\{DVD\}\{DVD\}>$ satisfaz a expressão regular $(TV)(DVD)^*$, mas, a sub-seqüência $<\{DVD\}\{DVD\}>$ não satisfaz. Portanto, uma seqüência $s$ só pode ser podada de $C_k$ se ela possui uma sub-seqüência $s'$ que satisfaz $\mathcal{R}$ e que não pertence a $F = F_1 \cup F_2 \cup \ldots \cup F_{k-1}$, pois, nesse caso com certeza $s$ não é freqüente.

Com base no que foi que foi dito anteriormente é possível afirmar que quanto maior for a seletividade de $\mathcal{R}$ menos padrões são gerados e menos padrões são podados, o que não é interessante, uma vez que um dos objetivos deste tipo de restrição é diminuir a quantidade de padrões que serão levados para a etapa de avaliação, etapa esta, considerada a mais cara computacionalmente na maioria dos algoritmos de mineração de dados.

Para tentar achar um equilíbrio entre a quantidade de padrões gerados e a quantidade de padrões podados, alguns algoritmos, como por exemplo, os da família SPIRIT [12] e SPIRIT-Log [14], consideram um *relaxamento* de $\mathcal{R}$, ou seja, uma restrição $\mathcal{R}'$ *mais fraca* que $\mathcal{R}$, durante o processo de mineração.

A formulação geral do problema de mineração de padrões seqüenciais proposicionais com restrições de expressões regulares é dada por:

- **Dado:** Um banco de dados $BD$, um suporte mínimo $\alpha$ e uma expressão regular $\mathcal{R}$.
- **Encontrar:** Todas as seqüências freqüentes com relação a $BD$, $\alpha$ e que satisfazem $\mathcal{R}$.

Garofalakis e Shim [12] foram os primeiros a introduzirem restrições de expressões regulares no processo de mineração. Eles desenvolveram os algoritmos da família SPIRIT para mineração de padrões seqüenciais proposicionais e incorporam este tipo de restrição no processo de mineração.

**Os Algoritmos SPIRIT.** A família de algoritmos SPIRIT é constituída por quatro algoritmos que exploram diferentes graus de seletividade de uma expressão regular utilizada nas etapas de geração e poda dos padrões candidatos.

Os quatro algoritmos da família SPIRIT são: SPIRIT(N), SPIRIT(L), SPIRIT(V) e SPIRIT(R). Cada um deles considera um relaxamento $\mathcal{R}'$ cuja seletividade aumenta na seguinte ordem: SPIRIT(N) $<$ SPIRIT(L) $<$ SPIRIT(V) $<$ SPIRIT(R). Estes algoritmos associam um autômato $A_{\mathcal{R}}$ à expressão regular $\mathcal{R}$ no processo de mineração.

- **SPIRIT(N):** não impõe nenhuma restrição às seqüências, apenas exige que todos os itens de uma seqüência $s$ apareçam em $\mathcal{R}$.
- **SPIRIT(L):** considera somente seqüências *legais* com respeito a algum estado do autômato $A_{\mathcal{R}}$. Uma seqüência $s =< s_1 s_2 \ldots s_n >$ é *legal* com respeito a um estado $q$ de $A_{\mathcal{R}}$ se existe um caminho no autômato que começa no estado $q$ e percorre a palavra $s_1 s_2 \ldots s_n$.
- **SPIRIT(V):** considera somente seqüências *válidas* com respeito a algum estado do autômato $A_{\mathcal{R}}$. Uma seqüência $s =< s_1 s_2 \ldots s_n >$ é *válida* com respeito a um estado $q$ de $A_{\mathcal{R}}$ se existe um caminho no autômato que começa no estado $q$ e termina em um estado final e que percorre a palavra $s_1 s_2 \ldots s_n$.
- **SPIRIT(R):** não considera relaxamentos, corresponde exatamente a $\mathcal{R}$. Considera somente seqüências *válidas*, ou seja, aquelas que começam no estado inicial e terminam em um estado final de $A_{\mathcal{R}}$.

Estes quatro algoritmos encontram seqüências freqüentes que satisfazem o relaxamento $\mathcal{R}'$ da expressão regular $\mathcal{R}$. Em uma etapa de pós-processamento, eles eliminam de $F = F_1 \cup \ldots \cup F_k$ os padrões que não satisfazem a expressão regular $\mathcal{R}$ original. O SPIRIT(R), por não considerar um relaxamento de $\mathcal{R}$, não realiza a etapa de pós-processamento.

Garofalakis e Shim avaliaram os quatro algoritmos e chegaram a conclusão de que o algoritmo SPIRIT(V) é o mais eficiente dos quatro, por esse motivo, ele será o único a ser detalhado. Detalhes dos outros algoritmos podem ser encontrados em [12].

**O Algoritmo SPIRIT(V).** Considere um autômato $A_{\mathcal{R}}$ associado a uma expressão regular $\mathcal{R}$. $C_k(q)$ e $F_k(q)$ correspondem, respectivamente, ao conjunto das $k$-seqüências candidatas e das $k$-seqüências freqüentes e *válidas* com respeito ao estado $q$ de $A_{\mathcal{R}}$.

1. **Geração:** Para que uma seqüência $s = <s_1 s_2 \ldots s_k>$ seja válida com respeito a algum estado $q_1$ de $A_{\mathcal{R}}$, é preciso que seu sufixo $<s_2 \ldots s_k>$ seja válido com respeito a algum estado $q_2$ de $A_{\mathcal{R}}$ e que exista um transição $q_1 \stackrel{s_1}{\rightarrow} q_2$ (indo de $q_1$ para $q_2$, com rótulo $s_1$) em $A_{\mathcal{R}}$. Portanto, o conjunto $C_k$ é constituído como segue. Para cada estado $q$ de $A_{\mathcal{R}}$, onde $q \stackrel{s_i}{\rightarrow} q'$, para cada seqüência $<s_1 \ldots s_{k-1}>$ de $F_{k-1}(q')$, a seqüência $<s_i\ s_1 \ldots s_{k-1}>$ é inserida em $C_k(q)$. O conjunto $C_k$ é a união desses conjuntos para cada estado $q$ de $A_{\mathcal{R}}$.

   **Exemplo 2.3.6** Considere o autômato $A_{\mathcal{R}}$ da figura 2.2 associado a uma expressão regular $\mathcal{R}$ e um conjunto de 2-seqüências, freqüentes e válidas com respeito aos estados de $A_{\mathcal{R}}$, $F_2 = F_2(q_0) \cup F_2(q_1) \cup F_2(q_3)$, onde, $F_2(q_0) = \{<a\ c>, <a\ e>\}$, $F_2(q_1) = \{<b\ c>\}$, $F_2(q_2) = \{<d\ e>\}$ e $F_2(q_3) = \{\emptyset\}$. O conjunto $C_3$ das 3-seqüências candidatas e válidas com relação a $A_{\mathcal{R}}$ é calculado da seguinte forma:

   1) Considerando as transições partindo de $q_0$:

      (a) $q_0 \stackrel{a}{\rightarrow} q_1$: como $F_2(q_1) = \{<b\ c>\}$, a 3-seqüência $<a\ b\ c>$ é construída.

      (b) $q_0 \stackrel{a}{\rightarrow} q_2$: como $F_2(q_2) = \{<d\ e>\}$, a 3-seqüência $<a\ d\ e>$ é construída.

      Temos, $C_3(q_0) = \{<a\ b\ c>, <a\ d\ e>\}$.

   2) Considerando as transições partindo de $q_1$:

      (a) $q_1 \stackrel{b}{\rightarrow} q_1$: como $F_2(q_1) = \{<b\ c>\}$, a 3-seqüência $<b\ b\ c>$ é construída.

      (b) $q_1 \stackrel{c}{\rightarrow} q_3$: como $F_2(q_3) = \emptyset$, nenhuma 3-seqüência é construída.

      Temos, $C_3(q_1) = \{<b\ b\ c>\}$.

   3) Considerando as transições partindo de $q_2$:

      (a) $q_2 \stackrel{d}{\rightarrow} q_2$: como $F_2(q_2) = \{<d\ e>\}$, a 3-seqüência $<d\ d\ e>$ é construída.

      (b) $q_2 \stackrel{e}{\rightarrow} q_3$: como $F_2(q_3) = \emptyset$, nenhuma 3-seqüência é construída.

      Temos, $C_3(q_2) = \{<d\ d\ e>\}$.

   Logo, o conjunto das 3-seqüências candidatas e válidas com relação a $A_{\mathcal{R}}$ é dado por $C_3 = C_3(q_0) \cup C_3(q_1) \cup C_3(q_2) = \{<a\ b\ c>, <a\ d\ e>, <b\ b\ c>, <d\ d\ e>\}$.

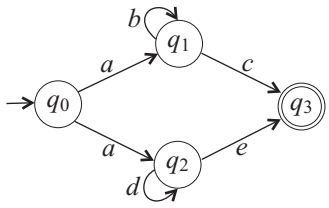



Figura 2.2: Autômato $A_{\mathcal{R}}$ associado a uma expressão regular $\mathcal{R}$

2. **Poda:** Para cada seqüência $s$ de $C_k$ são calculadas todas as sub-seqüências maximais de $s$ que são válidas com respeito a algum estado de $A_{\mathcal{R}}$, ou seja, as maiores sub-seqüências possíveis menores que $k$. Se algumas dessas seqüências não está em $F = F_1 \cup \ldots \cup F_{k-1}$, $s$ é podada de $C_k$.

### 2.3.3 O Problema de Mineração de Padrões Seqüenciais de Primeira Ordem

Padrões seqüenciais de primeira ordem são mais expressivos que padrões seqüenciais proposicionais e podem ser minerados em diversos domínios de aplicação. Por exemplo: padrões minerados podem representar seqüências de comandos de usuários UNIX ou seqüências de páginas Web acessadas por usuários ou seqüências de compras de clientes ao longo do tempo.

Um exemplo da mineração deste tipo de padrão pode ser o seguinte: Suponha que o administrador de um departamento de Tecnologia da Informação (TI) de uma companhia esteja interessado em descobrir seqüências de comandos do Unix que são freqüentemente utilizados pelos funcionários do departamento. O administrador de TI pode considerar o log do Unix que armazena seqüências de comandos do tipo $ls, vi\ artigo, mv\ artigo, latex\ artigo, rm\ artigo$ e converter cada seqüência de comandos em uma seqüência de átomos de primeira ordem da forma $ls$, $vi(artigo)$, $mv(artigo)$, $latex(artigo)$, $rm(artigo)$, possibilitando a descoberta de padrões da forma $<vi(X)\ rm(X)>$ , onde, $vi\ rm$ é uma a seqüência de comandos (predicados) freqüentemente requerida pelos usuários e $X$ é uma variável representando nomes de arquivos (parâmetro).

O problema de mineração de padrões seqüenciais de primeira ordem foi amplamente estudado por Jacobs e Blockeel em [17], onde propuseram formalismos para mineração

destes padrões sobre logs de usuários do sistema Unix. Outro importante trabalho que contribuiu no estado da arte de mineração de padrões seqüenciais de primeira ordem foi desenvolvido em 2002 por Lee e Raedt [13]. Neste trabalho eles apresentaram um formalismo lógico baseado em Programação Lógica Indutiva (PLI), chamado SeqLog, para especificar padrões seqüenciais de primeira ordem. A seguir serão descritos alguns conceitos, apresentados em [13], fundamentais para o entendimento deste problema. Neste caso são consideradas terminologias tradicionais da lógica de primeira ordem e como norma, as variáveis são designadas por letras maiúsculas e as constantes por letras minúsculas.

**Definição 2.3.7 (Átomo [13])**. Um **átomo** definido por $p(t_1, t_2, \ldots, t_n)$, consiste em um predicado $p$ seguido por $n$ termos $t_i$, onde, $t_i$ é uma variável ou uma constante.

**Definição 2.3.8 (Seqüência simples [13])**. Uma **seqüência simples** $s$ é uma lista ordenada de átomos denotada por $<s_1 s_2 \ldots s_n>$, onde, $s_i$ é o i-ésimo elemento de $s$. O *comprimento* de uma seqüência $s$ corresponde ao número de átomos de $s$. Esse tipo de seqüência é conhecida como seqüência lógica.

**Exemplo 2.3.7** Uma seqüência simples de comprimento 3 pode ser dada por $<vi(artigo)$ $mv(artigo)$ $latex(artigo)>$, onde, $vi, mv$ e $latex$ são predicados e $artigo$ é uma constante e o átomo $latex(artigo)$ ocorre após o átomo $mv(artigo)$ que ocorre após o átomo $vi(artigo)$.

**Definição 2.3.9 (Seqüência complexa [13])**. Uma **seqüência complexa** $s$ é uma lista ordenada de átomos separados por operadores denotada por $<s_1\ op_1\ s_2\ op_2\ \ldots\ op_{n-1}\ s_n>$. Os dois operadores empregados são o operador sucessor direto $\lhd$ (que é freqüentemente omitido) e seu fecho transitivo $<$. As seqüências complexas são utilizadas para representar padrões seqüenciais.

**Exemplo 2.3.8** Uma seqüência complexa (padrão seqüencial) pode ser dada por $<vi(artigo) < mv(artigo)>$, que significa que o átomo $mv(artigo)$ ocorre em algum lugar após o átomo $vi(artigo)$.

**Definição 2.3.10 (Sub-seqüência [13])**. Uma seqüência $a = <a_1 a_2 \ldots a_n>$ é uma **sub-seqüência** de uma seqüência $b = <b_1 b_2 \ldots b_m>$, com $m \geq n$, se existe inteiros $i_1 < i_2 < \ldots < i_n$ tal que $a_1 = b_{i_1}, a_2 = b_{i_2}, \ldots, a_n = b_{i_n}$.

**Exemplo 2.3.9** A seqüência $s_1 = <ls\ vi(X)>$ é uma sub-seqüência de $s_2 = <ls\ vi(X)\ mv(X)\ latex(X)>$ e $s_3 = <ls\ vi(X)\ mv(X)>$ é uma sub-seqüência de $s_2$. Entretanto, $s_3$ não é uma sub-seqüência de $s_4 = <ls\ vi(X)\ latex(X)\ rm(X)>$, pois, $mv(X) \neq latex(X)$ e $mv(X) \neq rm(X)$.

**Definição 2.3.11 (Substituição [13])**. Uma substituição $\theta$ é um conjunto da forma $\{v_1 \mapsto t_1, \ldots, v_n \mapsto t_n\}$, onde, $v_i$ são variáveis e $t_i$ são variáveis ou constantes. Seja $s = <s_1 \ldots s_n>$ uma seqüência, $\theta s$ denota a aplicação de $\theta$ em $s$, assim, $\theta s = <\theta s_1 \ldots \theta s_n>$.

**Exemplo 2.3.10** A forma $\theta = \{X \mapsto z\}$, é uma substituição. Já a forma $\theta' = \{z \mapsto X\}$ não é uma substituição, pois, $z$ não é uma variável. Quando a substituição $\theta$ é aplicada em um seqüência $s = <vi(X)\ mv(X)>$ obtemos uma seqüência $s' = <vi(z)\ mv(z)>$.

**Definição 2.3.12 (Subsunção [13])**. É dito que uma seqüência $a$ **subsume** uma seqüência $b$, denotado por $a \sqsubseteq b$, se existe uma substituição $\theta$, tal que, $\theta a$ é uma sub-seqüência de $b$. É dito que $a$ é *mais geral* que $b$, ou que $b$ é mais específica que $a$.

**Exemplo 2.3.11** Considere as seqüências $s_1 = <vi(X)\ mv(Y)>$ e $s_2 = <vi(z)\ mv(t)\ mr(w)>$. $s_1$ subsume $s_2$, se for considerada a substituição $\theta = \{X \mapsto z, Y \mapsto t\}$.

**Definição 2.3.13 (Banco de dados de seqüências [13])**. Em um banco de dados de seqüências $BD$ cada seqüência é uma lista ordenada de átomos com constantes em todos os termos. Essas seqüências são chamadas de seqüências de entrada, diferente das seqüências dos padrões, chamadas de seqüências lógicas.

As definições de **suporte** e **seqüência freqüente** são similares as apresentadas nas definições 2.3.5 e 2.3.6, respectivamente.

**Exemplo 2.3.12** Considere a seqüência $s = <vi(X) < mv(Y)>$ e o banco de dados $BD$ = $\{<vi(a)\ mv(b)\ rm(c)>, <vi(a)\ rm(c)\ mv(b)>, <vi(a)\ rm(c)>\}$. Note que a seqüência $s$ subsumes (está contida) duas seqüências de $BD$, portanto, $sup(s) = \frac{2}{3} = 0.66\%$. Se for considerado um suporte mínimo $\alpha = 50\%$, a seqüência $s$ é freqüente.

Com os principais conceitos entendidos, a formulação geral do problema da mineração de padrões seqüenciais de primeira ordem é definida por:

- **Dado:** Um banco de dados de seqüência $BD$ e um suporte mínimo $\alpha$.
- **Encontrar:** Todas as seqüências freqüentes em relação a $BD$ e $\alpha$.

**O SeqLog.** O SeqLog é um formalismo para expressar padrões seqüenciais de primeira ordem freqüentes em bancos de dados de seqüências. Como definido, Lee e Raedt distinguem *seqüências simples* (não possuem operador) de *seqüências complexas* (cujos átomos são relacionados pelo operador $<$) que são utilizadas para representar padrões seqüenciais. As seqüências podem ser constituídas por átomos, por exemplo, da forma $p(f(a, X), b, Y)$, onde, $p$ é um predicado, $f$ é uma função de aridade 2, $a$ e $b$ são constantes e $X$ e $Y$ são variáveis. Uma característica desse formalismo é o uso restrições de freqüência da forma $a \wedge m$, onde, $a$ é uma restrição antimonotônica, e $m$ é uma restrição monotônica. Maiores detalhes sobre estas propriedades do SeqLog podem ser encontradas em [13, 37].

Ainda em [13], Lee e Raedt apresentaram um algoritmo, chamado MineSeqLog, desenvolvido com base no SeqLog para a mineração de padrões seqüenciais de primeira ordem.

**O Algoritmo MineSeqLog.** O algoritmo MineSeqLog foi desenvolvido para minerar seqüências freqüentes expressas em SeqLog e também trabalha em iterações, gerando padrões mais específicos a cada iteração.

Além do banco de dados e das restrições de freqüência, o algoritmo MineSeqLog exige como entrada um conjunto $SF$ com nomes de predicados e suas aridades. Na primeira iteração é gerado um conjunto de padrões candidatos $C_1$, constituído por átomos gerados a partir de $SF$. Então o suporte de cada elemento de $C_1$ é calculado percorrendo o banco de dados. Na iteração $k$ são aplicados operadores de refinamento nos padrões de $F_{k-1}$ gerando assim padrões mais específicos que são inseridos em $C_k$. O suporte dos padrões de $C_k$ é calculado e os padrões freqüentes são inseridos em $F_k$. No final da execução o conjunto $F = F_0 \cup \ldots \cup F_i$, onde, $i$ é a última iteração, contém todos os padrões freqüentes. O algoritmo retorna todos os padrões maximais de $F$, para isso, são podados todos os sub-padrões de padrões em $F$. A fase da geração é executada como segue.

**Geração**: Na iteração $k$ são gerados padrões mais específicos a partir dos padrões contidos em $F_{k-1}$, gerado na iteração anterior. Para refinar (especializar) os padrões, são utilizados quatro operadores: **Lengyhening**, **Promotion**, **UniVar** e **Instantiation**. Esses operadores foram criados de forma que sejam *optimais*, ou seja, os padrões são gerados por um único caminho, isso evita que sejam gerados padrões duplicados, e *completos*, ou seja, dado um padrão $p$, são gerados todos os padrões aos quais $p$ é sub-seqüência, garantindo que não seja perdido nenhum padrão que possa satisfazer as restrições de freqüência. Em cada iteração, somente um dos quatro operadores de refinamento é aplicado sobre um padrão, garantindo que ele seja refinado iteração por iteração aumentando em um seu tamanho.

Cada padrão $q$ tem associado a ele um vetor $v(q)$ definido como um vetor 4-dimensional da forma $(l, p, u, i)$, onde, $l$ é o número de predicados em $q$, $p$ é número de operadores $\lhd$ em $q$, $u$ é o número de argumentos em $q$ menos o número de variáveis distintas (sem considerar argumentos constantes) em $q$ menos o número de constantes em $q$ e $i$ é o número de constantes em $q$. O *nível de refinamento* de $q$ é $|v(q)| = l + p + u + i$. Por exemplo, se $|v(q)| = 1$, então, $q$ pertence a iteração 1, se $|v(q)| = 2$, então, $q$ pertence a iteração 2.

Os quatro operadores devem obedecer algumas restrições para que as duas propriedades (optimal e completo) sejam garantidas. O operador *Lengthening* é aplicado em um padrão $q$ somente se $v(q) = (l, 0, 0, 0)$. Ele insere um novo átomo com o operador $<$ no fim da seqüência resultando em $v(q') = (l + 1, 0, 0, 0)$. O operador *Promotion* é aplicado em um padrão $q$ somente se $v(q) = (l, p, 0, 0)$. Ele promove um operador $<$ em $\lhd$ (que é freqüentemente omitido), resultando em $v(q') = (l, p + 1, 0, 0)$. O operador $UniVar$ é aplicado em um padrão $q$ somente se $v(q) = (l, p, u, i)$. Ele unifica duas variáveis, mas a variável escolhida para ser unificada não pode ser seguida por nenhuma outra variável que já tenha sido unificada, resultando em $v(q') = (l, p, u + 1, i)$. O operador *Instantiation* é aplicado em um padrão $q$ somente se $v(q) = (l, p, 0, i)$. Ele substitui uma variável por uma constante, mas nenhuma variável pode ser instanciada a esquerda de outra variável já instanciada, resultando em $v(q') = (l, p, 0, i + 1)$.

**Exemplo 2.3.13** Considere um padrão $q_7 = a(X)\ b(Y,g) < c(h,X)$, onde, $v(q_7) = (3,1,2,1)$. Esse padrão pode ser obtido a partir de um padrão $q_0 = \emptyset$, onde, $v(q_0) = (0,0,0,0)$. Primeiramente o operador *Lengthening* é aplicado em $q_0$ obtendo $q_1 = a(X_1)$, onde, $v(q_1) = (1,0,0,0)$. Aplicando *Lengthening* em $q_1$ pode-se obter $q_2 = a(X_1) < b(Y,G)$, onde $v(q_2) = (2,0,0,0)$. Aplicando-se novamente *Lengthening* em $q_2$ pode-se obter $q_3 = a(X_1) < b(Y,G) < c(H,X_2)$, onde $v(q_3) = (3,0,0,0)$. O padrão $q_4 = a(X_1)\ b(Y,G) < c(H,X_2)$, onde $v(q_4) = (3,1,0,0)$, pode ser obtido aplicando *Promotion* em $q_3$. Aplicando *Instantiation* em $q_4$ pode-se obter $q_5 = a(X_1)\ b(Y,g) < c(H,X_2)$, onde $v(q_5) = (3,1,1,0)$. Aplicando *Instantiation* em $q_5$ pode-se obter $q_6 = a(X_1)\ b(Y,g) < c(h,X_2)$, onde $v(q_6) = (3,1,2,0)$. Aplicando *UniVar* em $q_6$ obtem-se $q_7 = a(X)\ b(Y,g) < c(h,X)$, onde $v(q_7) = (3,1,2,1)$.

### 2.3.4 O Problema de Mineração de Padrões Seqüenciais de Primeira Ordem com Restrições de Expressões Regulares

Este problema de mineração é bem parecido com o apresentado na seção anterior. A diferença é que aqui são consideradas restrições de expressões regulares introduzidas no processo de mineração. Sua formulação geral é dada por:

- **Dado:** Um banco de dados $BD$, um suporte mínimo $\alpha$ e uma expressão regular $\mathcal{R}$.
- **Encontrar:** Todas as seqüências freqüentes em relação a $BD$, $\alpha$ e que satisfazem $\mathcal{R}$.

Contribuindo para a mineração de padrões seqüenciais de primeira ordem com restrições, um importante trabalho foi apresentado em 2003 por Masson e Jacquenet [14], onde foram introduzidos os algoritmos da família SPIRIT-Log que mineram os mesmos tipos de padrões que o MineSeqLog, mas, utilizam restrições de expressões regulares no processo de mineração.

**Os algoritmos SPIRIT-Log.** O algoritmo MineSeqLog utiliza restrições monotônicas e antimonotônicas no processo de mineração. Masson e Jacquenet não utilizaram esta

aproximação porque queriam incorporar restrições sintáticas para minerar seqüências lógicas e estas restrições não necessariamente são monotônicas e antimonotônicas. Por esse motivo adaptaram os algoritmos SPIRIT para a mineração de padrões seqüenciais de primeira ordem.

Similar à família SPIRIT, a família SPIRIT-Log é constituída por quatro algoritmos que trabalham em iterações e também exploram diferentes graus de seletividade de uma expressão regular $\mathcal{R}$ utilizada nas etapas de geração e poda dos padrões candidatos. A expressão regular $\mathcal{R}$ é definida pelo usuário. Durante o processo de mineração, um autômato $A_{\mathcal{R}}$ é associado a $\mathcal{R}$ e são consideradas somente seqüências aceitas por $A_{\mathcal{R}}$. Uma seqüência $s$ é aceita por $A_{\mathcal{R}}$ se a lista de predicados de $s$, correspondente a uma *string*, é aceita por $\mathcal{A}_R$. Por exemplo, considere o autômato da figura 2.3 associado a expressão regular $\mathcal{R} = mv\ cd^*\ (emacs|gcc\ gcc)$. A seqüência $< mv(F,D)cd(D)emacs(F)gcc(F,E) >$ é aceita por $A_{\mathcal{R}}$.

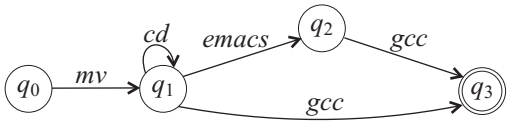



Figura 2.3: Automato $A_{\mathcal{R}}$ associado a expressão regular $mv\ cd^*\ (emacs|gcc\ gcc)$

Os quatro algoritmos principais da família SPIRIT-Log são SPIRIT-Log(N), SPIRIT-Log(L), SPIRIT-Log(V) e SPIRIT-Log(R). Cada um deles considera um relaxamento $\mathcal{R}'$ cuja seletividade aumenta na seguinte ordem: SPIRIT-Log(N) $<$ SPIRIT-Log(L) $<$ SPIRIT-Log(V) $<$ SPIRIT-Log(R). As restrições de cada um desses algoritmos são as mesmas dos da família SPIRIT e detalhes de cada uma foram apresentados na seção 2.3.2.

Masson e Jacquenet avaliaram esses quatro algoritmos e chegaram a conclusão de que o algoritmo SPIRIT-Log(L) é o mais eficiente, por isso, ele será o único que ser detalhado. Caso o leitor tenha interesse nos outros três algoritmos, consulte [14].

**O Algoritmo SPIRIT-Log(L).** Considere um autômato $A_{\mathcal{R}}$ associado a $\mathcal{R}$. $C_k(q)$ e $F_k(q)$ correspondem, respectivamente, ao conjunto das $k$-seqüências candidatas e das $k$-seqüências freqüentes que são *legais* com respeito ao estado $q$ de $A_{\mathcal{R}}$.

1. **Geração:** As $k$-seqüências lógicas candidatas para um estado $q$ são construídas como segue. Para toda seqüência $s \in F_{k-1}(q)$, se existe uma transição $q \xrightarrow{p} q'$ (indo de $q$ para $q'$, onde $p$ é o predicado no qual $s_1$ é construído) em $A_{\mathcal{R}}$, então, para toda seqüência $t$ em $F_{k-1}(q')$, $<s_2 \ldots s_{k-1}>$ e $<t_1 \ldots t_{k-2}>$ devem ser unificáveis, ou seja, deve existir uma substituição $\theta$, tal que $t_j = \theta s_{j+1}$ para $1 \leq j \leq k-2$. São adicionadas em $C_k$ todas as seqüências da forma $<\theta s\ \theta' t_{k-1}>$, onde $\theta'$ são todas as substituições possíveis das variáveis de $t_{k-1}$ por algumas variáveis de $s_1$.

   **Exemplo 2.3.14** Considere o autômato $A_{\mathcal{R}}$ da figura 2.4 e as 3-seqüências $s = <p(X,R)\ q(Y)\ r(X,S)> \in F_3(q_0)$ e $t = <q(L)\ r(N,T)\ s(J,T)> \in F_3(q_1)$. Considerando a substituição $\theta = \{Y \mapsto L, X \mapsto N, S \mapsto T\}$, o sufixo $<q(Y)\ r(X,S)>$ e o prefixo $<q(L)\ r(N,T)>$ podem ser unificados. Sendo assim, todas as substituições possíveis entre $(J,T)$ e $(X,R)$ são: $\{\epsilon, \{J \mapsto X\}, \{T \mapsto X\}, \{J \mapsto R\}, \{T \mapsto R\}, \{J \mapsto X, T \mapsto X\}, \{J \mapsto R, T \mapsto R\}, \{J \mapsto X, T \mapsto R\}, \{J \mapsto R, T \mapsto X\}\}$, onde $\epsilon$ é a substituição identidade. Finalmente, o conjunto $C_4(q_0)$ é constituído por:

   - Para $\theta' = \epsilon$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(J,T) >$
   - Para $\theta' = \{J \mapsto X\}$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(X,T) >$
   - Para $\theta' = \{T \mapsto X\}$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(J,X) >$
   - Para $\theta' = \{J \mapsto R\}$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(R,T) >$
   - Para $\theta' = \{T \mapsto R\}$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(J,R) >$
   - Para $\theta' = \{J \mapsto X, T \mapsto X\}$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(X,X) >$
   - Para $\theta' = \{J \mapsto R, T \mapsto R\}$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(R,R) >$
   - Para $\theta' = \{J \mapsto X, T \mapsto R\}$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(X,R) >$
   - Para $\theta' = \{J \mapsto R, T \mapsto X\}$, adicionamos $< p(N,R)\ q(L)\ r(N,T)\ s(R,X) >$

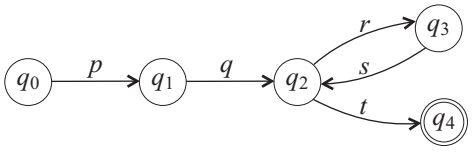



Figura 2.4: Automato $A_{\mathcal{R}}$.

2. **Poda:** Para cada seqüência $s$ de $C_k$ são calculadas todas as sub-seqüências maximais de $s$ que são legais com respeito a algum estado de $A_{\mathcal{R}}$. Se algumas dessas seqüências não unifica com nenhuma seqüência de $F = F_1 \cup \ldots \cup F_{k-1}$, $s$ é podada de $C_k$.

## 2.4 Padrões Temporais que Representam o Tempo por Intervalos

Padrões temporais que representam o tempo por intervalos são aqueles em que os eventos ocorrem em determinados períodos de tempo (durante o período entre duas horas, durante o período entre duas datas, etc.). Estes padrões são chamados de *padrões intervalares*.

Grande parte dos autores de trabalhos sobre mineração de padrões intervalares se baseiam em um formalismo lógico, chamado de Lógica Temporal de Intervalos (LTI), desenvolvido por Allen [2].

### 2.4.1 Lógica Temporal de Intervalos - LTI

Os aspectos dinâmicos do mundo, principalmente os que dizem respeito a estados, fatos ou acontecimentos, têm sido motivo de grande interesse nas mais variadas disciplinas. Em 1983, Allen [38] desenvolveu a Lógica Temporal de Intervalos (LTI), posteriormente aprimorada em [2], que é significantemente mais expressiva e mais natural para representar o tempo que outras aproximações, como por exemplo em Inteligência Artificial. Esta lógica considera períodos de tempo, relacionamentos entre estados e seus efeitos e é definida como uma LTPO 2-ordenada com *tempo* e *dados* ordenados, onde o tempo é representado por *períodos* ou *intervalos*, diferente da Lógica Temporal Linear (LTL) [39], onde o tempo é representado por *instantes* ou *pontos*.

Na LTI, Allen considera um período de tempo como sendo o tempo associado a algum evento e propõe representações e considerações para descrever a estrutura temporal básica usada na lógica. Para Allen, períodos de tempo são *intervalos* e os limites que definem o início e o fim dos intervalos são considerados *pontos*, por isso, ele não faz relacionamentos entre pontos (não períodos) e intervalos (períodos).

Para relacionar dois intervalos, Allen definiu 13 relacionamentos possíveis, através de predicados temporais (figura 2.5). Por exemplo, $before(i, j)$ significa que o intervalo $i$ ocorre antes do intervalo $j$ e $after(j, i)$ é o relacionamento inverso, $meets(i, j)$ significa que o intervalo $i$ termina no mesmo instante em que o intervalo $j$ começa e $metby(j, i)$ é o relacionamento inverso, e assim por diante.

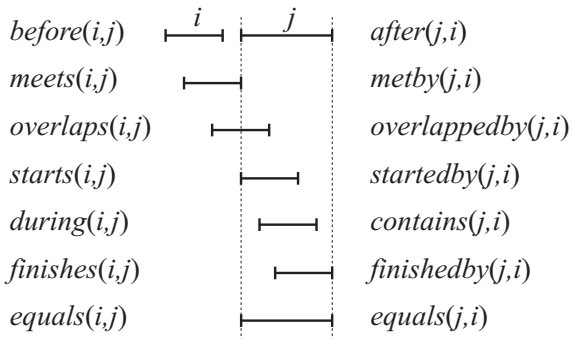



Figura 2.5: Possíveis relacionamentos entre dois intervalos (*adaptação de [2]*)

Assim como os padrões seqüenciais, os padrões intervalares também podem ser proposicionais ou de primeira ordem. Nas próximas seções serão apresentados estes problemas.

### 2.4.2 O Problema de Mineração de Padrões Intervalares Proposicionais

A mineração de padrões intervalares proposicionais pode ser aplicada com sucesso em diversas áreas como medicina, agropecuária, agronomia, etc. Os padrões minerados por este tipo de problema têm como objetivo representar relacionamentos temporais existentes entre eventos que ocorreram durante algum período de tempo para que futuros comportamentos possam ser preditos com base em acontecimentos passados.

Em 2001, Frank Höppner [40] usou a LTI de Allen pela primeira vez para tratar o problema de descoberta de padrões intervalares sobre bancos de dados temporais. Nesta seção serão descritas sucintamente as principais idéias propostas por Höppner. Caso o leitor tenha interesse em maiores informações consulte [40].

**Definição 2.4.1 (Banco de dados de seqüências de estado [40])**. Seja $BD$ um banco de dados de seqüências de estados. Um **estado** $s \in BD$ acontece durante um período de tempo $[a, b]$, onde, $a$ representa o inicio e $b$ representa o fim do estado $s$. Uma **seqüência de estados** em $BD$ é uma serie de triplas representando intervalos de estados da forma $(a_1, s_1, b_1), (a_2, s_2, b_2), \ldots, (a_n, s_n, b_n)$, onde, $a_i \leq a_{i+1}$ e $a_i < b_i$.

Höppner não exige que um estado termine antes do início de outro, entretanto, todo estado deve ser maximal.

**Definição 2.4.2 (Estado maximal [40])**. Um estado $(a_i, s, b_i)$ é **maximal** se não existe nenhum $(a_j, s, b_j)$ na seqüência, tal que, $[a_i, b_i]$ e $[a_j, b_j]$ sobreponha (*overlaps*) ou encontre (*meets*) um ao outro. Se o estado não for maximal, então ele é unido com o outro em um único intervalo, ou seja, $(min(a_i, a_j), s, max(b_i, b_j))$, onde, $min(a_i, a_j)$ retorna o menor valor, $a_i$ ou $a_j$, e $max(b_i, b_j)$ retorna o maior valor, $b_i$ ou $b_j$.

**Exemplo 2.4.1** Considere a seqüência $s = (1, fluoxetina, 3), (2, paracetamol, 6), (3, fluoxetina, 6)$. O estado $fluoxetina$ não é máximal. Portanto, os estados $fluoxetina$ devem ser unidos, resultando em $s = (1, fluoxetina, 6), (2, paracetamol, 6)$.

**Definição 2.4.3 (Relacionamentos entre os estados [40])**. Dados $n$ estados, a posição de cada intervalo com relação ao outro é representada por uma matriz $n \times n$, representada por $R$, cujos elementos $R[i, j]$ descrevem os **relacionamentos entre os estados** $s_i$ e $s_j$.

**Exemplo 2.4.2** Considere a seqüência de estados da figura 2.6(a). O estado $A$ é sempre seguido por $B$, e o espaço entre $A$ e $B$ é coberto por $C$. A matriz que relaciona os intervalos é mostrada na figura 2.6(b), onde, $b = before$, $a = after$, $o = overlaps$ e $ob = overlappedby$.

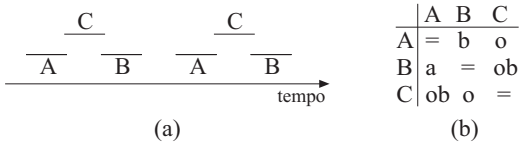



Figura 2.6: (a) seqüência de estados, (b) relacionamentos temporais

**Definição 2.4.4 (Padrão [40])**. Um **padrão** é um par $(s, R)$, onde, $s$ é uma seqüência de estados da forma $(a_1, s_1, b_1), \ldots, (a_n, s_n, b_n)$ e $R$ representa o relacionamento entre $[a_i, b_i]$ e $[a_j, b_j]$. O tamanho de um padrão $P$, denotado por $dim(P)$, é o número de intervalos de $P$. Se $dim(P) = k$, então, $P$ é um $k$-padrão.

**Definição 2.4.5 (Sub-padrão [40])**. Um padrão $(s_A, R_A)$ é um **sub-padrão** de $(s_B, R_B)$, denotado por $(s_A, R_A) \sqsubseteq (s_B, R_B)$, se $dim(s_A, R_A) \leq dim(s_B, R_B)$ e existe um mapeamento $\pi$, tal que, $\forall i, j \in \{1, ..., dim(s_A, R_A)\} : s_A(i) = s_B(\pi(i)) \wedge R_A[i, j] = R_B[\pi(i), \pi(j)]$.

Höppner considera somente os padrões que podem ser observados por um usuário dentro de uma janela de deslocamento de largura $t_{max}$. O **suporte** de um padrão $P$ definido por $sup(P)$ é o tempo total em que $P$ pode ser observado dentro desta janela. Dado um suporte mínimo $\alpha$, $P$ é freqüente em um banco de dados de seqüências de estados $BD$ se $sup(P) \geq \alpha$. Este problema de mineração é dado em linhas gerais por:

- **Dado:** Um banco de dados de seqüências de estados $BD$ e um suporte mínimo $\alpha$.
- **Encontrar:** Todos os padrões freqüentes em relação a $BD$ e $\alpha$.

**Algoritmo de Mineração Padrões Temporais Intervalares Proposicionais.** Para encontrar os 1-padrões freqüentes ($F_1$) é calculado o suporte de todos os 1-padrões do banco de dados. Na iteração $k$ são gerados padrões candidatos ($C_k$) a partir dos padrões freqüentes de $F_{k-1}$. A seguir são removidos de $C_k$ todos os padrões candidatos que não satisfazem a propriedade *Apriori*. Os padrões restantes são avaliados e os que possuírem suporte maior ou igual ao suporte mínimo estipulado pelo usuário são inseridos em $F_k$. Este procedimento é repetido até que mais nenhum padrão freqüente possa ser encontrado.

1. **Geração**: Para gerar um $k$-padrão, são unidos dois ($k$-1)-padrões freqüentes que possuem um prefixo de tamanho $k$-2 em comum. Considere dois ($k$-1)-padrões $S$ e $T$, que possuem as seqüências de estados $0, ..., k\text{-}2, s$ e $0, ..., k\text{-}2, t$, respectivamente. Um $k$-padrão $P = (s_P, R_P)$ é gerado como mostra a figura 2.7, onde $R_P[k\text{-}1, k] = r$ representa o relacionamento entre $s$ e $t$. Note que a sub-matriz $A$ é igual em $R_S, R_T$ e $R_P$. Em $R_P$, os relacionamentos entre os primeiros $k$-2 estados são os mesmos de $R_S$ e $R_T$, e $r$ pode ser um dos 13 relacionamentos definidos por Allen para relacionar $s$ e $t$, $ir$ é o inverso de $r$.

| $R_S$ | $sS$ 0 ... k-2 | $s$ |
|---|---|---|
| 0 ... k-2 | $A$ | $B$ |
| $s$ | $C$ | = |

(a) Padrão $S$

| $R_T$ | $sT$ 0 ... k-2 | $t$ |
|---|---|---|
| 0 ... k-2 | $A$ | $D$ |
| $t$ | $E$ | = |

(b) Padrão $T$

| $R_P$ | $sP$ 0 ... k-2 | s | $t$ |
|---|---|---|---|
| $sP$ 0 ... k-2 | $A$ | $B$ | $D$ |
| $s$ | $C$ | = | $r$ |
| $t$ | $E$ | $ir$ | = |

(c) Padrão $P$

Figura 2.7: Gerando um $k$-padrão $P$ a partir de dois ($k$-1)-padrões $S$ e $T$

2. **Poda**: Além de usar a propriedade *Apriori* para podar os padrões que possuem sub-padrões não freqüentes, Höppner usa outra técnica de poda baseada na transitividade dos relacionamentos temporais. Maiores detalhes sobre essa técnica de poda podem ser encontrados em [40].

3. **Avaliação**: Para calcular o suporte dos padrões candidatos, a seqüência estados é percorrida pela janela de deslocamento. Cada estado da seqüência do padrão possui um contador que é incrementado sempre que o estado é visível dentro da janela. No final, o suporte dos padrões é a soma dos contadores. Detalhes sobre o cálculo do suporte podem ser encontrados em [40].

### 2.4.3 O Problema de Mineração de Padrões Intervalares de Primeira Ordem

Nesta seção será abordado o problema de mineração de padrões intervalares de primeira ordem. Esses padrões possuem grande expressividade e podem ser aplicados com sucesso em diversas áreas como medicina (pode-se descobrir se o uso de determinados medicamentos durante certos períodos de tempo influenciam na evolução de sintomas em pacientes), agropecuária (pode-se descobrir se determinados períodos de vacinação influenciam ou não no desenvolvimento de rebanhos), agricultura (pode-se descobrir a relação entre o uso de fertilizantes e agrotóxicos para um bom desenvolvimento de plantações), etc.

O problema de mineração de padrões intervalares de primeira ordem foi inicialmente estudado por Lattner e Herzog em [9] e [41], onde criaram formalismos e desenvolveram algoritmos de possibilitam a mineração desses padrões. A seguir serão apresentados os principais conceitos definidos em [9] para formalizar este problema de mineração.

Considere $R_{esq} = \{ep_1, ep_2, \ldots\}$ como sendo um conjunto de esquemas de predicados, onde, $ep_i = < l_i, a_i >$, e $l_i$ é um predicado com aridade $a_i$. Considere também o conjunto $\mathcal{P}_{temp}$ dos predicados temporais definidos por Allen e um banco de dados $BD = \{s_1, s_2, \ldots\}$, onde cada $s_i$ é uma seqüência, defina a seguir.

**Definição 2.4.6 (Seqüência [9])**. Uma **seqüência** $s_i$ é uma tripla $(\mathcal{A}_i, \mathcal{TA}_i, \mathcal{C}_i)$, onde $\mathcal{C}_i$ é um conjunto de constantes, $\mathcal{A}_i$ é um conjunto de átomos $\{p_1(c_{1_1}, ..., c_{1_n}), p_2(c_{2_1}, ..., c_{2_n}), ...\}$, onde $p_i$ é uma instância de $ep_i \in R_{esq}$ com aridade $n$ e $c_{i_1}, ..., c_{i_n} \in \mathcal{C}_i$. $\mathcal{TA}_i$ é um conjunto de átomos temporais $\{r_1(p_a, op, p_b), r_2(p_b, op, p_c), ...\}$, onde, $p_a, p_b, p_c \in \mathcal{A}_i$ e $op \in \mathcal{P}_{temp}$.

**Exemplo 2.4.3** Considere um esquema $R_{esq} = \{<medicamento, 1>, <sintoma, 1>\}$. Uma seqüência $s$ pode ser definida por $(\{medicamento(fluoxetina), sintoma(náuseas)\}, \{r(medicamento, before, sintoma)\}, \{fluoxetina, náuseas\})$ que intuitivamente significa que o uso de $fluoxetina$ leva ao aparecimento de $náuseas$ no futuro.

**Definição 2.4.7 (Padrão [9])**. Seja um conjunto $\mathcal{V}$ de variáveis e um conjunto $\mathcal{P} = \{q_1, q_2, \ldots\}$ de padrões. Um **padrão** $q_i$ é uma tripla $(\mathcal{R}_i, \mathcal{TR}_i, \mathcal{V}_i)$, onde $\mathcal{V}_i$ é um conjunto de variáveis do padrão, $\mathcal{R}_i$ é um conjunto de átomos $\{p_1(v_{1_1}, \ldots, v_{1_n}), p_2(v_{2_1}, \ldots, v_{2_n}), \ldots\}$ tal que $v_{i_1}, \ldots, v_{i_n} \in \mathcal{V}_i$, e $\mathcal{TR}_i$ é um conjunto de átomos temporais tal como apresentado na definição anterior. Um padrão com $k$ átomos é um padrão de tamanho $k$ ou um $k$-padrão.

**Exemplo 2.4.4** Considere o esquema $R_{esq}$ do exemplo anterior e um conjunto $\mathcal{V} = \{X, Y, Z\}$ de variáveis. Um padrão $q$ pode ser dado por $(\{medicamento(X), sintoma(Y)\}, \{r(medicamento, before, sintoma)\}, \{X, Y\})$ que significa intuitivamente que o uso de um medicamento $X$ leva ao surgimento de um sintoma $Y$ no futuro.

Uma seqüência $s$ **satisfaz** um padrão $q$ se existe um mapeamento das constantes de $s$ em todas as variáveis de $q$, tal que todos os predicados de $q$ façam parte dos termos mapeados e todas as restrições temporais de $q$ sejam satisfeitas por $s$.

**Exemplo 2.4.5** Seja uma seqüência $s$=$(\{medicamento(fluoxetina), sintoma(nauzeas)\}, \{tr_1(medicamento, before, sintoma)\}, \{fluoxetina, nauzeas\})$ e um padrão $q = (\{medicamento(X), sintoma(nauzeas)\}, \{tr_1(medicamento, before, sintoma)\}, \{X\})$. A seqüência $s$ satisfaz $q$ com o mapeamento $fluoxetina \rightarrow X$. Entretanto, o padrão $q_1 = (\{medicamento(X), sintoma(nauzeas)\}, \{tr_1(medicamento, meets, sintoma)\}, \{X\})$ não é satisfeito por $s$, pois a restrição temporal de $q_1$ não é satisfeita por $s$.

Lattner e Herzog definiram uma avaliação um pouco diferente da apresentada na maioria dos trabalhos sobre mineração de padrões temporais baseados na técnica *Apriori*. Cada padrão é avaliado em 5 funções diferentes: tamanho relativo, freqüência relativa, coerência, restritividade temporal e preferência dos predicados. O **suporte** de um padrão $q$ é obtido multiplicando os valores de cada uma das 5 funções por pesos diferentes, os resultados são somados é divididos pela soma dos pesos. Detalhes destas funções podem ser encontrados em [9]. Um padrão $q$ é considerado freqüente se seu suporte for maior que um suporte mínimo $\alpha$. A formulação geral deste problema de mineração é dada por:

- **Dado:** Um banco de dados $BD$ e um suporte mínimo $\alpha$.
- **Encontrar:** Todos os padrões freqüentes em relação a $BD$ e $\alpha$.

Para minerar o padrão definido anteriormente, Lattner e Herzog criaram alguns algoritmos apresentados em [9] e otimizados em [41]. Embora mais complexos, são mais detalhados os algoritmos apresentados em [9], por isso eles são os únicos a serem descritos.

**Algoritmo de Mineração de Padrões Intervalares de Primeira Ordem.** O algoritmo principal desenvolvido por Lattner e Herzog também é baseado na técnica *Apriori*, portanto trabalha em iterações. Na primeira iteração é gerado um conjunto $C_1$ constituído por 1-padrões candidatos. Em uma iteração $k$ são gerados $k$-padrões ($C_k$) através de algoritmos específicos e em seguida todos são avaliados. Os padrões que não possuem suporte maior ou igual a um suporte mínimo são podados de $C_k$ e os que restaram são inseridos em um conjunto $F_k$ de padrões freqüentes. Na iteração $k+1$ são gerados ($k$+1)-padrões a partir dos padrões freqüentes de $F_k$.

1. **Geração**: Lattner e Herzog desenvolveram quatro algoritmos para a geração de novos padrões. Um primeiro algoritmo especializa um padrão inserindo um novo átomo com variáveis que ainda não foram usadas no padrão. Um segundo algoritmo especializa um padrão inserindo uma nova relação temporal para qualquer par de predicados do padrão que ainda não foram relacionados. Um terceiro algoritmo especializa um padrão unificando um par de variáveis. Um quarto e último algoritmo

substitui um predicado mais geral por um predicado mais específico. Detalhes sobre esses quatro algoritmos podem ser encontrados em [9].

2. **Poda**: A única poda utilizada por Lattner e Herzog diz respeito a um suporte mínimo. Portanto o algoritmo não utiliza a propriedade *Apriori*, o que faz com que seja desperdiçado processamento em padrões que certamente não serão freqüentes.

3. **Avaliação**: Em cada iteração, após a etapa da geração, o banco de dados é percorrido e para cada padrão são obtidos cinco valores que no fim da avaliação resultam em uma porcentagem que é o suporte total do padrão.

### 2.4.4 O Problema de Mineração de Padrões Intervalares de Primeira Ordem com Restrições de Expressões Regulares

Esse problema ainda foi pouco explorado no meio acadêmico. Um dos principais trabalhos relacionado a esta abordagem foi publicado em 2005 por Sandra de Amo et al. [16]. O trabalho de Sandra de Amo é parecido com o de Lattner e Herzog [9], sendo que a diferença fundamental entre eles é que em [9] a única forma de restrição dos padrões é um suporte mínimo estipulado pelo usuário, e em [16], além do suporte mínimo foram introduzidas restrições de expressões regulares similares às utilizadas nos algoritmos da família SPIRIT.

Para facilitar o entendimento serão apresentados agora alguns dos principais conceitos definidos em [16]. Considere três conjuntos disjuntos de símbolos $\mathcal{P}$ (predicados), $\mathcal{V}$ (variáveis de dados) e $\mathcal{C}$ (constantes de dados). Considere também um conjunto de predicados temporais $\mathcal{P}_T = \{before, meets, overlaps, starts, during, finishes\}$ (predicados definidos por Allen [2]), um conjunto $\mathcal{V}_t$ (variáveis temporais) e um esquema de banco de dados $\mathbf{R} = \{K, R_1, \ldots, R_n\}$, onde cada esquema relacional $R_i$ tem aridade $k_i \geq 2$ e $K$ tem aridade $m \geq 1$. A partir deste ponto, variáveis serão representadas por letras minúsculas e constantes simplesmente pelos seus valores.

**Definição 2.4.8 (Padrão).** Um **padrão** é uma tripla $(K, \mathcal{D}, \mathcal{T})$, onde $K$ é um átomo de dados especial da forma $K(x_1, \ldots, x_n)$, chamado de *referência*, com $x_i \in \mathcal{V}$, $\mathcal{D} = \{p_1(s_1, \ldots, s_n, e_1), \ldots, p_k(s_1, \ldots, s_n, e_k)\}$ é um conjunto de átomos de dados, onde $p_i \in \mathcal{P}$,

$s_1, \ldots, s_n \in \mathcal{V} \cup \mathcal{C}$ e $e_i \in \mathcal{V}_t$. $\mathcal{T}$ é um conjunto de átomos temporais da forma $r(e, f)$, onde $r \in \mathcal{P}_T$ e $e, f \in \mathcal{V}_t$ e são avaliadas em intervalos $[a, b]$, onde $a$ representa o início e $b$ representa o fim do intervalo.

**Exemplo 2.4.6** Considere um padrão $P = (K, \mathcal{D}, \mathcal{T})$, onde, $K = Paciente(x)$, $\mathcal{D} = \{Medicamento(x, y, e), Sintoma(x, z, f)\}$ e $\mathcal{T} = \{before(e, f)\}$. Intuitivamente esse padrão nos diz que um paciente $x$ que tomou um medicamento $y$ durante um período $e$ apresentou um sintoma $z$ em um período futuro $f$. Note que o interesse é sempre em analisar o comportamento dos pacientes registrados na tabela *Paciente*, e por isso $Paciente(x)$ é chamado de referência.

Neste problema de mineração as expressões regulares $\mathcal{R}$ são definidas sobre um conjunto de modos e devem ser satisfeitas pelos padrões.

**Definição 2.4.9 (Modo).** Um **modo** sobre $\mathbf{R}$ é uma expressão da forma $R(u_1, ..., u_s, \#)$, onde cada $u_i \in \mathcal{V}$ ou é um símbolo \$. O símbolo # é um novo símbolo e $R$ é um dos predicados $R_i \in \mathbf{R}$. Um átomo de dados $A$ *é de acordo* com um modo $R(u_1, \ldots, u_s, \#)$ se $A = R(y_1, \ldots, y_s, e)$ e para cada $i = 1, \ldots, s$ tem-se: (1) Se $u_i \in \mathcal{V}$, então $y_i = u_i$; (2) Se $u_i = *$, então $y_i \in \mathcal{V} \cup \mathcal{C}$; (3) $e \in \mathcal{V}_t$.

**Exemplo 2.4.7** Considere o modo $Medicamento(x, \$, \#)$ e os átomos de dados $Medicamento(x, fluoxetina, e)$ e $Medicamento(y, fluoxetina, e)$, o primeiro átomo é de acordo com modo, já, o segundo não é, pois, a condição (1) não é satisfeita.

Seja $\Sigma$ um conjunto de modos sobre $\mathbf{R}$. Em [16] são consideradas linguagens regulares, (conjunto de *string*) sobre o alfabeto $\Sigma$, especificadas por expressões regulares.

Um padrão temporal $P = (K, \mathcal{D}, \mathcal{T})$, onde, $\mathcal{D}$ é visto como uma seqüência de átomos da forma $< p_1(t^1_1, \ldots, t^1_{l_1}, e_1), \ldots, p_k(t^1_1, \ldots, t^1_{l_1}, e_k >$, com $e_1 < \ldots < e_k$, satisfazendo uma expressão regular $\mathcal{R}$ se: (1) Existe uma *string* $w_1, w_2, \ldots, w_n \in \mathcal{R}$, onde $w_i = p_i(u_i, \ldots, u_{l_i}, \#)$ tal que, $p_i(t^1_1, \ldots, t^1_{l_1}, e_i)$ é de acordo com $w_i$; (2) Para todo átomo de dados $p_i(t^1_1, \ldots, t^1_{l_1}, e_i) \in \mathcal{D}$, se $u_j = \$$ e $x_j$ é uma variável, então $s_j$ tem somente uma ocorrência em $\mathcal{D}$.

**Exemplo 2.4.8** Seja a expressão regular $\mathcal{R} = Medicamento(x, \$, \#)^* Sintoma(x, \$, \#)$. Os padrões $P_1 = (Paciente(x), \{Medicamento(x, fluoxetina, e), Sintoma(x, febre, f)\}, \{before(e, f)\}$ e $P_2 = (Paciente(x), \{Medicamento(x, y_1, e), Medicamento(x, y_2, f)\}, \{before(e, f)\}$ satisfazem $\mathcal{R}$. Já, o padrão $P_3 = (Paciente(x), \{Medicamento(x, y, e), Sintoma(x, y, f)\}), \{before(e, f)\}$ não satisfaz $\mathcal{R}$, pois a condição 2 não é satisfeita.

Pelos mesmos motivos apresentados nos algoritmos da família SPIRIT [12] descritos na seção 2.3.2, em [16] também é considerado um relaxamento $\mathcal{R}'$ *mais fraco* que $\mathcal{R}$. A diferença é que em [16] o relaxamento $\mathcal{R}'$ representa um prefixo associado a $\mathcal{R}$. Com isso, são gerados somente padrões prefixos válidos. A formulação geral desse tipo de problema de mineração é dada por:

- **Dado:** Um banco de dados $BD$, um suporte mínimo $\alpha$ e uma restrição $\mathcal{R}$.
- **Encontrar:** Todos os padrões freqüentes em relação a $BD$, $\alpha$ e que satisfazem $\mathcal{R}$.

Um protótipo chamado MILPRIT foi desenvolvido em [16] com o objetivo de minerar padrões temporais definidos para este problema de mineração. Este protótipo foi uma implementação preliminar muito ineficiente e possui algumas falhas. Uma delas é o fato de não realizar gerenciamento de memória. Com isso, é possível executá-lo somente sobre bancos de dados muito pequenos, uma vez que quando executado sobre bancos de dados grandes a memória principal é preenchida logo nos primeiros níveis (iterações) pelos padrões gerados fazendo com que a execução seja abortada. Outra falha ocorre na etapa de geração dos padrões candidatos, uma vez que determinados padrões não são gerados ou são gerados padrões que não são interessantes para o usuário.

**MILPRIT.** O MILPRIT foi desenvolvido baseado na técnica *Apriori* e contêm idéias básicas para a mineração de padrões temporais intervalares. Diferentemente do algoritmo apresentado em [9], na etapa da poda o MILPRIT utiliza a propriedade *Apriori*, garantindo que somente padrões potencialmente freqüentes sejam levados para a etapa da avaliação, onde é empregado o cálculo de suporte tradicional. É verificado por quantas tuplas do banco de dados um padrão é satisfeito e este valor é dividido pela quantidade total de tuplas da tabela de referência. Na etapa de geração são gerados somente padrões

válidos com respeito a $\mathcal{R}'$ (prefixo associado a $\mathcal{R}$). Em linhas gerais, na iteração $k$ é gerado um conjunto de padrões candidatos $C_k$ a partir dos padrões freqüentes e válidos contidos em $F_{k-1}$, obtido na iteração anterior. São podados de $C_k$ todos os padrões que possuem sub-padrões válidos e que não estejam em $F_{k-1}$, ou seja, não são freqüentes. O banco de dados é percorrido para calcular o suporte de todos os padrões de $C_k$, os padrões que possuem suporte maior ou igual a um suporte mínimo estipulado pelo usuário são considerados freqüentes e são inseridos em $F_k$. A seguir, descrevemos a etapa de geração.

1. **Geração:** Na fase da geração da iteração $k$ são gerados padrões mais específicos a partir dos padrões contidos em $F_{k-1}$. Para especializar os padrões são utilizados dois operadores de especialização, o **operador de extensão** e o **operador de instanciação** que possuem as mesmas propriedades (*optimais* e *completos*) dos operadores definidos em [13] e apresentados na seção 2.3.3. Somente um deles é aplicado de cada vez em um padrão de uma determinada iteração, garantindo que ele seja especializado iteração por iteração aumentando em um seu tamanho.

   Cada padrão $P = (K, \mathcal{D}, \mathcal{T})$ tem associado a ele um *vetor de refinamento*, definido por $v(P) = (n, c)$, onde $n$ é o tamanho (número de átomos de dados) de $\mathcal{D}$ e $c$ é o número de constantes qua aparecem em $\mathcal{D}$. O *nível de refinamento* de $P$, denotado por $l(P)$ é definido como $n + c$. Por exemplo, o padrão $P = (Paciente(x), \{Medicamento(x, fluoxetina, e), Sintoma(x, y, f)\}$ tem $v(P) = (2, 1)$, portanto, $l(P) = 3$. Com isso, $P$ pertence a iteração 3.

   Para garantir as duas propriedade do primeiro parágrafo, os dois operadores de especialização devem obedecer algumas condições. O operador de extensão só pode ser executado sobre um padrão $P$ se $v(P) = (n, 0)$, ou seja, se $P$ não tem nenhuma constante. O operador de extensão produz um conjunto de padrões $P'$ a partir de $P$ inserindo um novo átomo de dados no fim de $\mathcal{D}$ e uma nova variável temporal em $\mathcal{T}$. O operador de instanciação executado sobre $P$ produz um conjunto de padrões $P'$ a partir de $P$ substituindo algumas variáveis de $\mathcal{D}$ por uma constante $c$ de modo que não exista nenhuma constante após a variável instanciada em $\mathcal{D}$, no mesmo átomo ou nos átomos que o seguem.

**Exemplo 2.4.9** Considere a expressão regular $\mathcal{R} = Medicamento(x, \$, \#)^*$ $Sintoma(x, \$, \#)$ e um padrão $P_1 = (Paciente(x), \{Medicamento(x, y, e_1)\}, \mathcal{T}_1)$. Esse padrão pode ser especializado por extensão, resultando em um padrão $P_2 = (Paciente(x), \{Medicamento(x, y, e_1), Sintoma(x, z, e_2)\}, \mathcal{T}_2)$, e por instanciação, resultando no padrão $P_3 = (Paciente(x), \{Medicamento(x, fluoxetina, e_1)\}, \mathcal{T}_1)$. O padrão $P_2$ pode ser especializado por extensão e por instanciação resultando em $P_4 = (Paciente(x), \{Medicamento(x, y_1, e_1), Medicamento(x, y_2, e_2), Sintoma(x, z, e_3)\}, \mathcal{T}_3)$ e $P_5 = (Paciente(x), \{Medicamento(x, fluoxetina, e_1), Sintoma(x, z, e_2)\}, \mathcal{T}_2)$, respectivamente. O padrão $P_3$ não pode ser especializado por extensão, pois ele já tem uma constante e poderia gerar um padrão igual ao $P_5$.

## 2.5 Considerações Finais do Capítulo

Neste capitulo foram abordadas as principais tarefas existentes na abrangente área de mineração de dados e foram detalhadas os principais conceitos e algoritmos desenvolvidos com o objetivo de possibilitar a mineração de padrões sobre bancos de dados temporais.

Como descrito, na literatura existem vários padrões temporais que são especificados por formalismos da Lógica Temporal Proposicional e recentemente muitos estudos têm sido focados em descobrir conhecimentos expressados pela Lógica Temporal de Primeira Ordem. Por exemplo, pesquisas em Programação Lógica Indutiva (PLI) [42], que propõem formalismos mais expressivos que aqueles usados até então para representar padrões temporais. Tais formalismos são expressivos o suficiente para especificar muitos padrões, como por exemplo para modelar *logs* de usuários do sistema Unix [17].

É preciso ter em mente que novos problemas surgem a todo momento. Conseqüentemente, novos padrões devem ser modelados e novos formalismos e métodos elaborados para tornar possível sua mineração.

No próximo capítulo será formalizado o problema de mineração do padrão temporal híbrido que esta sendo proposto nesta dissertação. Serão apresentados os principais tópicos necessários para a compreensão da solução proposta da mineração de padrões temporais híbridos em bancos de dados temporais.

# Capítulo 3

# O Problema de Mineração de Padrões Temporais Híbridos

O problema de mineração de *padrões temporais híbridos* ou *pth's*, que são padrões temporais de primeira ordem que representam o tempo explicitamente em termos de pontos e/ou intervalos, será amplamente abordado neste capítulo.

## 3.1 Formalização do Problema

Uma das propostas deste trabalho é a criação de um novo padrão temporal, chamado padrão temporal híbrido ou *pth*, especificado por formalismos da Lógica Temporal de Intervalos. Nesta seção primeiramente serão introduzidos os conceitos básicos desta lógica e as adaptações realizadas para especificar o *pth*. Depois será especificado o banco de dados onde o *pth* pode ser minerado e em seguida definido formalmente o *pth* juntamente com o cálculo do *suporte*. Finalizando a seção, será apresentada a tarefa de mineração básica deste problema de mineração.

### 3.1.1 Adaptação da Lógica Temporal de Intervalos

Como dito na seção 2.4.1, Allen considera que períodos de tempo são *intervalos* e os limites que definem o início e o fim dos intervalos são *pontos*, por este motivo ele não definiu relacionamentos entre pontos (não períodos) e intervalos (períodos) na LTI.

Como uma das propostas deste trabalho é especificar o *pth* por formalismos da LTI de Allen, ela teve que ser adaptada para tratar também casos onde o tempo é represen-

tado por *instantes* ou *pontos* e não somente por *períodos* ou *intervalos*. A adaptação da LTI, realizada neste trabalho, considera um *ponto* como sendo um período de tempo sem duração, ou seja, um período de tempo que tem duração nula. Assim, como *pontos* e *intervalos* são considerados períodos de tempo, a LTI adaptada considera relacionamentos entre eles.

Antes de definir formalmente a LTI adaptada para este trabalho, serão detalhadas as adaptações realizadas na estrutura temporal da lógica, que é de fundamental importância para o entendimento da mesma.

**Estrutura Temporal.** Considere uma estrutura temporal $\mathcal{T}_{temp}$ constituída pelos conjuntos disjuntos descritos a seguir:

- $\mathcal{V}_{temp}$: conjunto de variáveis do tipo *tempo*, chamadas de *variáveis temporais*.
- $\mathcal{C}_{temp}$: conjunto de constantes do tipo *tempo*, chamadas de *constantes temporais*.
- $\mathcal{P}_{temp}$: conjunto de predicados temporais (binários).

Os *termos temporais* são *variáveis* ou *constantes* temporais e são relacionados pelos predicados temporais $\mathcal{P}_{temp} = \{before, meets, overlaps, starts, during, finishes\}$. Esses predicados foram definidos por Allen na LTI apresentada na seção 2.4.1. O predicado *equals*, que também foi definido por Allen, não foi considerado na adaptação da LTI. Assim, são considerados somente *pth's* representados por conjuntos de *eventos*. O predicado temporal *equals* implica que um mesmo intervalo pode ser associado a eventos diferentes, e portanto, se ele fosse considerado, *pth's* representados por conjuntos de *conjuntos de eventos* também deveriam ser tratados, o que tornaria mais complexa a definição do problema. Isto irá ficar claro após a proposição 3.1.1.

O conjunto $\mathcal{V}_{temp}$ é constituído por dois conjuntos: $\mathcal{V}_{it}$, conjunto de variáveis do tipo *tempo* chamadas de *variáveis temporais intervalares* e $\mathcal{V}_{pt}$, conjunto de variáveis do tipo *tempo* chamadas de *variáveis temporais pontuais*. No restante desta dissertação, para evitar confusão, as variáveis temporais de $\mathcal{V}_{it}$ serão denotadas por $(e, f, g, e_1, \ldots)$, e as variáveis temporais de $\mathcal{V}_{pt}$ serão denotadas por $(s, t, u, s_1, \ldots)$. Os símbolos $(\tau, \gamma, \tau_1, \ldots)$ serão utilizados para representar, sem distinção de tipo (intervalar ou pontual), as variáveis temporais de $\mathcal{V}_{temp} = \mathcal{V}_{it} \cup \mathcal{V}_{pt}$.

O conjunto $\mathcal{V}_{it}$ tem como domínio o conjunto $\mathcal{I} = \{[i,j] \mid i,j \in \mathbb{N} \text{ e } i < j\}$. Os elementos de $\mathcal{I}$ são intervalos $[i,j]$, onde, $i$ e $j$ são números naturais mapeados em datas com $i$ representando o inicio e $j$ representando o fim do intervalo. O conjunto $\mathcal{V}_{pt}$ tem como domínio o conjunto $\mathbb{N}$. Os elementos de $\mathcal{V}_{pt}$ são números naturais também mapeados em datas. É importante ter em mente que naturalmente um *ponto* pode ser visto como um *intervalo* $[i,j]$, onde, $i = j$, assim, na LTI adaptada, um ponto é um período de tempo com início igual ao fim, ou seja, um *intervalo nulo*.

Para representar os possíveis relacionamentos entre duas variáveis temporais, a LTI adaptada para este trabalho considera três casos:

- **Caso 1 - Possíveis relacionamentos entre dois intervalos**

  São considerados doze possíveis variações de relacionamentos intuitivos que podem ocorrer entre dois intervalos, sendo seis primitivos e seis inversos. O predicado *equals* é omitido.

  Considerando as variáveis temporais intervalares $e$ e $f$, onde $e$ representa o intervalo $[a,b]$ e $f$ representa o intervalo $[a',b']$, os seis predicados temporais primitivos são interpretados da seguinte maneira:

  1. $before(e,f) := \{([a,b],[a',b']) \mid a < b < a' < b'\}$
  2. $meets(e,f) := \{([a,b],[a',b']) \mid a < b = a' < b'\}$
  3. $overlaps(e,f) := \{([a,b],[a',b']) \mid a < b' < a < b'\}$
  4. $starts(e,f) := \{([a,b],[a',b']) \mid a' = a < b < b'\}$
  5. $during(e,f) := \{([a,b],[a',b']) \mid a' < a < b < b'\}$
  6. $finishes(e,f) := \{([a,b],[a',b']) \mid a' < a < b = b'\}$

  Os seis predicados temporais inversos são interpretados da seguinte maneira:

  1. $after(f,e) \equiv before(e,f)$
  2. $meetby(f,e) \equiv meets(e,f)$
  3. $overlappedby(f,e) \equiv overlaps(e,f)$
  4. $startedby(f,e) \equiv starts(e,f)$
  5. $contains(f,e) \equiv during(e,f)$
  6. $finishedby(f,e) \equiv finishes(e,f)$

  A figura 3.1 ilustra a semântica dos predicados primitivos definidos anteriormente.

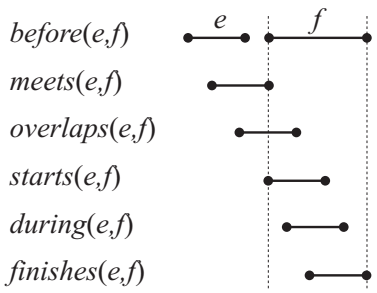



Figura 3.1: Possíveis relacionamentos entre dois intervalos

- **Caso 2 - Possíveis relacionamentos entre dois pontos**

  Somente um predicado primitivo e um inverso é considerado para relacionar dois pontos.

  Considerando as variáveis temporais pontuais $s$ e $t$, onde, $s$ representa o intervalo nulo $[a, b]$ e $t$ representa o intervalo nulo $[a', b']$, o único predicado temporal primitivo que relaciona dois pontos é interpretado da seguinte maneira:

  1. $before(s,t) := \{([a,b],[a',b']) \mid a = b < a' = b'\}$

  O único predicado temporal inverso é interpretado da seguinte maneira:

  1. $after(t,s) \equiv before(s,t)$

  Nos relacionamentos entre dois pontos os predicados *meets*, *starts* e $finishes$ têm as mesmas propriedades do *equals*, ou seja, $meets(s,t) \equiv starts(s,t) \equiv finishes(s,t) \equiv equal(s,t)$, por esta razão não são considerados. Para que o *overlaps* seja considerado as duas variáveis temporais envolvidas devem representar intervalos, uma vez que por *overlaps* subentende-se que um determinado acontecimento X inicou em algum momento anterior a um determinado acontecimento Y ter iniciado, continuou ocorrendo durante algum tempo e terminou após Y ter iniciado e antes de Y ter terminado. Como neste caso as duas variáveis envolvidas são pontos, o predicado *overlaps* não é considerado. Para que o *during* seja considerado, a última variável envolvida ($t$) deve representar um intervalo, uma vez que por *during* subentede-se que um determinado acontecimento X ocorreu durante um

determinado acontecimento Y. Portanto, Y obrigatóriamente deve ser um intervalo. Como neste caso a variável $t$ é um ponto, o predicado *during* não é considerado. A figura 3.2 ilustra a semântica do predicado primitivo definido anteriormente.

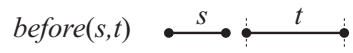



Figura 3.2: Único relacionamento possível entre dois pontos

- **Caso 3 - Possíveis relacionamentos entre ponto e intervalo e entre intervalo e ponto**

  Oito predicados, sendo quatro primitivos e quatro inversos, são considerados. Neste caso, é de fundamental importância a ordem de uma variável temporal em relação a outra, pois, isto restringe o uso de determinados predicados.

  Considerando a variável temporal pontual $s$ e a variável temporal intervalar $e$, onde $s$ representa o intervalo nulo $[a, b]$ e $e$ representa o intervalo $[a', b']$, os predicados temporais primitivos são interpretados da seguinte maneira:

  1. $before(s, e) := \{([a, b], [a', b']) \mid a = b < a' < b'\}$
  2. $starts(s, e) := \{([a, b], [a', b']) \mid a' = a = b < b'\}$
  3. $during(s, e) := \{([a, b], [a', b']) \mid a' < a = b < b'\}$
  4. $finishes(s, e) := \{([a, b], [a', b']) \mid a' < a = b = b'\}$
  5. $before(e, s) := \{([a', b'], [a, b]) \mid a' < b' < a = b\}$

  Os predicados temporais inversos são interpretados da seguinte maneira:

  1. $after(e, s) \equiv before(s, e)$
  2. $startedby(e, s) \equiv starts(s, e)$
  3. $contains(e, s) \equiv during(s, e)$
  4. $finishedby(e, s) \equiv finishes(s, e)$
  5. $after(s, e) \equiv before(e, s)$

Nos relacionamentos entre ponto e intervalo, respectivamente, o predicado *meets* não é considerado, pois $meets(s, e) \equiv starts(s, e)$ e para evitar redundância optou-se por descartar o *meets* e considerar o *starts*. Para que o predicado *overlaps* seja considerado, todas as variáveis temporais envolvidas devem ser intervalos, o que não acontece neste caso.

Nos relacionamentos entre intervalo e ponto, respectivamente, os predicados *meets*, *overlaps*, *starts*, *during* e $finishes$ não são considerados para manter uma ordem sobre as variáveis temporais. Esta ordem será explicada na proposição 3.1.2. Outros motivos pelos quais os predicados *meets*, *overlaps* e *during* não são considerados são: $meets(e, s) \equiv finishes(s, e)$ e o predicado $finishes$ já é considerado no relacionamento entre ponto e intervalo, respectivamente; o predicado *overlaps* não pe considerado pelo mesmo motivo do parágrafo anterior; e para que o predicado *during* seja considerado, a última variável temporal envolvida ($s$) deve ser um intervalo, o que não acontece neste caso.

A figura 3.3 ilustra a semântica dos predicados temporais definidos anteriormente.

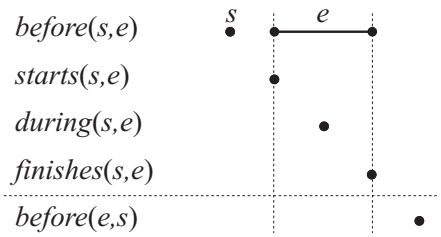



Figura 3.3: Possíveis relacionamentos entre um ponto e um intervalo (acima da linha pontilhada), e entre um intervalo e um ponto (abaixo da linha pontilhada)

Nesta dissertação são considerados somente os predicados temporais primitivos, uma vez que eles são suficientes para representar todos os possíveis relacionamentos entre variáveis temporais de qualquer tipo, como mostrado nas figuras 3.1, 3.2 e 3.3.

**Definição 3.1.1 (Lógica Temporal de Intervalos (LTI) Adaptada)**. Considere a linguagem temporal $\mathcal{L}_T$ constituída pelos conjuntos de símbolos $\mathcal{V}_{temp}$, $\mathcal{C}_{temp}$ e $\mathcal{P}_{temp}$ descritos acima, e pelos seguintes símbolos:

- Um conjunto de predicados $\mathcal{P}red$ com aridade $\geq 1$, cuja última entrada é do tipo *tempo* e as outras entradas são do tipo *dado* $(dado, dado, \ldots, tempo)$.

- Dois símbolos de igualdade $=_t$ e $=_d$.

- Um conjunto $\mathcal{V}ar$ de variáveis e $\mathcal{C}onst$ de constantes disjuntos dos conjuntos $\mathcal{V}_{temp}$, $\mathcal{C}_{temp}$, e $\mathcal{P}_{temp}$, ambos do tipo *dado*.

  Um *termo de dado* é uma *variável* ou uma *constante* do tipo *dado*.

A LTI adaptada também é um lógica de primeira ordem 2-ordenada sobre a linguagem $\mathcal{L}_T$ cujas fórmulas são definidas do seguinte modo:

1. Se $P \in \mathcal{P}red$ com aridade $n+1$, $s_1, \ldots, s_n$ são termos de dados e $\tau$ é um termo temporal, então, a expressão $P(s_1, \ldots, s_n, \tau)$ e $s_1 =_d s_2$ são fórmulas da LTI adaptada. Essas fórmulas são chamadas de *átomos de dados*.

2. Se $r \in \mathcal{P}_{temp}$ e $\tau, \gamma$ são termos temporais, então $r(\tau, \gamma)$ e $\tau =_t \gamma$ são fórmulas da LTI. Essas fórmulas são chamadas de *átomos temporais*.

3. Se $F$ e $G$ são fórmulas da LTI adaptada, então $F \wedge G, F \vee G$ e $\neg F$ são fórmulas da LTI adaptada.

4. Se $F$ é uma fórmula da LTI adaptada $e$ é uma variável temporal e $x$ uma variável de dado, então $\exists e F$ e $\exists x F$ são fórmulas da LTI adaptada.

Os átomos de dados e os átomos temporais são chamados de *fórmulas atômicas* ou simplesmente *átomos*.

**Exemplo 3.1.1** Considere $\mathcal{P}red = \{Hormônio, Birads\}$ um conjunto de predicados com aridade 3, definidos do seguinte modo:

- $Hormônio(x, y, e)$: a paciente $x$ tomou o hormônio $y$ durante o intervalo $e$.

- $Birads(x, z, s)$: a paciente $x$ apresentou birads $z$ no ponto $s$.

A seguinte expressão é uma fórmula da LTI adaptada:

$$(\forall x)((\forall e)Hormônio(x, progesterona, e) \longrightarrow (\exists s)Birads(x, 3, s) \wedge before(e, s))$$

Esta fórmula intuitivamente significa que para qualquer intervalo $e$, onde uma paciente $x$ faz o uso do hormônio *progesterona*, existe um ponto $s$ posterior, onde essa mesma paciente apresenta birads 3.

As seguintes fórmulas, derivadas das relações que interpretam os predicados temporais na estrutura temporal $\mathcal{T}_{temp}$, são válidas na LTI adaptada:

1. $\forall\tau\forall\gamma\ (\tau \neq \gamma\ \rightarrow\ (before(\tau,\gamma)\ \vee\ meets(\tau,\gamma)\ \vee\ overlaps(\tau,\gamma)\ \vee\ during(\tau,\gamma)\ \vee\ starts(\tau,\gamma)\ \vee\ finishes(\tau,\gamma)))$
2. $\forall\tau\forall\gamma\ \neg(r(\tau,\gamma)\ \wedge\ r'(\tau,\gamma)), \forall r,\ \forall r' \in \mathcal{P}_{temp}, r \neq r'$

A primeira fórmula expressa o fato de que os predicados temporais representam todas as possibilidades de relacionamentos entre intervalos e/ou pontos $\tau$ e $\gamma$, de acordo com as restrições descritas na estrutura temporal. A segunda fórmula expressa o fato de que existe uma e somente uma relação entre $\tau$ e $\gamma$, ou seja, os predicados temporais não podem ser satisfeitos simultaneamente.

### 3.1.2 O Banco de Dados

Nesta seção será apresentado o banco de dados temporal sobre o qual os *pth's* são minerados. Considere o esquema de banco de dados $\mathbf{R} = \{K, R_1, \ldots, R_n\}$, onde cada esquema relacional $R_i$ tem aridade $k_i \geq 2$ e tipo $(dado, \ldots, dado, tempo)$, e $K$ tem aridade $m$ e tipo $(dado,\ \ldots, dado)$. Um *conjunto de dados temporais* sobre $\mathbf{R}$ é um conjunto de relações temporais $\{k, r_1, \ldots, r_n\}$, onde $r_i$ é um conjunto de tuplas $(a_1, \ldots, a_{n_i}, \tau)$ sobre $R_i$, com $\tau \in \mathcal{I}$ ou $\tau \in \mathbb{N}$, $a_j \in \mathbf{dom}$ (domínio dos dados), e $n_i + 1$ é a aridade de $R_i$, $k$ é um conjunto de tuplas sobre $K$.

Uma relação temporal $r$ é *fechada* se, para toda tupla $(a_1, \ldots, a_k, \tau)$ e $(a_1, \ldots, a_k, \gamma) \in r$, tem-se $\tau \neq \gamma$, ou seja, $\tau \cap \gamma = \emptyset$. Esto significa que períodos de tempo, com ou sem duração, associados a um mesmo fato são maximais. Um banco de dados é *fechado* se suas relações são fechadas. Nesta dissertação são considerados somente bancos de dados fechados.

**Exemplo 3.1.2** Seja $\mathbf{R} = \{Paciente(NomePac),\ Birads(NomePac,\ Categoria,\ T_{pt}),\ Gravidez(NomePac,\ Status,\ T_{pt}),\ Fumo(NomePac,\ Qtde,\ T_{it}),\ Hormônio(NomePac,\ NomeHorm, T_{it})\}$ o esquema de um banco de dados temporal de uma clínica de radiologia de mama. Os atributos $T_{it}$ e $T_{pt}$ tem como domínio $\mathcal{I}$ e $\mathbb{N}$, respectivamente. O atributo $NomePac$ denota o nome das pacientes registradas na tabela $Paciente$. Os atributos $Categoria$, $Status$, $Qtde$ e $NomeHorm$ denotam informações do histórico clínico das pacientes. A figura 3.4 ilustra tal banco de dados e a linha do tempo que representa o mapeamento dos atributos temporais em datas.

*Paciente*

| *IdPac* | *NomePac* |
|---|---|
| 01 | Maria |
| 02 | Priscila |
| 03 | Carla |
| 04 | Paula |
| 05 | Cintia |

*Birads*

| *NomePac* | *Categoria* | $T_{pt}$ |
|---|---|---|
| Maria | 3 | 10 |
| Paula | 3 | 9 |
| Aline | 2 | 2 |
| Carla | 3 | 5 |

*Gravidez*

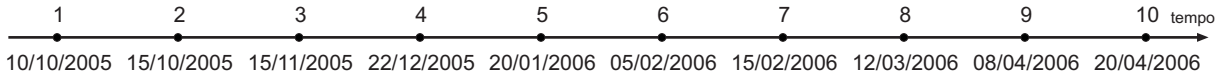
| *NomePac* | *Status* | $T_{pt}$ |
|---|---|---|
| Maria | sim | 3 |
| Carla | sim | 2 |
| Paula | sim | 1 |

*Fumo*

| *NomePac* | *Qtde* | $T_{it}$ |
|---|---|---|
| Maria | 5 | [4,6] |
| Priscila | 10 | [2,4] |
| Paula | 10 | [2,5] |
| Cintia | 20 | [3,6] |

*Hormônio*

| *NomePac* | *NomeHorm* | $T_{it}$ |
|---|---|---|
| Maria | estradiol | [5,8] |
| Paula | progesterona | [6,8] |
| Priscila | progesterona | [5,9] |
| Carla | noretisterona | [4,6] |




10/10/2005 15/10/2005 15/11/2005 22/12/2005 20/01/2006 05/02/2006 15/02/2006 12/03/2006 08/04/2006 20/04/2006

Figura 3.4: Exemplo de um banco de dados de pacientes e de um mapeamento em datas

Os domínios deste banco de dados são definidos do seguinte modo:

- $\mathbf{dom}(1, Paciente) = \mathbf{dom}(1, Birads) = \mathbf{dom}(1, Gravidez) = \mathbf{dom}(1, Fumo) = \mathbf{dom}(1, Hormônio) =$ {Maria, Priscila, Carla, Paula, Cintia}
- $\mathbf{dom}(2, Birads) = \{2, 3\}$
- $\mathbf{dom}(2, Gravidez) = \{sim\}$
- $\mathbf{dom}(2, Fumo) = \{5, 10, 20\}$
- $\mathbf{dom}(2, Hormônio) =$ {estradiol, progesterona, noretisterona}

- $\mathbf{dom}(3, Birads) = \mathbf{dom}(3, Gravidez) = \{1, 2, 3, 5, 9, 10\}$
- $\mathbf{dom}(3, Fumo) = \mathbf{dom}(3, Hormônio) = \{[2, 4], [2, 5], [3, 6], [4, 6], [5, 8], [5, 9], [6, 8]\}$

onde, $\mathbf{dom}(i, p)$ é um conjunto de valores tomados pelo $i$-ésimo termo de um átomo $p$.

Considerando a paciente Maria, o banco de dados informa que sua última gravidez foi no ponto 3 (em 15/11/2005), depois ela fumou durante o intervalo $[4, 6]$ (de 22/12/2005 a 05/02/2006), usou o hormônio *estradiol* durante o intervalo $[5, 8]$ (de 20/01/2006 a 12/03/2006) e no ponto 10 (em 20/04/2006) ela apresentou Bi-Rads categoria 3.

### 3.1.3 O Padrão Temporal Híbrido

Com a LTI adaptada para servir de formalismo para especificar o *pth* já entendida e com o banco de dados onde o *pth* pode ser minerado já definido, o padrão temporal híbrido é definido formalmente agora.

**Definição 3.1.2 (Padrão Temporal Híbrido).** Um **padrão temporal híbrido** é uma tripla $(K, \mathcal{E}, \mathcal{T})$, onde:

- $\mathcal{E}$ é conjunto de átomos de dados da forma $p(t_1, \ldots, t_n, \tau)$, com $p \in \mathcal{P}red$, $t_i \in \mathcal{C}onst \cup \mathcal{V}ar$ com , e $\tau \in \mathcal{V}_{temp}$.
- $K$ é um átomo de dados especial da forma $K(x_1, \ldots, x_n)$, cujas variáveis $x_1, \ldots, x_n$ aparecem em $\mathcal{E}$. O átomo de dados $K$ é chamado de *átomo de referência*.
- $\mathcal{T}$ é um conjunto de átomos temporais da forma $r(\tau, \gamma)$, com $r \in \mathcal{P}_{temp}$ e $\tau, \gamma \in \mathcal{V}_{temp}$.

Cada variável temporal que ocorre em $\mathcal{T}$ tem uma e somente uma ocorrência em $\mathcal{E}$, e cada variável temporal que ocorre em $\mathcal{E}$ tem pelo menos uma ocorrência em $\mathcal{T}$. Abstraindo o que foi dito anteriormente, cada variável temporal intervalar ou pontual $\tau$ que ocorre em um *pth* é associada a um único átomo de dados $p(t_1, \ldots, t_n, \tau)$, e pode ser utilizada uma ou mais vezes nos relacionamentos com outras variáveis temporais.

Um *pth* $P = (K, \mathcal{E}, \mathcal{T})$ pode ser expresso sob o ponto de vista da lógica temporal pela seguinte fórmula da LTI adaptada:

$$\exists\tau_1 \ldots \exists\tau_n \; \exists x_1 \exists x_2 \ldots \exists x_m \; (\bigwedge_{p\in\mathcal{E}} p \wedge \bigwedge_{r\in\mathcal{T}} r) \wedge K$$

onde, $\tau_1, \ldots, \tau_n$ são as variáveis temporais que aparecem em $\mathcal{T}$ e $x_1, \ldots, x_m$ são as variáveis de dados que aparecem em $\mathcal{E}$ e não aparecem em $K$. Esta fórmula é chamada de *fórmula da LTI adaptada associada a* $P$ e é denotada por $Q_P$.

**Exemplo 3.1.3** Seja $P = (K, \mathcal{E}, \mathcal{T})$ um *pth* com $K = Paciente(x)$, $\mathcal{E} = \{Fumo(x, y, e), Birads(x, z, s)\}$ e $\mathcal{T} = \{before(e, s)\}$. A fórmula da LTI adaptada associada a $P$ é a seguinte:

$$Q_P = \exists(e, s, y, z)(Fumo(x, y, e) \wedge Birads(x, z, s) \wedge before(e, s)) \wedge (x)$$

Intuitivamente, este *pth* significa que uma paciente $x$ fumou $y$ cigarros por dia durante o intervalo $e$ e posteriormente, no ponto $s$, após ter parado de fumar, apresentou Bi-Rads $z$. Note que o interesse é em analisar somente o comportamento das pacientes registradas na tabela *Paciente*. Por esta razão, $K = Paciente(x)$ é chamado de *átomo de referência.*

Neste problema de mineração interessa uma classe específica de *pth's*, onde o conjunto de átomos temporais $\mathcal{T}$ é *consistente* e *completo.* Um conjunto de átomos temporais $\mathcal{T}$ é considerado consistente se todas as suas variáveis temporais podem ser *instanciadas* de modo consistente. Um conjunto de átomos temporais $\mathcal{T}$ é considerado completo se todas as suas variáveis temporais estiverem relacionadas, explicita ou implicitamente, pelos predicados temporais de $\mathcal{P}_{temp}$.

**Definição 3.1.3 (Instanciação)**. Uma **instanciação** de uma variável temporal intervalar $e$ (resp. uma variável temporal pontual $s$) é um mapeamento $\upsilon(e)$ associando um intervalo $[i, j]$ a $e$ (resp. um ponto $i$ a $s$), onde $i$ e $j$ são números naturais mapeados em datas.

**Exemplo 3.1.4** Considere a variável temporal intervalar $e$ e a variável temporal pontual $s$. A variável $e$ pode ser mapeada, por exemplo, no intervalo $[2, 4]$ que, de acordo com a linha do tempo da figura 3.4, é mapeado nas datas $[15/10/2005, 22/12/2005]$. A variável $s$ pode ser mapeada, por exemplo, no ponto 5 que é mapeado na data 20/01/2006. Para esses dois mapeamentos a fórmula $before(e, s)$ é verdadeira, e a fórmula $meets(e, s)$ é falsa, pois $e$ não encontra com $s$.

**Definição 3.1.4 (Consistência)**. Seja $\mathcal{T} = \{A_1, \ldots, A_n\}$ um conjunto de átomos temporais. O conjunto $\mathcal{T}$ é **consistente** se existe uma instanciação das variáveis temporais, tal que a fórmula $A_1 \wedge \ldots \wedge A_n$ seja satisfeita, ou seja, para cada átomo $A_i(\tau, \gamma) \in \mathcal{T}, (\upsilon(\tau), \upsilon(\gamma))$ pertence a interpretação de $A_i$ de acordo com $\mathcal{T}_{temp}$.

**Exemplo 3.1.5** Considere o conjunto $\mathcal{T} = \{before(e, f)$,$before(f, s)$,$before(s, e)\}$, e os mapeamentos $\upsilon(e) = [01, 03]$, $\upsilon(f) = [05, 06]$ e $\upsilon(s) = 07$, resultando, de acordo com a linha do tempo da figura 3.4, em:

$$\begin{aligned}\mathcal{T} = \{&before([10/10/2005, 15/11/2005], [20/01/2006, 05/02/2006]),\\ &before([20/01/2006, 05/02/2006], 15/02/2006),\\ &before(15/02/2006, [10/10/2005, 15/11/2005])\}\end{aligned}$$

O conjunto $\mathcal{T}$ é *inconsistente*, pois os dois primeiros átomos implicam que $e$ ocorre antes de $s$ e o terceiro átomo diz que $s$ ocorre antes de $e$, sendo assim, não existe uma maneira de instanciar as variáveis temporais $e, f$ e $s$ de modo que as três fórmulas temporais sejam simultaneamente verdadeiras. Entretanto, o conjunto $T' = \{before(e, f)$, $before(f, s)$, $before(e, s)\}$, é *consistente*, uma vez que nenhum átomo temporal contradiz as informações transmitidas pelos outros.

**Definição 3.1.5 (Completude)**. Seja $\mathcal{T}$ um conjunto de átomos temporais. $\mathcal{T}$ é **completo** se para cada par de variáveis temporais $(\tau, \gamma) \in \mathcal{T}$ é possível inferir um relacionamento entre $\tau$ e $\gamma$, ou seja, existe um predicado $r \in \mathcal{P}_{temp}$, tal que, $r(\tau, \gamma)$ ou $r(\gamma, \tau)$.

**Exemplo 3.1.6** O conjunto $\mathcal{T} = \{before(e, s), starts(s, g)\}$ é *completo* porque o relacionamento entre $e$ e $g$ é completamente determinado. Embora não apareça explicitamente no conjunto a única possibilidade é $before(e, g)$. Entretanto, o conjunto $\mathcal{T} = \{before(e, s)$, $during(s, g)\}$ não é completo porque o relacionamento entre $e$ e $g$ não é deterministicamente implicado pelos dois relacionamentos $before(e, s)$ e $during(s, g)$, podendo ser $before(e, g)$, $meets(e, g)$, $overlaps(e, g)$, $starts(e, g)$ e $during(e, g)$.

A razão pela qual o conjunto de átomos temporais $\mathcal{T}$ deve ser completo e consistente é que nessas condições as variáveis temporais de um *pth* podem ser ordenadas de modo natural.

O motivo da ordenação das variáveis temporais de um conjunto de átomos temporais $\mathcal{T}$ ficará claro na proposição 3.1.3. Para que esta ordenação seja possível, primeiramente é necessário generalizar o domínio dos dados temporais. Como mostrado anteriormente, o domínio $\mathcal{I}$ é considerado para as variáveis temporais intervalares e o domínio $\mathbb{N}$ é considerado para as variáveis temporais pontuais. É importante relembrar que um número natural $n$ pode ser visto como um intervalo $[n, n]$. Sendo assim, considere um novo conjunto $\mathcal{I}^* = \mathcal{I} \cup \mathbb{N}$ constituído por intervalos $[i, j]$, com $i \leq j$, ou seja, constituído por intervalos não nulos $(i < j)$ e por intervalos nulos $(i = j)$ considerados pontos.

**Definição 3.1.6 (Ordem total sobre os intervalos).** Seja $[a, b]$ e $[c, d]$ dois intervalos pertencentes a $\mathcal{I}^*$. É dito que $[a, b] \leq [c, d]$ se e somente se uma das seguintes condições forem satisfeitas:

1. $a = c$ e $b = d$
2. $b < d$
3. $b = d$ e $a > c$

É evidente que se $[a, b] \leq [c, d]$, então, $b \leq d$. Entretanto, se $[a, b] \leq [c, d]$ e $b = d$, então $a \geq c$. Com isso é possível provar através da reflexividade, anti-simetria e transitividade, que $\leq$ é uma ordem total sobre $\mathcal{I}^*$.

- **Reflexividade:** A relação $[a, b] \leq [a, b]$ é verificada desde que a condição (1) seja verificada. Assim, a propriedade da reflexividade é garantida.
- **Anti-simetria:** Considerando as relações $[a, b] \leq [c, d]$ e $[c, d] \leq [a, b]$. Então, $b \leq d$ e $d \leq b$. Logo, $b = d$. Partindo da condição (3), conclui-se que $a > c$ e $c > a$. Logo, $a = c$. Assim, a condição (1) é verificada e conclui-se que $[a, b] = [c, d]$. Portanto, a propriedade da anti-simetria é garantida.
- **Transitividade:** Considerando as relações $[a, b] \leq [c, d]$ e $[c, d] \leq [e, f]$. Assim, $b \leq d \leq f$. Então $b \leq f$. Se $b < f$, tem-se que $[a, b] \leq [e, f]$. Se $b = f$, então $b = d = f$. Partindo do fato de que $[a, b] \leq [c, d]$ e $b = d$, conclui-se que $a \geq c$. De maneira similar pode-se provar que $c \geq e$. Assim, $a \geq e$. Desde que $b = f$ e $a \geq e$, conclui-se que $[a, b] \leq [e, f]$.

Considere dois intervalos $[a, b]$ e $[c, d]$ pertencentes a $\mathcal{I}^*$. Se $b < d$, então $[a, b] \leq [c, d]$. Se $b > d$, então $[c, d] \leq [a, b]$. Se $b = d$ e $a > c$, então $[a, b] \leq [c, d]$. Se $b = d$ e $a < c$, então $[c, d] \leq [a, b]$. Se $b = d$ e $a = c$, então $[a, b] = [c, d]$. Com isso, através das três propriedades anteriores, foi provado que a relação de ordem $\leq$ é total.

Para qualquer $[a, b]$ e $[c, d]$ pertencentes a $\mathcal{I}^*$, é denotado por $[a, b] < [c, d]$ o fato que $[a, b] \leq [c, d]$ e $[a, b] \neq [c, d]$.

Com a definição de ordem total sobre os intervalos bem clara, é possível estabelecer um ordenamento único das variáveis temporais que aparecem em um conjunto de átomos temporais $\mathcal{T}$ completo e consistente.

**Proposição 3.1.1** Seja $\mathcal{T} = \{A_1, \ldots, A_k\}$ um conjunto completo e consistente de átomos temporais e $\upsilon$ e $\upsilon'$ dois mapeamentos das variáveis temporais de $\mathcal{T}$, tal que cada fórmula $A_i$ de $\mathcal{T}$ seja simultaneamente verdadeira, ou seja, $(\mathcal{T}_{temp}, \upsilon) \models \mathcal{T}$ e $(\mathcal{T}_{temp}, \upsilon') \models \mathcal{T}$. Seja também $\tau$ e $\gamma$ duas variáveis temporais que ocorrem em $\mathcal{T}$. Assim:

1. Se $\upsilon(\tau) \leq \upsilon(\gamma)$, então $\upsilon'(\tau) \leq \upsilon'(\gamma)$

**Prova:** Pelo fato de $\mathcal{T}$ ser completo e consistente, é possível afirmar sem perda de generalidade que para qualquer par de variáveis $(\tau, \gamma) \in \mathcal{T}$ existe um predicado temporal $r$, tal que $(\mathcal{T}_{temp}, \upsilon) \models r(\tau, \gamma)$ e $(\mathcal{T}_{temp}, \upsilon') \models r(\tau, \gamma)$. Assim, $\upsilon(\tau) \leq \upsilon(\gamma)$ se e somente se $r(\tau, \gamma)$ for $before(\tau, \gamma)$ ou $meets(\tau, \gamma)$ ou $overlaps(\tau, \gamma)$ ou $starts(\tau, \gamma)$ ou $during(\tau, \gamma)$ ou $finishes(\tau, \gamma)$ ou $\tau =_t \gamma$. Como $(\mathcal{T}_{temp}, \upsilon') \models r(\tau, \gamma)$, então, para todo mapeamento $\upsilon'$ considerado, $\upsilon'(\tau) \leq \upsilon'(\gamma)$.

**Proposição 3.1.2 (Ordenamento único das variáveis temporais)**. Seja um conjunto $\mathcal{T} = \{A_1, \ldots, A_k\}$ completo e consistente. Como mostrado anteriormente, se $r(\tau, \gamma)$, então $\upsilon(\tau) \leq \upsilon(\gamma)$. Como por definição não existe repetição entre as variáveis temporais do conjunto $\mathcal{T}$, ou seja, $\forall\tau, \forall\gamma \in \mathcal{T}, \tau \neq \gamma$, existe um ordenamento $\tau <_{\mathcal{T}} \gamma$, onde $<_{\mathcal{T}}$ é um novo símbolo que representa um ordenamento sobre as variáveis temporais de $\mathcal{T}$. Para generalizar essa idéia, considere o conjunto $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ das variáveis temporais de $\mathcal{T}$. Assim, existe um ordenamento único $\gamma_1 <_{\mathcal{T}} \gamma_2 <_{\mathcal{T}} \ldots <_{\mathcal{T}} \gamma_n$, que satisfaz a fórmula:

$$\forall\gamma_1 \ldots \forall\gamma_n((A_1 \wedge \ldots \wedge A_k) \rightarrow \gamma_1 <_{\mathcal{T}} \gamma_2 <_{\mathcal{T}} \ldots <_{\mathcal{T}} \gamma_n)$$

**Proof.** Pelo fato de $\mathcal{T} = \{A_1, \ldots, A_k\}$ ser completo e consistente e de não ocorrer repetições entre entre suas variáveis temporais, existe um mapeamento $\upsilon$, tal que $(\mathcal{T}_{temp}, \upsilon) \models \{A_1 \wedge \ldots \wedge A_k\}$ e $(\mathcal{T}_{temp}, \upsilon) \models \tau \neq \gamma$, para cada $\tau, \gamma$ aparecendo em $\mathcal{T}$. Considerando um conjunto de intervalos $\{\upsilon(\gamma) \mid \gamma \; aparece \; em \; \mathcal{T}\}$, é possível ordená-los como $\upsilon(\gamma_1) < \upsilon(\gamma_2) < \ldots < \upsilon(\gamma_n)$. Com isso, é possível ordenar as variáveis temporais como $\gamma_1 <_{\mathcal{T}} \gamma_2 <_{\mathcal{T}} \ldots <_{\mathcal{T}} \gamma_n$. Considerando $\upsilon'$ outro mapeamento, tal que $(\mathcal{T}_{temp}, \upsilon') \models (A_1 \wedge \ldots \wedge A_k)$ e $(\mathcal{T}_{temp}, \upsilon') \models \tau \neq \gamma$, concluimos a partir da proprosição 3.1.1 que $\upsilon'(\gamma_1) < \upsilon'(\gamma_2) < \ldots < \upsilon'(\gamma_n)$. Assim, a fórmula anterior é satisfeita por $\mathcal{T}$.

**Exemplo 3.1.7** Seja o conjunto de átomos temporais $\mathcal{T} = \{before(e, s), starts(s, g)\}$ completo, consistente e sem repetição. A ordem sobre o conjunto das variáveis temporais $\{e, s, g\}$ é $e <_{\mathcal{T}} s <_{\mathcal{T}} g$. De fato, se para qualquer mapeamento $\upsilon$ das variáveis temporais que ocorrem em $\mathcal{T}$, $\upsilon(e) = [a_e, b_e]$, $\upsilon(s) = [a_s, b_s]$ e $\upsilon(g) = [a_g, b_g]$, tal que os predicados de $\mathcal{T}$ sejam simultaneamente verdadeiros, então $\upsilon(e) < \upsilon(s) < \upsilon(g)$.

**Proposição 3.1.3 (Ordenamento único dos átomos de dados)**. Seja um *pth* $P = (K, \mathcal{E}, \mathcal{T})$ com $\mathcal{T}$ consistente e completo, e o ordenamento único das variáveis temporais de $\mathcal{T}$, definido acima. Nestas condições é possível ordenar os átomos de dados de $P$ de um modo único.

**Proof.** Por definição, cada variável temporal $\gamma \in \mathcal{T}$ é associada a um único átomo de dados $p(a_1, \ldots, a_k, \gamma) \in \mathcal{E}$. Assim, é possível ordenar os átomos de $\mathcal{E}$ de acordo com a ordem das variáveis temporais de $\mathcal{T}$.

**Exemplo 3.1.8** Seja $P = (K, \mathcal{E}, \mathcal{T})$ um *pth*, onde, $\mathcal{T} = \{before(e, s), \; starts(s, g)\}$ é um conjunto completo e consistente de átomos temporais. A ordem sobre o conjunto das variáveis temporais $\{e, s, g\}$ é $e <_{\mathcal{T}} s <_{\mathcal{T}} g$, como mostrado no exemplo anterior. Como cada variável temporal é associada a um único átomo de dados, pode-se considerar $\mathcal{E} = \{p_1(a_1, \ldots, a_k, e), p_2(a_1, \ldots, a_k, s), p_3(a_1, \ldots, a_k, g)\}$. Portanto, o ordenamento dos átomos de $\mathcal{E}$ segue o ordenamento das variáveis temporais de $\mathcal{T}$.

Todo este trabalho de ordenamento das variáveis temporais e, conseqüentemente dos átomos de dados de um *pth* foi realizado com o objetivo de simplificar sua interpretação,

uma vez que um *pth* possui uma complexa estrutura dividida em três partes, $K$, $\mathcal{E}$ e $\mathcal{T}$, como apresentado na definição 3.1.2. Portanto, deste ponto em diante sempre que um *pth* $P = (K, \mathcal{E}, \mathcal{T})$ for apresentado, o conjunto $\mathcal{E}$ pode ser considerado uma *seqüência* de átomos $p_1 <_{\mathcal{T}} \ldots <_{\mathcal{T}} p_n$, ordenados de acordo com a ordem das variáveis temporais de $\mathcal{T}$. Assim, $\mathcal{E}$ é denotado por $< p_1, \ldots, p_n >$. Isto permite obter padrões mais específicos a partir de padrões mais gerais acrescentando novos átomos de dados no fim da seqüência de $\mathcal{E}$, como será mostrado na seção 4.3.3 do capítulo 4.

### 3.1.4 O Critério de Seleção

Nesta seção será definido o conceito de *suporte* de um *pth* $P$ sobre um banco de dados temporal $BD$. Além de definir o suporte de um *pth*, será introduzida também a idéia de *pth* freqüente.

**Definição 3.1.7 (Suporte de um *pth*).** Seja $BD$ um banco de dados temporal sobre o esquema $\mathbf{R} = \{K, R_1, \ldots, R_n\}$, e $P = (K(x_1, \ldots, x_m), \{E_1, \ldots, E_s\}, \{T_1, \ldots, T_l\})$ um *pth*.

O **suporte** de $P$ com respeito ao banco de dados $BD$ denotado por $sup(P/BD)$, é a porcentagem de tuplas em $BD$ que satisfazem $P$. $sup(P/BD)$ é definido por:

$$sup(P/BD) = \frac{|\{u \in K | u \models Q_P\}|}{|K|}$$

onde, $Q_P$ é uma consulta relacional (fórmula da LTI associada a $P$) $\exists\tau_1 \ldots \exists\tau_n \exists y_1 \ldots \exists y_k$ $(E_1 \wedge E_2 \wedge \ldots \wedge E_s \wedge T_1 \wedge T_2 \wedge \ldots \wedge T_l)$, onde, $\tau_1 \ldots \tau_n$ são variáveis temporais que aparecem em $\mathcal{T}$ e $y_1 \ldots y_k$ são variáveis de dados que aparecem em $\mathcal{E}$ e não aparecem no átomo de referência $K(x_1, \ldots, x_m)$. A expressão $u \models Q_P$ significa que $u$ satisfaz a consulta relacional $Q_P$ quando executada sobre BD.

Um *pth* $P$ é **freqüente** com respeito a $BD$ se dado um limiar mínimo de suporte $0 \leq \alpha \leq 1$, $sup(P/BD) \geq \alpha$.

**Exemplo 3.1.9** Considerando o banco de dados $BD$ apresentado na figura 3.4 e o *pth* $P = (Paciente(x), \{Hormônio(x, y, e), Birads(x, 3, s)\}, \{before(e, s)\})$. A consulta relacional $Q_P$ pode ser especificada também no formalismo da álgebra relacional pela expressão $\Pi_{NomePac}(Paciente \bowtie Hormônio \bowtie Birads)$.

A resposta de $Q_P$ é $\{(Maria, Paula)\}$. Pois *Priscila* fez uso do hormônio *progesterona* e não apresentou nenhuma categoria de Bi-Rads, *Carla* apresentou Bi-Rads categoria 3 durante, não depois, de ter usado o hormônio *noretisterona* e *Cintia* não fez uso de hormônio e não apresentou Bi-rads algum, por isso, não são respostas de $Q_P$. Assim, o suporte do *pth* $P$ é $\frac{2}{5} = 0.4\%$. Se $\alpha = 0.2\%$, então, $P$ é um *pth* freqüente em BD.

### 3.1.5 Tarefa de Mineração Básica

A tarefa de mineração considerada básica para o problema de mineração proposto nesta dissertação será apresentada logo abaixo. Na seção 4.2 será apresentada a tarefa de mineração principal, incluindo restrições de expressões regulares.

- **Dado**: Um banco de dados temporal $BD$ e um limiar mínimo de suporte $\alpha$.
- **Encontrar**: Todos os *pth* freqüentes com relação a $BD$ e $\alpha$.

## 3.2 Considerações Finais do Capítulo

Neste capítulo o problema de mineração de padrões temporais híbridos foi amplamente detalhado. Inicialmente foi definida a Lógica Temporal de Intervalos de Allen e as adaptações realizadas para ela servir de formalismo para especificar o novo padrão temporal. A LTI adaptada considera casos onde o tempo é representado em termos de pontos e intervalos, uma vez que a LTI original considera somente tempo intervalar.

Posteriormente foi definido o banco de dados temporal onde os *pth's* podem ser minerados e em seguida o padrão temporal híbrido. Um extenso trabalho foi realizado sobre o conjunto de átomos temporais com o objetivo de simplificar a representação e, conseqüentemente, o entendimento dos *pth's*. Para isso, foram desenvolvidas técnicas para visualizá-lo como uma seqüência de átomos de dados, permitindo também que ele seja especializado inserindo um novo átomo de dados no fim da seqüência.

Após a apresentação formal do *pth* foram definidos os conceitos de suporte e *pth* freqüente sobre bancos de dados temporais. Finalmente, o capítulo foi encerrado com a apresentação da tarefa básica deste problema de mineração.

# Capítulo 4

# Restrições no Processo de Mineração

Neste capítulo serão apresentadas as restrições que foram introduzidas no processo de mineração com o objetivo de otimizar a busca por novos *pth's*. A primeira restrição é especificada por expressões regulares similares às utilizadas nos algoritmos da família SPIRIT [12] e SPIRIT-Log [14]. Esse tipo de restrição reduz o espaço de busca dos *pth's* e permite que o usuário controle do processo de mineração. Além das restrições de expressões regulares serão apresentadas também restrições especificadas por operadores de especialização. Esses operadores possuem propriedades especiais para controlar a geração de novos *pth's*, garantindo a eficiência e eficácia do método.

## 4.1 Expressões Regulares sobre os PTH's

Os sistemas convencionais de mineração permitem ao usuário apenas definir o suporte mínimo como mecanismo de restringir os padrões a serem minerados. Com isso, ele é sobrecarregado com uma enorme quantidade de padrões que não o interessam. Os algoritmos da família SPIRIT [12] e SPIRIT-Log [14], além do suporte mínimo, permitem o uso de expressões regulares como uma ferramenta para incorporar o foco do usuário ao processo de mineração, promovendo assim a redução do espaço de busca dos padrões. Nesta dissertação são definidas expressões regulares sobre um conjunto de *modos* chamado *alfabeto*, e todo *pth* gerado deve satisfazer uma determinada expressão regular. Portanto, a expressão regular considerada tem a função de restringir o "formato" dos *pth's*. É impor-

tante salientar que os algoritmos da família SPIRIT e SPIRIT-Log mineram padrões seqüenciais. Portanto, as restrições de expressões regulares são definidas considerando o tempo como pontual. Neste trabalho está sendo proposto um método de utilizar tais restrições de expressões regulares na mineração de padrões temporais híbridos, onde o tempo é pontual e/ou intervalar.

**Definição 4.1.1 (Modo)** Um **modo** sobre um esquema $\mathbf{R}$ é uma expressão da forma $p(u_1, ..., u_n, \#)$, onde $p$ é um predicado pertencente a $\mathbf{R}$ com aridade $n+1$, $u_i \in \mathcal{V}ar$ ou $u_i$ é um símbolo \$, e $\#$ é um novo símbolo. Um modo caracteriza e identifica os atributos das relações de $\mathbf{R}$ e serve de certa forma como um molde para a construção de um átomo. Um átomo $A$ *é de acordo com um modo* $p(u_1, ..., u_n, \#)$ ($A$ foi gerado a partir de $p(u_1, ..., u_n, \#)$) se $A = p(x_1, ..., x_n, \tau)$ e para cada $i = 1, ..., n$, temos:

- Se $u_i \in \mathcal{V}ar$, então $x_i = u_i$;
- Se $u_i = \$$, então $x_i \in \mathcal{V}ar$ ou $x_i \in \mathcal{C}onst$;
- $\#$ é substituído por $\tau \in \mathcal{V}_{temp}$, ou seja, $\tau \in \mathcal{V}_{it}$ ou $\tau \in \mathcal{V}_{pt}$.

**Exemplo 4.1.1** Considere o esquema $\mathbf{R} = \{Paciente(NomePac),\ Birads(NomePac, Categoria, T_{pt}),\ Fumo(NomePac, Qtde, T_{it}),\ Hormônio(NomePac, NomeHorm, T_{it})\}$, onde $Paciente(NomePac)$ é uma tabela de referência e portanto, $NomePac$ é um atributo de referência. Assim, um possível alfabeto de modos sobre este esquema poderia ser $\{Paciente(x),\ Birads(x, \$, \#),\ Fumo(x, \$, \#),\ Hormônio(x, \$, \#)\}$. Note que a variável $x$ é uma variável de referência, portanto, estes modos informam que o primeiro atributo de cada tabela é de referência e os valores armazenados nestas colunas pertencem a um mesmo domínio, neste caso, nome de pacientes. Considere agora os átomos $Hormônio(x, progesterona, e)$ e $Hormônio(y, progesterona, e)$. O primeiro é de acordo com o modo $Hormônio(x, \$, \#)$. Em contrapartida, o segundo não é de acordo com o modo, pois, como $x$ é uma variável no modo, ela deve ser mantida no átomo.

Um *alfabeto de modos* sobre $\mathbf{R}$ é denotado pelo símbolo $\Sigma$. São consideradas *linguagens* especificadas por expressões regulares sobre o alfabeto $\Sigma$ com o objetivo de reduzir o

tamanho do espaço de busca dos *pth's*. Relembrando, uma linguagem sobre um alfabeto $\Sigma$ é um conjunto de símbolos ou *strings* sobre $\Sigma$.

**Definição 4.1.2 (Expressão regular)** Uma **expressão regular** $\mathcal{R}$ sobre um alfabeto de modos $\Sigma = \{p_1(u_1, \ldots, u_n, \#), p_2(u_1, \ldots, u_n, \#), \ldots, p_k(u_1, \ldots, u_n, \#)\}$ pode ter a forma $p_1(u_1, \ldots, u_n, \#)^*\ p_2(u_1, \ldots, u_n, \#)$, que neste caso especifica uma linguagem, onde, * significa que o predicado $p_1$ pode não ocorrer ou ocorrer uma ou mais vezes, e obrigatoriamente deve terminar com o predicado $p_2$.

**Exemplo 4.1.2** Considere o alfabeto de modos $\Sigma = \{Hormônio(x, \$, \#), Fumo(x, \$, \#), Birads(x, \$, \#)\}$, e a expressão regular $\mathcal{R} = Hormônio(x, \$, \#)^* Fumo(x, \$, \#) Birads(x, \$, \#)$ sobre $\Sigma$. Uma linguagem especificada pela restrição $\mathcal{R}$ pode ser $Hormônio(x, \$, \#)$ $Hormônio(x, \$, \#) Fumo(x, \$, \#) Birads(x, \$, \#)$.

Para facilitar a representação de uma expressão regular cada modo é rotulado com uma letra, por exemplo, $w_i = p_i(u_1, \ldots, u_n, \#)$, significa que $w_i$ representa $p_i$. Por motivos de simplificação, a mesma notação $\mathcal{R}$ é considerada para denotar a expressão regular e a linguagem associada a ela. Assim, dada uma expressão regular $\mathcal{R}$ sobre um alfabeto $\Sigma$, um *pth* $(K, \mathcal{E}, \mathcal{T})$ satisfaz $\mathcal{R}$ se: $\mathcal{E}$ é uma seqüência de átomos de dados $< p_1, \ldots, p_n >$ e existe uma *string* $w_1, \ldots, w_n \in \mathcal{R}$, onde $p_i$ é de acordo com $w_i$ para cada $i = 1, \ldots, n$. O conjunto de *strings* que verificam a expressão regular $\mathcal{R}$ e denotado por $W(\mathcal{R})$ e o conjunto de *prefixos* de *strings* em $W(R)$ é denotado por $P_{ref}(\mathcal{R})$, ou seja, $P_{ref}(\mathcal{R}) = \{w \mid ww' \in \mathcal{R}\ para\ algum\ w'\}$. Essas notações serão importantes a seguir.

**Definição 4.1.3 (Espaço de busca)** Seja um esquema de banco de dados $\mathbf{R} = \{K, R_1, \ldots, R_n\}$, um alfabeto de modos $\Sigma$ sobre $\mathbf{R}$ e uma expressão regular $\mathcal{R}$ sobre $\Sigma$. O **espaço de busca** definido por $\mathcal{R}$ é um conjunto de *pth's* $(K, \mathcal{E}, \mathcal{T})$, tal que, $\mathcal{E} = < p_1(t_1^1, \ldots, t_l^1, \tau_1), \ldots, p_n(t_1^n, \ldots, t_l^n, \tau_n) >$ satisfaz as seguintes propriedades:

1. Deve existir uma *string* $w_1 \ldots w_n \in W(\mathcal{R})$ tal que $p_i$ é de acordo com $w_i$, para cada $i = 1, \ldots, n$. A *string* $w_1 \ldots w_n$ é chamada de *palavra associada ao pth.*

2. Para todo átomo de dados $p_i(t_1^i, \ldots, t_l^i, \tau_i) \in \mathcal{E}$ de acordo com um modo $w_i = p_i(u_1^i, \ldots, u_l^i, \#)$, se $u_k^i = \$$ e $t_k^i \in \mathcal{V}ar$, então $t_k^i$ tem apenas uma ocorrência em $\mathcal{E}$.

É denotado por $\mathcal{W}(\mathcal{R})$ o espaço de busca dos *pth's* especificado por $\mathcal{R}$, e por $\mathcal{P}ref(\mathcal{R})$, o conjunto de *pth's* definidos de modo similar a $\mathcal{W}(\mathcal{R})$, por considerar *strings* $w_1 \dots w_n \in P(\mathcal{R})$ ao invés de $W(\mathcal{R})$ na condição (1). Os *pth's* pertencentes a $\mathcal{W}(\mathcal{R})$ são chamados de *válidos*, e os *pth's* pertencentes a $\mathcal{P}ref(\mathcal{R})$ de *prefixos-válidos* ou simplesmente *p-válidos*.

Como será visto no próximo capítulo, na fase da geração do algoritmo MILPRIT* somente *pth's* pertencentes a $\mathcal{P}ref(\mathcal{R})$ são gerados. A razão pela qual são gerados *pth's* p-válidos ao invés de válidos ficará clara na seção 5.1.

**Proposição 4.1.1** Seja $\mathcal{W}(\mathcal{R})$ e $\mathcal{P}ref(\mathcal{R})$ dois espaços de busca, onde $\mathcal{R}$ é uma expressão regular sobre $\Sigma$. Então $\mathcal{W}(\mathcal{R}) \subseteq \mathcal{P}ref(\mathcal{R})$.

A prova desta proposição foi dada na definição anterior, ou seja, para computar os *pth's* que pertencem a $\mathcal{W}(\mathcal{R})$ é suficiente computar os que pertencem a $\mathcal{P}ref(\mathcal{R})$.

**Exemplo 4.1.3** Considere a expressão regular $\mathcal{R} = Hormônio(x, \$, \#)^* Fumo(x, \$, \#)$ $Birads(x, \$, \#)$ e os *pth's* $P_1$, $P_2$ e $P_3$ definidos por:

$P_1 = (Paciente(x), \{Hormônio(x, estradiol, e_1), Fumo(x, 10, f_2), \ Birads(x, 3, s_3)\},$
$\{during(e_1, f_2), before(e_1, s_3), before(f_2, s_3)\})$

$P_2 = (Paciente(x), \{Hormônio(x, estradiol, e_1), Fumo(x, 10, f_2)\}, \{before(e_1, f_2)\})$

$P_3 = (Paciente(x), \{Hormônio(x, y, e_1), Hormônio(x, y, e_2)\}, \{before(e_1, e_2)\})$

O *pth* $P_1$ pertence a $\mathcal{W}(\mathcal{R})$ e o *pth* $P_2$ pertence a $\mathcal{P}ref(\mathcal{R})$. Entretanto, o *pth* $P_3$ não pertence a $\mathcal{P}ref(\mathcal{R})$ pois a propriedade (2) da definição 4.1.3 não é verificada.

## 4.2 Tarefa de Mineração com Restrições

A principal tarefa de mineração desta dissertação inclui restrições de expressões regulares estipuladas pelo usuário além de um suporte mínimo também estipulado pelo usuário.

- **Dado**: Um banco de dados $BD$, um suporte mínimo $\alpha$, e uma restrição $\mathcal{R}$.
- **Encontrar**: Todos os *pth's* freqüentes com relação a $BD$, $\alpha$ e que satisfazem $\mathcal{R}$.

## 4.3 Operadores de Especialização

A maioria dos algoritmos de mineração de dados tipicamente computa todas as soluções através do uso de um algoritmo de busca completo. Quando uma busca completa é executada, o resultado final pode conter padrões duplicados. Com isso, um tempo precioso é desperdiçado simplesmente computando algo desnecessário [37].

No protótipo MILPRIT [16], a especialização dos padrões é realizada por *operadores de especialização* (*instanciação* e *extensão*) com as mesmas características (*completos* e *optimais*) dos operadores de refinamento utilizados por Lee e Raedt no SeqLog [13] para controlar a busca por padrões. Nesta dissertação os formalismos teóricos definidos no MILPRIT para a geração de novos padrões foram estendidos para o problema proposto. Essa extensão foi realizada com o objetivo de garantir a eficiência e eficácia do método, assim, durante o processo de mineração são gerados todos o padrões possíveis a partir de um dado padrão e nunca são gerados padrões duplicados.

A relação de especialização entre dois padrões é dada por:

**Definição 4.3.1 (Relação de especialização)** Seja $\mathcal{Q}$ um conjunto de *pth's* e $P_1 = (K, \mathcal{E}_1, \mathcal{T}_1)$ e $P_2 = (K, \mathcal{E}_2, \mathcal{T}_2)$ dois *pth's* pertencentes a $\mathcal{Q}$. $P_1$ é mais específico que $P_2$ ou $P_2$ é mais geral que $P_1$, denotado por $P_1 \preceq P_2$, se existe uma substituição $\theta$ das variáveis de $P_2$ nos termos de $P_1$, tal que:

- Para cada $p_i \in \mathcal{E}_2$, existe $p_j \in \mathcal{E}_1$, tal que, $\theta p_i = p_j$
- Para cada $r_i \in \mathcal{T}_2$, existe $r_j \in \mathcal{T}_1$, tal que, $\theta r_i = r_j$

A notação $P_1 \prec P_2$ é utilizada se: $P_1 \preceq P_2$ e $P_1 \neq P_2$. O *pth* $P_2$ é considerados um sub-padrão de $P_1$.

**Exemplo 4.3.1** Seja dois *pth's*:

$$P_1 = (Paciente(x),\ \{Hormônio(x, estradiol, e_1),\ Fumo(x, y, f_2),\ Birads(x, 3, s_3)\},$$
$$\{during(e_1, f_2), before(e_1, s_3), before(f_2, s_3)\})$$
$$P_2 = (Paciente(x),\ \{Hormônio(x, y, e_1), Fumo(x, z, g_2)\}, \{before(e_1, g_2)\})$$

Se for considerada a substituição $\theta y = estradiol$, $\theta z = y$ e $\theta g_2 = f_2$, então $P_1 \prec P_2$.

**Definição 4.3.2 (*Pth's* equivalentes)** Seja $\mathcal{Q}$ um conjunto de *pth's* e $P_1$ e $P_2$ dois *pth's* pertencentes a $\mathcal{Q}$. $P_1$ é **equivalente** a $P_2$ denotado por $P_1 \sim P_2$, se $P_1$ é mais específico que $P_2$ ($P_1 \preceq P_2$) e $P_2$ é mais específico que $P_1$ ($P_2 \preceq P_1$).

**Exemplo 4.3.2** Seja dois *pth's*:

$$P_1 = (Paciente(x),\ \{Hormônio(x, y, e_1), Birads(x, z, t_2)\}, \{before(e_1, t_2)\})$$
$$P_2 = (Paciente(x),\ \{Hormônio(x, w, e_1), Birads(x, h, t_2)\}, \{before(e_1, t_2)\})$$

Se for considerada a substituição $\theta y = w$ e $\theta z = h$, então $P_2 \preceq P_1$, se for considerada a substituição $\theta w = y$ e $\theta h = z$, então $P_1 \preceq P_2$. Portanto, $P_1 \sim P_2$.

**Definição 4.3.3 (Operador de especialização)** Seja $\mathcal{Q}$ um conjunto de *pth's*. Um **operador de especialização** $\rho$ é uma aplicação $\rho : \mathcal{Q} \rightarrow \mathcal{Q}$, tal que:

1. $\rho(P) \preceq P$
2. $\forall P' \in \mathcal{Q}$, se $\rho(P) \preceq P' \preceq P$, então $P' = \rho(P)$ ou $P' = P$

A condição (1) significa que quando um operador de especialização $\rho$ é aplicado sobre um *pth* $P$ é gerado um ou mais *pth's* mais específicos que $P$. A condição (2) significa que quando um operador de especialização $\rho$ é aplicado sobre um *pth* $P$ é gerado um ou mais *pth's* mais específicos que $P$, tal que não existe nenhum *pth* $P'$ mais específico que $P$ e mais geral que $\rho(P)$, ou seja, $\rho(P)$ é imediatamente mais específico que $P$.

**Definição 4.3.4 (Operador de extensão)** Seja $\mathcal{Q}$ um conjunto de *pth's* e $P = (K, \{p_1(y_1, \ldots, y_l, \tau_1)\}, \{\mathcal{T}\})$ um *pth*. Um operador $\rho$ é um **operador de extensão** se existe um átomo $p_2(y_1, \ldots, y_l, \tau_2)$, tal que, $\forall P \in \mathcal{Q}$, $\rho(P) = P' = (K, \{p_1(y_1, \ldots, y_l, \tau_1), p_2(y_1, \ldots, y_l, \tau_2)\}, \{\mathcal{T}'\})$ ou $\rho(P) = P$.

**Definição 4.3.5 (Operador de instanciação)** Seja $\mathcal{Q}$ um conjunto de *pth's* e $P = (K, \{p_1(x, y, \tau_1)\}, \{\mathcal{T}\})$ um *pth*. Um operador $\rho$ é um **operador de instanciação** se existe uma substituíção $\theta\{x \rightarrow a\}$, tal que, $\forall P \in \mathcal{Q}$, $\rho(P) = P' = (K, \{p_1(a, y, \tau_1)\}, \{\mathcal{T}\})$ ou $\rho(P) = P$.

O grau de especialização de um *pth* é medido em termos de um *vetor de refinamento* associado a ele.

**Definição 4.3.6 (Vetor de refinamento)** Seja $\mathcal{Q}$ um conjunto de *pth's* e $P = (K, \mathcal{E}, \mathcal{T})$ um *pth* pertencente a $\mathcal{Q}$. O **vetor de refinamento** associado a $P$ é definido por $v(P) = (n, c)$, onde, $n$ é a cardinalidade do conjunto $\mathcal{E}$ e $c$ é o número de constantes em $\mathcal{E}$. Além disso, o nível de refinamento de $P$ é denotado por $l(P) = n + c$. Múltiplas ocorrências de uma constante são contadas múltiplas vezes.

**Exemplo 4.3.3** Seja três *pth's*:

$P_1 = (Paciente(x), \{Hormônio(x, estradiol, e_1),\ Hormônio(x, y, e_2)\}, \{before(e_1, e_2)\})$

$P_2 = (Paciente(x), \{Hormônio(x, estradiol, e_1),\ Fumo(x, 10, f_2)\}, \{before(e_1, f_2)\})$

$P_3 = (Paciente(x), \{Hormônio(x, estradiol, e_1),\ Fumo(x, 10, f_2),\ Birads(x, h, s_3)\},$
$\{during(e_1, e_2), before(e_1, s_3), meets(f_2, s_3)\})$

O vetor de refinamento de $P_1$ é $v(P_1) = (2, 1)$ e o nível de refinamento de $P_1$ é $l(P_1) = 3$. O vetor de refinamento de $P_2$ é $v(P_2) = (2, 2)$ e o nível de refinamento de $P_2$ é $l(P_2) = 4$. O vetor de refinamento de $P_3$ é $v(P_3) = (3, 2)$ e o nível de refinamento de $P_3$ é $l(P_3) = 5$. Pode-se dizer que $P_2$ é imediatamente mais específico que $P_1$ e $P_3$ é imediatamente mais específico que $P_2$.

**Proposição 4.3.1** Todo operador de instanciação e todo operador de extensão é um operador de especialização.

**Prova:** Todo operador de instanciação $\rho$ aplicado sobre um *pth* $P$ substitui uma única variável de $P$ resultando em um *pth* $P'$ imediatamente mais específico que $P$, ou resulta no próprio $P$, portanto, $\rho(P) \preceq P$ é não existe nenhum *pth* $P'$, tal que, $\rho(P) \preceq P' \preceq P$. Assim, as duas condições da definição 4.3.3 são satisfeitas, portanto, todo operador de instanciação é um operador de especialização. Todo operador de extensão $\rho$ aplicado sobre um *pth* $P$ insere um único átomo em $P$ resultando em um *pth* $P'$ imediatamente mais específico que $P$, ou resulta no próprio $P$, portanto, $\rho(P) \preceq P$ é não existe nenhum *pth* $P'$, tal que, $\rho(P) \preceq P' \preceq P$. Assim, as duas condições da definição 4.3.3 são satisfeitas, portanto, todo operador de extensão é um operador de especialização.

Para garantir a eficiência e eficácia do método, a classe de todos os operadores de especialização deve ser *completa* e *optimal*.

**Definição 4.3.7 (Classe de operadores de especialização completa)** Uma **classe $\mathcal{C}$ de operadores de especialização é completa** para um conjunto de *pth's* $\mathcal{Q}$ se $\forall P \in \mathcal{Q}$ existe $P' \in L_0$, onde, $L_0$ é o conjunto dos *pth's* mais gerais de $\mathcal{Q}$, tal que, $P$ é obtido aplicando-se sucessivamente os operadores de $\mathcal{C}$ sobre $P'$.

De acordo com a definição anterior, uma classe de operadores de especialização completa percorre todo o espaço de busca dos *pth's* e gera todos os *pth's* possíveis a partir dos *pth's* mais gerais. Para evitar que um mesmo *pth's* seja obtido várias vezes, uma classe de operadores de especialização deve ser optimal.

**Definição 4.3.8 (Classe de operadores de especialização optimal)** Uma **classe $\mathcal{C}$ de operadores de especialização é optimal** para um conjunto de *pth's* $\mathcal{Q}$ se $\forall P, P_1, P_2 \in \mathcal{Q}$, em que $P$ pode ser obtido aplicando sucessivamente os operadores de $\mathcal{C}$ sobre $P_1$ e $P_2$, tem-se que $P_1 \sim P_2$.

### 4.3.1 Especialização

Considere o espaço de busca $\mathcal{P}ref(\mathcal{R})$. Para garantir uma especialização eficiente e eficaz para $\mathcal{P}ref(\mathcal{R})$, neste trabalho foram consideradas duas classes de operadores de especialização que especializam os *pth's* pertencentes a $\mathcal{P}ref(\mathcal{R})$ nível por nível, ou seja, a cada iteração os operadores de especialização são aplicados sobre os *pth's* pertencentes a $\mathcal{P}ref(\mathcal{R})$ aumentando em um seu nível de refinamento.

**Definição 4.3.9 (Classe $\mathcal{C}_E$ de operadores de extensão)** Seja $\mathcal{R}$ uma expressão regular, $\mathcal{P}ref(\mathcal{R})$ o espaço de busca dos *pth's* definido por $\mathcal{R}$ e um *pth* $P = (K, \mathcal{E}, \mathcal{T}) \in \mathcal{P}ref(\mathcal{R})$ com $\mathcal{E} = \{p_1, \ldots, p_n\}$. Uma **classe $\mathcal{C}_E$ de operadores de extensão** é constituída por operadores denotados por $\rho_E$ que são executados sobre $P$ da seguinte forma:

1. Se $v(P) = (n, c)$, com $c > 0$, então $\rho_E(P) = P$.
2. Se $v(P) = (n, 0)$ e existe um átomo $p_{n+1}(y_1, \ldots, y_l, \tau_{n+1})$, então $\rho_E(P) = P' = (K, \mathcal{E}', \mathcal{T}') \in \mathcal{P}ref(\mathcal{R})$, tal que:
   (a) $\mathcal{E}' = \mathcal{E} \cup \{p_{n+1}(y_1, \ldots, y_l, \tau_{n+1})\}$
   (b) $\mathcal{T}' = \mathcal{T} \cup \{r_1(\tau_1, \tau_{n+1}), \ldots, r_n(\tau_n, \tau_{n+1})\}$, onde $r_i \in \mathcal{P}_{temp}$ e $\mathcal{T}'$ é completo e consistente.

A definição anterior diz que os operadores de extensão não podem ser aplicados sobre *pth's* que contenham constantes (item 1 da definição). A especialização de um *pth* é realizada acrescentando um novo átomo no conjunto dos átomos de dados $\mathcal{E}$ (item 2(a) da definição). Alem disso, para garantir que continue completo e consistente, são inseridos novos átomos temporais no conjunto de átomos temporais $\mathcal{T}$ (item 2(b) da definição).

**Exemplo 4.3.4** Seja a expressão regular $\mathcal{R} = Hormônio(x, \$, \#)^* Fumo(x, \$, \#)^* Birads(x, \$, \#)$, o espaço de busca $\mathcal{P}ref(\mathcal{R})$ definido por $\mathcal{R}$, um *pth* $P = (Paciente(x), \{Hormônio(x, y, e_1)\}, \{\emptyset\}) \in \mathcal{P}ref(\mathcal{R})$ e os *pth's* $P_1$, $P_2$, $P_3$ e $P_4$.

$P_1 = (Paciente(x), \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2)\}, \{before(e_1, e_2)\})$

$P_2 = (Paciente(x), \{Hormônio(x, y, e_1), Fumo(x, z, f_2)\}, \{meets(e_1, f_2)\})$

$P_3 = (Paciente(x), \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, z, f_3)\},$
$\{before(e_1, e_2), before(e_1, f_3), finishes(e_2, f_3)\})$

$P_4 = (Paciente(x), \{Hormônio(x, y, e_1), Fumo(x, z, f_2), Birads(x, h, s_3)\},$
$\{during(e_1, f_2), before(e_1, s_3)\})$

Os *pth's* $P_1$ e $P_2$ pertencem a $\rho_E(P)$. O *pth* $P_3$ pertence a $\rho_E(P_1)$. O *pth* $P_4$ não pertence a $\rho_E(P_2)$, pois, $P_4$ não é completo, ou seja, não existe nenhuma relação implícita ou explicita entre as variáveis temporais $f_2$ e $s_3$.

**Definição 4.3.10 (Classe $\mathcal{C}_I$ de operadores de instanciação)** Seja $\mathcal{R}$ uma expressão regular, $\mathcal{P}ref(\mathcal{R})$ o espaço de busca dos *pth's* definido por $\mathcal{R}$ e um *pth* $P = (K(x_1, \ldots, x_m), \mathcal{E}, \mathcal{T}) \in \mathcal{P}ref(\mathcal{R})$, com $\mathcal{E} = \{p_1(y_1^1, \ldots, y_l^1, \tau_1), \ldots, p_n(y_1^n, \ldots, y_l^n, \tau_n)\}$. Uma **classe $\mathcal{C}_I$ de operadores de instanciação** é constituída por operadores denotados por $\rho_I$, os quais quando executados sobre $P$ produzem $P' = (K, \mathcal{E}', \mathcal{T})$ se existe uma substituição $\theta = \{y_j^i \rightarrow a\}$, onde, $a \in \mathcal{C}onst$ e $y_j^i \neq x_q$, de acordo com as seguintes condições:

1. O modo $w_i = p_i(u_1^i, \ldots, u_l^i, \#)$ que gerou o átomo $p_i(y_1^i, \ldots, y_l^i, \tau_i)$ deve possuir um símbolo na posição $u_j^i$;

2. Não pode existir nenhum $k > j$, tal que, $u_k^i =$ e $y_k^i \in \mathcal{C}onst$;

3. Não pode existir nenhum $h > i$ e $k \in \{1, \ldots, l\}$, tal que, $w_h = p_h(u_1^h, \ldots, u_l^h, \#)$, $u_k^h = \$$ e $y_k^h \in \mathcal{C}onst$.

A definição anterior diz que os operadores de instanciação só podem ser aplicados a *pth's* se o modo que gerou o átomo que contém a variável que vai ser instanciada possui o símbolo \$ na mesma posição da variável (item 1 da definição), e não pode existir nenhuma constante a direita da variável que vai ser instanciada no mesmo átomo (item 2 da definição) ou nos próximos átomos (item 3 da definição).

**Exemplo 4.3.5** Considere a expressão regular $\mathcal{R} = Hormônio(x,\$,\#)^* Fumo(x,\$,\#)^* Birads(x,\$,\#)$ e os *pth's* $P_1, P_2, P_3$ e $P_4$.

$P_1 = (Paciente(x),\ \{Hormônio(x,y,e_1), Fumo(x,z,f_2)\},\ \{during(e_1,f_2)\})$

$P_2 = (Paciente(x),\ \{Hormônio(x,y,e_1), Fumo(x,10,f_2)\},\ \{during(e_1,f_2)\})$

$P_3 = (Paciente(x),\ \{Hormônio(x,estradiol,e_1), Fumo(x,z,f_2)\},\ \{during(e_1,f_2)\})$

$P_4 = (Paciente(x),\ \{Hormônio(x,estradiol,e_1), Fumo(x,10,f_2)\},\ \{during(e_1,f_2)\})$

Os *pth's* $P_2$ e $P_3$ pertencem a $\rho_I(P_1)$, $P_4 \notin \rho_I(P_1)$, $P_4 \notin \rho_I(P_2)$, $P_4 \in \rho_I(P_3)$.

Observe que, se os operadores da classe de operadores de extensão pudessem ser aplicados sobre *pth's* com constantes, violando o item 1 da definição 4.3.9, e se os operadores da classe de operadores de instanciação pudessem instanciar variáveis a esquerda de uma variável que já foi instanciada nos *pth's*, violando o item 2 ou 3 da definição 4.3.10, poderiam ser gerados *pth's* a partir de *pth's* não equivalentes, ou seja, por caminhos diferentes, como mostrado na figura 4.1. A extensão através da seta pontilhada não é permitida, pois viola a optimalidade do método.

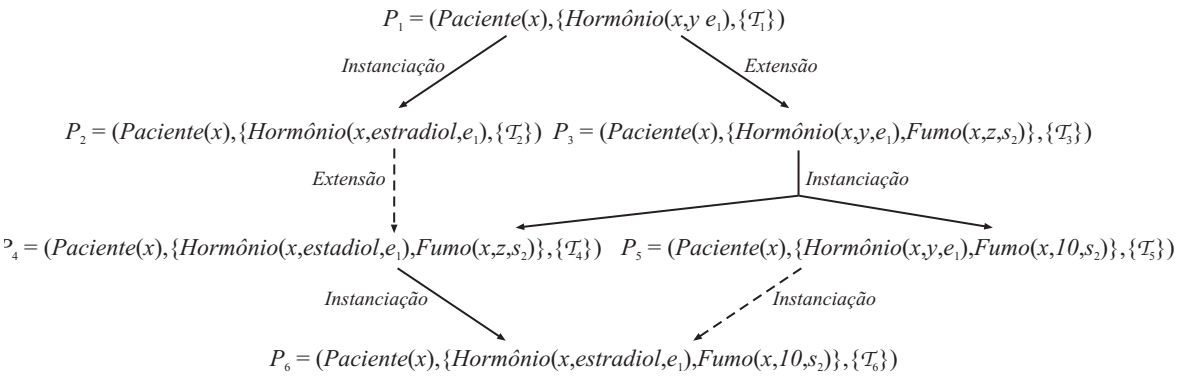



Figura 4.1: Exemplo de dois *pth's* $P_4$ e $P_6$ obtidos a partir de *pth's* não equivalentes

**Teorema 4.3.1** Seja $\mathcal{R}$ uma expressão regular, $\mathcal{P}ref(\mathcal{R})$ o espaço de busca dos *pth's* definido por $\mathcal{R}$, a classe de operadores de extensão $\mathcal{C}_E$ e a classe de operadores de instanciação $\mathcal{C}_I$. Se existe uma classe $\mathcal{C}$ de operadores de especialização definidos por $\rho$, tal que, $\mathcal{C} = \mathcal{C}_E \cup \mathcal{C}_I$, então $\mathcal{C}$ é completa e optimal para $\mathcal{P}ref(\mathcal{R})$.

**Prova:**

Para provar que $\mathcal{C}$ é completa para $\mathcal{P}ref(\mathcal{R})$ é necessário mostrar que para todo *pth* $P \in \mathcal{P}ref(\mathcal{R})$, onde, $P = (K, \mathcal{E}, \mathcal{T})$ com $\mathcal{E} = \{p_1(y_1^1, \ldots, y_l^1, \tau_1), \ldots, p_i(y_1^i, \ldots, y_j^i, \ldots, y_l^i, \tau_i), \ldots, p_n(y_1^n, \ldots, y_l^n, \tau_n)\}$, existe um $P' \in \mathcal{P}ref(\mathcal{R})$, tal que $P$ é obtido aplicando os operadores de $\mathcal{C}$ sobre $P'$, ou seja, $\rho(P') = P$. Para isso, dois casos devem ser tratados:

- **1º caso:** Se $v(P) = (n, 0)$.

  Seja $W_n = w_1 \ldots w_n$ a palavra associada a $P$.

  Seja $P' = (K, \mathcal{E}', \mathcal{T}')$ com $\mathcal{E}' = \{p_1(y_1^1, \ldots, y_l^1, \tau_1), \ldots, p_{n-1}(y_1^{n-1}, \ldots, y_l^{n-1}, \tau_{n-1}\}$ e $\mathcal{T}' = \{r(\tau_i, \tau_j) \in \mathcal{T} \mid i \geq 1, j \leq n-1\}$. A palavra associada a $P'$ é $W_{n-1} = w_1 \ldots w_{n-1}$, que é um prefixo de $W_n$. Por conseqüência disso, $W_{n-1}$ é um prefixo da linguagem especificada por $\mathcal{R}$ e $P' \in \mathcal{P}ref(\mathcal{R})$. Enfim, é verificado que $\rho(P') = P$.

- **2º caso:** Se $v(P) = (n, c)$, com $c > 0$.

  Seja $W_n$ a palavra associada a $P$ e $y_k^i$ a última entrada instanciada em $P$.

  Seja $P' = (K, \mathcal{E}', \mathcal{T})$ com $\mathcal{E}' = \{p_1(y_1^1, \ldots, y_l^1, \tau_1), \ldots, p_i(y_1^i, \ldots, y_j'^i, \ldots, y_l^i, \tau_i), \ldots, p_n(y_1^n, \ldots, y_l^n, \tau_n)\}$, onde $y_j'^i$ é uma nova variável, diferente das já existentes em $P'$, tal que o modo que gerou $p_i$ possui um símbolo $*$ na posição $j$. A palavra associada a $P'$ é a mesma de $P$, portanto $P' \in \mathcal{P}ref(\mathcal{R})$. Enfim, é verificado que $\rho(P') = P$.

Para provar que $\mathcal{C}$ é optimal para $\mathcal{P}ref(\mathcal{R})$ é necessário mostrar que para todo *pth* $P, P_1, P_2 \in \mathcal{P}ref(\mathcal{R})$, se $P$ é obtido aplicando os operadores de $\mathcal{C}$ sobre $P_1$ e $P_2$, então $P_1 \sim P_2$. Para isso, dois casos devem ser tratados.

- **1º caso:** Se $v(P) = (n, 0)$.

  Seja dois *pth's* $P_1 = (K, \mathcal{E}_1, \mathcal{T}_1)$ e $P_2 = (K, \mathcal{E}_2, \mathcal{T}_2)$ com:

- $\mathcal{E}_1 = \{p_1(t_1^1, \ldots, t_l^1, \tau_1), \ldots, p_{n-1}(t_1^{n-1}, \ldots, t_l^{n-1}, \tau_{n-1})\}$
- $\mathcal{E}_2 = \{p_1(z_1^1, \ldots, z_l^1, \tau_1), \ldots, p_{n-1}(z_1^{n-1}, \ldots, z_l^{n-1}, \tau_{n-1})\}$

Considere $\rho(P_1) = P$ e $\rho(P_2) = P$, portanto $P_1, P_2 \in \mathcal{P}ref(\mathcal{R})$. Como nenhum átomo de dados de $P$ contém constantes, então nenhum átomo de dados de $P_1$ e $P_2$ contém constantes, e portanto $P_1 \sim P_2$.

- **2º caso:** Se $v(P) = (n, c)$, com $c > 0$.

Seja dois *pth's* $P_1 = (K, \mathcal{E}_1, \mathcal{T}_1)$ e $P_2 = (K, \mathcal{E}_2, \mathcal{T}_2)$ com:

- $\mathcal{E}_1 = \{p_1(t_1^1, \ldots, t_l^1, \tau_1), \ldots, p_i(\ldots, t_j^i, \ldots, \tau_i), \ldots, p_n(t_1^n, \ldots, t_l^n, \tau_n)\}$
- $\mathcal{E}_2 = \{p_1(z_1^1, \ldots, z_l^1, \tau_1), \ldots, p_i(\ldots, z_j^i, \ldots, \tau_i), \ldots, p_n(z_1^n, \ldots, z_l^n, \tau_n)\}$
- $W_1 = w_1 = p_1(u_1^1, \ldots, u_l^1, \#) \ldots w_n = p_n(u_1^n, \ldots, u_l^n, \#)$
- $W_2 = w_1 = p_1(v_1^1, \ldots, v_l^1, \#) \ldots w_n = p_n(v_1^n, \ldots, v_l^n, \#)$

onde, $W_1$ e $W_2$ são as palavras associadas a $P_1$ e $P_2$ respectivamente.

- Considere $\rho(P_1) = P$, $\rho_I(P_2) = P$ e $P_1 \neq P_2$.

1 Como $\rho(P_1) = P$, $\exists i_1, j_1$ únicos, tal que, $u_{j_1}^{i_1} = \$$, $t_{j_1}^{i_1} \in \mathcal{V}ar$ e $y_{j_1}^{i_1} \in \mathcal{C}onst$. Além disso, $\forall i, j$, se $i \neq i_1$ ou $j \neq j_1$, $t_j^i = y_j^i$.

2 Como $\rho(P_2) = P$, $\exists i_2, j_2$ únicos, tal que, $v_{j_2}^{i_2} = \$$, $z_{j_2}^{i_2} \in \mathcal{V}ar$ e $y_{j_2}^{i_2} \in \mathcal{C}onst$. Além disso, $\forall i, j$, se $i \neq i_2$ ou $j \neq j_2$, $z_j^i = y_j^i$.

Agora dois casos devem ser tratados:

**1º caso:** Se $i_1 = i_2$ e $j_1 < j_2$.

Como $P_1 \neq P_2$, considere $t_{j_1}^{i_2} = z_{j_2}^{i_2} \in \mathcal{C}onst$. Como $\rho(P_1) = P$, então considere $t_{j_1}^{i_2} = y_{j_1}^{i_2} = y_{j_2}^{i_2} \in \mathcal{C}onst$. Para $\rho(P_2) = P$ sería necessário $z_{j_1}^{i_1} \in \mathcal{C}onst$. Isso viola o item 2 da definição 4.3.10, ou seja, uma variável $z$ em não pode ser instanciada se ela encontra-se em uma posição $j_1$ precedendo uma posição $j_2$ onde uma variável $z_{j_2}^{i_2}$ já foi instanciada.

**2º caso:** Se $i_1 < i_2$.

Como $P_1 \neq P_2$, então $u_{j_1}^{i_1} = v_{j_2}^{i_2} = \$$ e $t_{j_1}^{i_1}, z_{j_2}^{i_2} \in \mathcal{C}onst$. Assim, para que seja gerado um $P$, tal que, $\rho(P_2) = P$ e $\rho(P_1) = P$, a variável $z_{j_1}^{i_1}$ deveria ser instanciada, isso contraria o item 3 da definição 4.3.10, ou seja, uma variável não pode ser instanciada se ela se encontra em um átomo a um índice $i_1$ antecedendo um átomo $p_{i_2}(\ldots, z_{j_2}^{i_2}, \ldots, \tau_{i_2})$, onde, uma variável $z_{j_2}^{i_2}$ já foi instanciada.

## 4.4 Considerações Finais do Capítulo

Neste capítulo foram definidas expressões regulares que são restrições introduzidas no processo de mineração com o objetivo de reduzir o espaço de busca dos padrões. Além de reduzir o espaço de busca dos padrões este tipo de restrição permite que o usuário defina o tipo dos *pth's* que deseja minerar, assim, ele não é sobrecarregado com *pth's* que não o interessam.

Foram definidos também operadores de especialização que quando são aplicados sobre os *pth's* durante o processo de mineração geram um conjunto de *pth's* mais específicos a cada nível. Estes operadores satisfazem algumas propriedades especiais que garantem completude e optimilidade ao método, ou seja, garantem que sejam gerados todos os *pth's* possíveis a partir de um determinado *pth*, e que não sejam gerados *pth's* duplicados.

# Capítulo 5

# O Algoritmo MILPRIT*

Neste capítulo será apresentado o algoritmo MILPRIT* (**M**ining **I**nterval **L**ogic **P**atterns with **R**egular expressIons consTraints) desenvolvido para realizar a mineração de padrões temporais híbridos. O MILPRIT* encontra todos os *pth's* freqüentes em um banco de dados temporal $BD$ com relação a um suporte mínimo $\alpha$ e a uma expressão regular $\mathcal{R}$ especificada pelo usuário. Este algoritmo foi implementado na linguagem C++ sobre o sistema operacional Linux.

É importante ressaltar que o algoritmo MILPRIT* foi desenvolvido a partir do protótipo MILPRIT [16], que realiza mineração de padrões temporais intervalares, como descrito na seção 2.4.4. O protótipo MILPRIT foi implementado com as idéias básicas da mineração de padrões temporais intervalares e possui alguns problemas. O primeiro deles é o fato de não realizar gerenciamento de memória, o que limita sua utilização, uma vez que a cada iteração o número de padrões gerados aumenta consideravelmente, ocupando toda a memória principal disponível. Outro problema do protótipo MILPRIT está na fase de geração dos padrões, onde determinados padrões não são gerados ou são gerados padrões que não são interessantes para o usuário.

Para solucionar os problemas apresentados no parágrafo anterior foi desenvolvido um novo algoritmo, chamado MILPRIT*, que utiliza algumas técnicas para otimizar a ocupação da memória principal e a etapa de geração. O algoritmo MILPRIT* também estende a técnica de mineração de padrões temporais intervalares visando realizar a mineração de padrões temporais híbridos. As técnicas utilizadas para otimizar a ocupação da

memória e a fase de geração serão apresentadas na seção 5.1.4.

A partir deste ponto serão utilizadas as notações $C_k$ e $F_k$ para denotar o conjunto de *pth's candidatos* e *freqüentes*, respectivamente.

## 5.1 O Algoritmo

O MILPRIT* é um algoritmo de mineração de *pth's* baseado na técnica *Apriori*, ou seja, a cada iteração $k$ executa as etapas de *geração dos candidatos*, *poda dos candidatos* e *avaliação*, produzindo um conjunto $F_k$ constituído de *pth's* freqüentes imediatamente mais específicos que os *pth's* contidos em um conjunto $F_{k-1}$, obtido na iteração anterior.

O MILPRIT* utiliza além da restrição baseada no suporte, restrições especificadas através de expressões regulares $\mathcal{R}$. Seguindo a idéia introduzida primeiramente nos algoritmos da família SPIRIT [12] e posteriormente nos algoritmos da família SPIRIT-Log [14], o MILPRIT* incorpora uma restrição $\mathcal{R}'$, que é um "relaxamento" de $\mathcal{R}$, ou seja, $\mathcal{R}'$ é uma restrição mais fraca que $\mathcal{R}$. O motivo do uso de $\mathcal{R}'$ ao invés de $\mathcal{R}$ será explicado nos próximos parágrafos.

Considere momentaneamente que a restrição original $\mathcal{R}$ seja incorporada pelo algoritmo MILPRIT*. Em uma iteração $k$, na fase de geração é gerado um conjunto $C_k$ contendo somente *pth's* candidatos que satisfazem $\mathcal{R}$. Posteriormente, na fase de poda, de acordo com a propriedade *antimonotonica*, são podados de $C_k$ todos os *pth's* que possuem pelo menos um sub-padrão (um *pth* mais geral) não freqüente. Entretando, pelo fato da expressão regular $\mathcal{R}$ não ser *antimonotônica*, não podem ser podados *pth's* que possuem pelo menos um sub-padrão que não satisfaz $\mathcal{R}$, pois fazendo isso, poderiam ser podados *pth's* interessantes.

Para que um *pth* $P \in C_k$ seja podado, deve existir algum *pth* $P' \succ P$ ($P'$ mais geral que $P$) que não seja freqüente. Sabe-se que um *pth* $P' \notin F = F_1 \cup \ldots \cup F_{k-1}$ se e somente se: (1) $P'$ não é freqüente ou (2) $P'$ não satisfaz a restrição $\mathcal{R}$. Assim, para ter certeza que $P$ não é freqüente (pode ser podado), basta $P'$ satisfazer $\mathcal{R}$ e $P' \notin F$. Com isso, quanto *maior* for a seletividade de $\mathcal{R}$ *menor* será o número de padrões gerados e *menor* será o número de padrões podados. A diminuição do número de padrões podados não é

interessante, pois um dos principais objetivos desse tipo de restrição é diminuir o número de *pth's* que serão levados para a etapa de avaliação. A relação entre a seletividade de $\mathcal{R}$ e a quantidade de *pth's* gerados e podados é mostrada na figura 5.1.

| Seletividade de $\mathcal{R}$ | *PTH's* Gerados | *PTH's* Gerados |
|---|---|---|
| $\Uparrow$ | $\Downarrow$ | $\Downarrow$ |

Figura 5.1: Relação entre a restrição $\mathcal{R}$ e as etapas de geração e poda

Visando um equilíbrio entre a quantidade de *pth's* gerados e a quantidade de *pth's* podados, ou seja, visando diminuir a quantidade de *pth's* gerados e aumentar a quantidade de *pth's* podados, o MILPRIT* considera uma restrição $\mathcal{R}'$, que é um *relaxamento* da restrição $\mathcal{R}$. Desta forma são gerados somente *pth's* que satisfazem $\mathcal{R}'$, e portanto, é considerado o *prefixo associado a restrição* $\mathcal{R}$. Assim, somente *pth's prefixos válidos* ou simplesmente *p-válidos* são gerados.

**Definição 5.1.1** Seja $A_{\mathcal{R}}$ um autômato associado a uma expressão regular $\mathcal{R}$ e $P = \{K, \mathcal{E}, \mathcal{T}\}$ um *pth*. $P$ é *p-válido* com relação $A_{\mathcal{R}}$ se existe um caminho em $A_{\mathcal{R}}$ que começa no estado inicial e produz a seqüência de átomos de dados $\mathcal{E}$, não terminando necessariamente em um estado final de $A_{\mathcal{R}}$.

Resumidamente, o algoritmo MILPRIT* pode ser descrito como segue. Para cada iteração $k$, o conjunto $C_k$ de *pth's p-válidos* é obtido pela especialização dos *pth's p-válidos* e freqüentes de $F_{k-1}$ obtido na iteração anterior. Após a etapa de geração, é executada a etapa de poda, onde são podados de $C_k$ todos os *pth's* $P$ que possuem pelo menos um *pth* $P' \succ P$, tal que $P'$ é *p-válido* ($P'$ satisfaz $\mathcal{R}'$) e $P' \notin F = F_1 \cup \ldots \cup F_{k-1}$ ($P'$ não é freqüente). Na etapa de avaliação, o banco de dados é percorrido para calcular o suporte de todos os *pth's* que restaram em $C_k$ após a poda. Enfim são inseridos em $F_k$ todos os *pth's* com suporte maior ou igual a um suporte mínimo estipulado pelo usuário.

Uma visão geral da estrutura do algoritmo MILPRIT* é dada na figura 5.2. O símbolo $\Sigma_1$ representa um conjunto de átomos de dados $\{p_1(y_1^1, \ldots, y_l^1, \tau_1), \ldots, p_n(y_1^n, \ldots, y_l^n, \tau_n)\}$ obtido a partir do alfabeto de modos $\Sigma = \{p_1(u_1^1, \ldots, u_l^1, \#), \ldots, p_n(u_1^n, \ldots, u_l^n, \#)\}$. O parâmetro de entrada representado pelo símbolo $\delta$ é a granularidade escolhida pelo usuário

para ser considerada durante o processo de mineração. Detalhes sobre a granularidade serão apresentados na fase de avaliação na seção 5.1.3.

**Algoritmo** MILPRIT*($BD$-banco de dados, $\alpha$-suporte mínimo, $\mathcal{R}$-expressão regular, $\delta$-granularidade)

1. $k := 1$;
2. $F := F_1 = \{P \in \Sigma_1 : P$ é freqüente e $P \in \mathcal{P}ref(R)\}$;
3. $k := 2$;
4. **Repeat**
   5. $i := 0$;
   6. **Repeat**
      7. $i := i + 1$;
      8. $C :=$ Gen($F_{k-1}[i]$,$\mathcal{R}$); (***Etapa de Geração***)
      9. $P := \{P \in C : \exists P' \in \mathcal{P}ref(R)$, tal que, $P' \notin F$ e $P'$ é mais geral que $P\}$;
      10. $C_k := C_k \cup (C - P)$; (***Etapa de Poda***)
   11. **Until** $i = F_{k-1}.size$; ($F_{k-1}.size$ retorna a quantidade de padrões em $F_{k-1}$)
   12. $F_k := \{P \in C_k :$ sup(P) $\geq \alpha\}$; (***Etapa de Avaliação***)
   13. $F := F \cup F_k$;
   14. $k := k + 1$;
15. **Until** $F_{k-1} = \emptyset$;

Figura 5.2: Algoritmo MILPRIT*

Nas próximas seções serão detalhadas as etapas de geração dos candidatos, poda dos candidatos e avaliação do algoritmo MILPRIT*.

### 5.1.1 Etapa de Geração dos Candidatos

A função Gen($F_{k-1}[i]$,$\mathcal{R}$) gera *pth's* com nível de refinamento $k$, *especializando* cada *pth* $i$ de $F_{k-1}$, cujo nível de refinamento é $k-1$, de tal modo que os *pth's* gerados sejam *p-válidos*. A especialização de um *pth* é realizada a cada iteração por funções que aplicam os operadores de especialização definidos no capítulo anterior.

Cada *pth* $P$ de $F_{k-1}$ é especializado primeiramente pela função $Instantiation(P, \mathcal{R})$, apresentada na figura 5.4, que aplica o operador de instanciação sobre $P$ gerando um conjunto $PI$ de *pth's* instanciados. Posteriormente, se $v(P) = (n, 0)$, ou seja, se $P$ não contêm constantes, então a função $Extension(P, \mathcal{R})$, apresentada na figura 5.7, aplica o operador de extensão sobre $P$ gerando um conjunto $PE$ de *pth's* estendidos. Portanto, a especialização de cada *pth* $P$ de $F_{k-1}$ resulta em dois conjuntos, $PI$ e $PE$, de *pth's*

especializados que são inseridos em um conjunto $C_k$ de *pth's* candidatos. No fim da etapa de geração, o conjunto $C_k$ é formado por todos os *pth's* obtidos pela especialização de todos *pth's* de $F_{k-1}$.

Uma importante função exigida tanto pela função $Instantiation(\ )$ quanto pela função $Extension(\ )$ é a função $Match(\ )$, que encontra a *palavra* associada a um determinado *pth*.

**A Função** $Match(\ )$**.** Dado uma expressão regular $\mathcal{R}$ e um *pth* $P = (K, \mathcal{E}, \mathcal{T})$ com $\mathcal{E} = \{p_1(y_1^1, \ldots, y_l^1, \tau_1), \ldots, p_n(y_1^n, \ldots, y_l^n, \tau_n\}$, a função $Match(P, \mathcal{R})$, apresentada na figura 5.3, retorna uma *string* $W = w_1 \ldots w_n$ que representa a palavra associada a $P$.

**Função** $Match(P, \mathcal{R})$

1. W = $\emptyset$;
2. **For** $i = 1$ to $n$ **do begin**
3. **For each** $w \in Alfabeto(\mathcal{R})$ tal que $w = p_j(u_1^j, \ldots, u_l^j, \#)$ **do begin**
4. ok = true;
5. **If** $(p_i = p_j)$ **then**
6. **For** $k = 1$ to $l$ **do**
7. **If** $((u_k^j \in \mathcal{V}ar)$ e $(y_k^i \neq u_k^j))$ **then** ok = false;
8. **If** (ok) **then** $w_i = w$;
9. **End for**;
10. **If** (ok) **then** $W = W + w_i$;
11. **Else return** 0;
12. **End for**;
13. **Return** $W$;

Figura 5.3: Função $Match(P, \mathcal{R})$

Os passos realizados pela função $Match(P, \mathcal{R})$ para a construção de uma palavra $W$ associada a um *pth* $P$ são dados por:

1. Primeiramente inicia a palavra $W$ como vazia, ou seja, $W = \emptyset$ (linha 1).
2. Da linha 2 até a linha 12, encontra o modo que gerou a cada átomo de dados $p_i(y_1^i, \ldots, y_l^i, \tau_i)$ do *pth* da seguinte forma:
   - Percorre o alfabeto $Alfabeto(\mathcal{R})$, constituído de modos da expressão regular $\mathcal{R}$, para encontrar o modo $w$ que gerou o átomo de dados $p_i(y_1^i, \ldots, y_l^i, \tau_i)$.

– Seja $w = p_j(u_1^j, \ldots, u_l^j, \#)$. Para todo $k \in \{1, \ldots, l\}$, se $u_k^j \in \mathcal{V}ar$ e $u_k^j = y_k^i$, então o modo $w$ é gerou o átomo de dados $p_i(y_1^i, \ldots, y_l^i, \tau_i)$.

- Insere o modo $w$ no fim de $W$, ou seja, $W = W + w$ (linha 10).

3. Retorna A palavra $W$ associada ao padrão $P$ (linha 13).

**Exemplo 5.1.1** Considere a expressão regular $\mathcal{R} = Hormônio(x, \$, \#)^* Fumo(x, \$, \#)$ $Birads(x, \$, \#)$, o *pth* $P = (Paciente(x)$, $\{Hormônio(x, y_1, e_1)$, $Hormônio(x, y_2, e_2)$, $Fumo(x, z, f_3)\}$, $\{before(e_1, e_2)$,$meets(e_1, f_3)$,$during(e_2, f_3)\}$) e o alfabeto $Alfabeto(\mathcal{R}) = \{w_1 = Hormônio(x, \$, \#)$, $w_2 = Fumo(x, \$, \#)$, $w_3 = Birads(x, \$, \#)\}$.

1. Inicialmente a palavra associada a $P$ é vazia, $W = \emptyset$.

2. Os modos que geraram os átomos de dados de $\mathcal{E}$ (da linha 2 até a 12) são:

   - Para $i = 1$, $Hormônio(x, \$, \#)$ é o modo que gerou o átomo de dados $Hormônio(x, y_1, e_1)$, portanto, $W = W + w_1 = w_1$.
   - Para $i = 2$, $Hormônio(x, \$, \#)$ é o modo que gerou o átomo de dados $Hormônio(x, y_2, e_2)$, portanto, $W = W + w_1 = w_1 w_1$.
   - Para $i = 3$, $Fumo(x, \$, \#)$ é o modo que gerou o átomo de dados $Fumo(x, z, f_3)$, portanto, $W = W + w_2 = w_1 w_1 w_2$.

3. A palavra associada ao *pth* $P$ é $W = w_1 w_1 w_2$.

**A Função de Especialização por Instanciação**

A especialização por instanciação de um *pth* é realizada pela função $Instantiation(\ )$, apresentado na figura 5.4. Esta função tem como entrada um *pth* $P$ e uma expressão regular $\mathcal{R}$ e retorna um conjunto de *pth's* $PI = \rho_I(P)$.

Quando um *pth* $P = (K, \mathcal{E}, \mathcal{T})$ com $\mathcal{E} = \{p_1(y_1^1, \ldots, y_l^1, \tau_1), \ldots, p_n(y_1^n, \ldots, y_l^n, \tau_n\}$ é submetido a função $Instantiation(P, \mathcal{R})$, são gerados *pth's* $P' = (K, \mathcal{E}', \mathcal{T})$ pela substituição de uma variável $y_k^i$ de $\mathcal{E}$ por uma constante, desde que:

- O átomo de dados $p_i(y_1^i, \ldots, y_l^i, \tau_i)$ não contenha constante nas entradas $y_{k'}^i$ para todo $k' \in \{k, \ldots, l\}$; ou

- Os átomos de dados $p_j(y_1^j, \ldots, y_l^j, \tau_j)$ com $j > i$ não contenham constantes nas entradas $y_k^j$ para todo $k \in \{1, \ldots, l\}$; e

- A entrada $u_k^i$ do modo que gerou o átomo $p_i(y_1^i, \ldots, y_l^i, \tau_i)$ seja \$.

**Função** $Instantiation(P, \mathcal{R})$

1. $PI = \emptyset$;
2. $W = Match(P, \mathcal{R})$; $(W = w_1 \ldots w_n)$;
3. $i_0 = 0$;
4. $j_0 = 0$;
5. **For** $i = 1$ to $n$ **do begin**
6. $\quad w_i = p_i(u_1^i, \ldots, u_l^i, \#)$;
7. $\quad$ **For** $j = 1$ to $l$ **do**
8. $\quad\quad$ **If** $((u_j^i = \$)$ e $(y_j^i \in \mathcal{C}onst))$ **then** $i_0 = i$ e $j_0 = j$;
9. **End for**;
10. $I = \emptyset$;
11. **For** $i = i_0$ to $n$ **do begin**
12. $\quad w_i = p_i(u_1^i, \ldots, u_l^i, \#)$;
13. $\quad$ **For** $j = j_0 + 1$ to $l$ **do**
14. $\quad\quad$ **If** $((u_j^i = \$)$ **then** $I = I \cup \{y_j^i\}$; ($I$ *é conjunto das variáveis que podem ser instanciadas*)
15. $\quad j_0 = 0$;
16. **End for**;
17. **For each** $y_k^i \in I$ **do begin**
18. $\quad$ **For each** $a \in \mathcal{C}onst$ **do begin**
19. $\quad\quad P' = P$;
20. $\quad\quad \theta = \{y_k^i \mapsto a\}$; (*uma substituição*)
21. $\quad\quad P' = (K, \mathcal{E}', \mathcal{T})$; (a substituição $\theta$ *é aplicada em* $\mathcal{E}$ *gerando* $\mathcal{E}'$)
22. $\quad\quad PI = PI \cup \{P'\}$;
23. $\quad$ **End for**;
24. **End for**;
25. **Return** $PI$;

Figura 5.4: Função $Instantiation(P, \mathcal{R})$

A função $Instantiation(P, \mathcal{R})$ especializa um *pth* $P$ da seguinte forma:

1. Primeiramente, encontra a palavra $W$ associada ao *pth* $P$ com a ajuda da função $Match(P, \mathcal{R})$.

2. Encontra os índices $i_0$ e $j_0$ que representam, respectivamente, o átomo de dados $p_i$ e a entrada $y_j^i$, tal que, $y_j^i$ foi a última entrada instanciada em $P$.

3. Insere em um conjunto $I$ todas as variáveis de dados que podem ser instanciadas em $P$, ou seja, $I = y_k^i$ para $(i = i_0$ e $k > j_0)$ ou $(i > i_0$ e $k > 0)$ e $u_k^i = \$$ (da linha 10 até a 16).

4. Para cada variável $y_k^i \in I$ (da linha 17 até a 24):

   - Gera *pth's* $P' = P$ substituindo a variável $y_k^i$ de $P'$ pelas constantes pertencentes a $\mathcal{C}onst$. Todo $P'$ gerado é inserido no conjunto $PI$.

5. Retorna o conjunto $PI$ de *pth's* obtidos pela instanciação de $P$.

**Exemplo 5.1.2** Considere a expressão regular $\mathcal{R} = Hormônio(x, \$, \#)^* Fumo(x, \$, \#)$ $Birads(x, \$, \#)$ e o seguinte conjunto $F_3$ constituído por um único *pth* $P$.

$$F_3 = \{\ P = (Paciente(x), \{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),\ Fumo(x, z, f_3)\},$$
$$\{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})\ \}$$

1. A palavra associada a $P$ é $W = w_1 w_1 w_2 = Hormônio(x, \$, \#) Hormônio(x, \$, \#)$ $Fumo(x, \$, \#)$.

2. Como nenhuma variável de $P$ foi instanciada, o conjunto $I$ é constituído por todas elas, ou seja, $I = \{y_1, y_2, z\}$. Se existisse em $F_3$ um outro *pth* $P' = (Paciente(x),$ $\{Hormônio(x, y, e_1), Hormônio(x, estradiol, e_2), \{before(e_1, e_2)\})$, onde a última entrada instanciada foi a segunda entrada $(j_0 = 2)$ do segundo átomo de dados $(i_0 = 2)$, não existiria nenhuma variável para ser instanciada, e portanto $I = \{\emptyset\}$. Assim, $P'$ não seria instanciado.

3. Considerando os domínios $dom(2, Hormônio) = \{estradiol\}$ e $dom(2, Fumo) = \{10, 20\}$, a substituição de cada variável de $I$ por todas as constantes dos domínios resulta nos seguintes *pth's*:

$$P_1' = (Paciente(x), \{Hormônio(x, estradiol, e_1), Hormônio(x, y, e_2), Fumo(x, z, f_3)\},$$
$$\{before(e_1, e_2), meets(e_1, f_3), during(e_2, f_3)\})$$

$$P'_2 = (Paciente(x), \{Hormônio(x, y, e_1), Hormônio(x, estradiol, e_2), Fumo(x, z, f_3)\},$$
$$\{before(e_1, e_2), meets(e_1, f_3), during(e_2, f_3)\})$$

$$P'_3 = (Paciente(x), \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, 10, f_3)\},$$
$$\{before(e_1, e_2), meets(e_1, f_3), during(e_2, f_3)\})$$

$$P'_4 = (Paciente(x), \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, 20, f_3)\},$$
$$\{before(e_1, e_2), meets(e_1, f_3), during(e_2, f_3)\})$$

4. $P'_1, P'_2, P'_3$ e $P'_4$ são inseridos no conjunto $PI = \rho_I(P)$.

### O Algoritmo de Especialização por Extensão

A especialização por extensão de um *pth* é realizada pela função $Extesion(\ )$, apresentada na figura 5.7. Esta função tem como entrada um *pth* $P$ e uma expressão regular $\mathcal{R}$ e retorna um conjunto de *pth's* $PE = \rho_E(P)$.

Quando um *pth* $P = (K, \mathcal{E}, \mathcal{T})$ com $\mathcal{E} = \{p_1(y_1^1, \ldots, y_l^1, \tau_1), \ldots, p_n(y_1^n, \ldots, y_l^n, \tau_n\}$ é submetido a função $Extension(P, \mathcal{R})$, é gerado um conjunto de *pth's* $P' = (K, \mathcal{E}', \mathcal{T}')$ pela inserção de um novo átomo de dados $p_{n+1}$ no fim da seqüência $\mathcal{E}$, tendo como conseqüência a inserção de uma nova variável temporal $\tau_{n+1}$ no conjunto de átomos temporais $\mathcal{T}$.

Além da função $Match(\ )$, a função $Extension(\ )$ utiliza uma outra importante função, chamada $Textension(\ )$, que especializa (insere a nova variável temporal $\tau_{n+1}$) de forma completa e consistente o conjunto de átomos temporais de um *pth*.

Como todo conjunto de átomos temporais $\mathcal{T}$ gerado é completo e consistente, então para todo $i, j \in \{1, \ldots, n\}$, onde $n$ é o número de variáveis temporais em $\mathcal{T}$, existe $r \in \mathcal{P}_{temp}$, tal que, $r(\tau_i, \tau_j)$ ou $r(\tau_j, \tau_i)$. Com isso, é possível representar um conjunto $\mathcal{T}$ por uma matriz $n \times n$. Na figura 5.5, $T_{i,j}$ denota o predicado temporal que relaciona as variáveis temporais $\tau_i$ e $\tau_j$. $T_{i,j}$ pode assumir os valores: $before$ ou $meets$ ou $overlaps$ ou $starts$ ou $during$ ou $finishes$, dependendo do tipo de cada variável envolvida, como detalhado na adaptação da estrutura temporal da LTI, apresentada na seção 3.1.1.

Embora todo predicado temporal $T_{j,i}$ possa ser deduzido de $T_{i,j}$, neste trabalho apenas os predicados $T_{i,j}$ com $i < j$ são considerados. A partir deste ponto $T_{i,j} = NULL$ denotará um conjunto de átomos temporais $\mathcal{T}$ que não é completo, ou seja, que tem a relação entre duas variáveis temporais $\tau_i$ e $\tau_j$ indeterminada.

| $\mathcal{T}$ | ... | $\tau_j$ | ... |
|---|---|---|---|
| $\vdots$ | | | |
| $\tau_i$ | | $T_{i,j}$ | |
| | | | |

Figura 5.5: Estrutura da matriz que representa um conjunto $\mathcal{T}$

**Exemplo 5.1.3** Considere o conjunto de átomos temporais $\mathcal{T} = \{before(\tau_1, \tau_2)$, $overlaps(\tau_1, \tau_3)$, $during(\tau_2, \tau_3)\}$. Este conjunto é representado pela seguinte matriz:

| $\mathcal{T}$ | $\tau_1$ | $\tau_2$ | $\tau_3$ |
|---|---|---|---|
| $\tau_1$ | | $before$ | $overlaps$ |
| $\tau_2$ | | | $during$ |
| $\tau_3$ | | | |

**A Função** $Textension(\ )$. Dado um conjunto de átomos temporais $\mathcal{T}$, referente a um *pth* $P = (K, \mathcal{E}, \mathcal{T})$ com $\mathcal{E} = \{p_1, \ldots, p_n\}$ e o modo $w_{n+1}$ que irá gerar o novo átomo de dados que irá ser inserido em $P$, a função $Textension(\mathcal{T}, w_{n+1})$, apresentada na figura 5.6, retorna um conjunto $ST$ contendo todos os possíveis conjuntos de átomos temporais $\mathcal{T}'$ consistentes e completos que contêm $\mathcal{T}$ e uma nova variável temporal $\tau_{n+1}$ do tipo (intervalar ou pontual) definido pelo modo $w_{n+1}$.

A construção dos conjuntos $\mathcal{T}'$ a partir de um conjunto $\mathcal{T}$ com $n$ variáveis temporais, é realizada de acordo com os seguintes passos:

1. Primeiramente, para cada $r \in \mathcal{P}_{temp}$ que pode relacionar, de acordo com a adaptação da estrutura temporal da LTI, a variável temporal $\tau_n$ e a nova variável temporal $\tau_{n+1}$ (da linha 2 até a 7), é criada uma matriz $\mathcal{T}' = \mathcal{T}$. São definidas como indeterminadas ($NULL$) todas as entradas $\mathcal{T}'_{i,n+1}$ com $i \in \{1, \ldots, n-1\}$ e a entrada $\mathcal{T}'_{n,n+1}$ é inicializada com $r$. Segundo a adaptação da estrutura temporal da LTI, o valor de $r$ é definido da seguinte maneira:

   - Se $\tau_n$ for do tipo intervalo e $\tau_{n+1}$ também for do tipo intervalo (caso 1 da estrutura temporal), então $r = before$ ou $r = meets$ ou $r = overlaps$ ou $r = starts$ ou $r = during$ ou $r = finishes$.

**Função** Textension($\mathcal{T}$,$w_{n+1}$)

1. $ST := \emptyset$;
2. **For each** $r \in \mathcal{P}_{temp}$ **do begin**
3. **If** ($r$ pode relacionar $\tau_n$ com $\tau_{n+1}$ de acordo com $\mathcal{T}_{temp}$) **then**;
4. $\mathcal{T}' := \mathcal{T}$;
5. $\mathcal{T}'_{n,n+1} := r$ e $\forall i \in \{1, \ldots, n-1\}$ $\mathcal{T}'_{i,n+1} := NULL$;
6. $ST := ST \cup \{\mathcal{T}'\}$;
7. **End for**
8. **For** $i = n - 1$ **down to** $i = 1$ **do begin**
9. **For each** $\mathcal{T}' \in ST$ **do begin**
10. **For each** $r \in \mathcal{P}_{temp}$ **do begin**
11. poss := true;
12. **If** $n > 2$ **then**
13. **For** $j = i + 1$ **to** $j = n$ **do begin**
14. **If** ($\mathcal{T}'_{i,j}(\tau_i, \tau_j) \wedge \mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1}) \wedge r(\tau_i, \tau_{n+1})$ é válido) **then**;
15. poss := poss $\wedge$ true;
16. **Else** poss := false;
17. **End for**;
18. **If** ($poss$) **then begin**
19. $\mathcal{T}'' := \mathcal{T}'$;
20. $\mathcal{T}''_{i,n+1} := r$;
21. $ST' := ST' \cup \{\mathcal{T}''\}$;
22. **End if**;
23. **End for**;
24. **End for**;
25. $ST := ST'$;
26. $ST' := \emptyset$;
27. **End for**;
28. **Return** $ST$;

Figura 5.6: Função $Textension(\mathcal{T}, w_{n+1})$

- Se $\tau_n$ for do tipo ponto e $\tau_{n+1}$ também for do tipo ponto (caso 2 da estrutura temporal), então $r = before$.
- Se $\tau_n$ for do tipo ponto e $\tau_{n+1}$ for do tipo intervalo (caso 3 da estrutura temporal), então $r = before$ ou $r = starts$ ou $r = during$ ou $r = finishes$.
- Se $\tau_n$ for do tipo intervalo e $\tau_{n+1}$ for do tipo ponto (caso 3 da estrutura temporal), então $r = before$.

2. Todos os conjuntos $\mathcal{T}'$ criados no passo anterior são inseridos em um conjunto $ST$.

3. Os passos da linha 8 até a 27 são aplicados sobre todos os conjuntos $T' \in ST$ até que para todo $i \in \{1, \ldots, n-1\}$, $\mathcal{T}'_{i,n+1} \neq NULL$, ou seja, até que toda entrada seja fixada.

   - Todo conjunto $\mathcal{T}'$ com no máximo duas variáveis temporais é consistente. Entretanto, antes de inserir um novo predicado $r$ em um conjunto $\mathcal{T}'$ com mais de duas variáveis temporais (linha 12), é preciso testar se a consistência de $\mathcal{T}'$ é conservada.

   - Na linha 14, para cada $r \in \mathcal{P}_{temp}$ que torna a fórmula $\mathcal{T}'_{i,j}(\tau_i, \tau_j) \wedge \mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1}) \wedge r(\tau_i, \tau_{n+1})$ válida é criado um conjunto $\mathcal{T}'' = \mathcal{T}'$ e $r$ é inserido em $\mathcal{T}''_{i,n+1}$ (linha 20). Caso contrário, $r$ torna $\mathcal{T}'$ inconsistente.

   - Para verificar se a fórmula da linha 14 é válida, foram definidas todas as possibilidades que conservam a consistência de $\mathcal{T}'$ incluindo o novo predicado temporal $r$. Estas possíbilidades estão anexadas no Apêndice A desta dissertação.

4. É retornado o conjunto $ST$ com todos os possíveis conjuntos de átomos temporais consistentes e completos obtidos a partir de $\mathcal{T}$ pela inclusão de uma nova variável temporal $\tau_{n+1}$.

**Exemplo 5.1.4** Considere a expressão regular $\mathcal{R} = Hormônio(x, \$, \#)^* Fumo(x, \$, \#)$ $Birads(x, \$, \#)$ e o *pth* $P = (Paciente(x), \{Hormônio(x, y_2, e_1), Hormônio(x, y_2, e_2), Fumo(x, z, f_3)\}, \{before(e_1, e_2), meets(e_1, f_3), during(e_2, f_3)\})$ do conjunto $F_3$ do exemplo anterior. De acordo com a expressão regular, $P$ pode ser estendido somente pela inclusão do átomo de dados $Birads(x, h, s_4)$. Portanto, a variável temporal $s_4$ é pontual, ou seja, o modo $w_{n+1}$ que gera o átomo $Birads(x, h, s_4)$ possui um termo temporal pontual.

O conjunto de átomos temporais $\mathcal{T}$ do *pth* $P$ é representado pela seguinte matriz:

| $\mathcal{T}$ | $e_1$ | $e_2$ | $f_3$ |
|---|---|---|---|
| $e_1$ | | $before$ | $meets$ |
| $e_2$ | | | $during$ |
| $f_3$ | | | |

1. Como a última variável temporal de $\mathcal{T}$, $f_3$, é intervalar e a variável temporal que irá ser inserida, $s_4$, é pontual, a extensão de $\mathcal{T}$ começa com a geração de um conjunto $\mathcal{T}'$, onde $\mathcal{T}' = \mathcal{T}$, tal que $\mathcal{T}'_{i,4} = NULL$ com $i \in \{1, \ldots, n-1\}$ e $\mathcal{T}'_{3,4} = r = before$ (da linha 2 até a 7). É gerado somente um conjunto $\mathcal{T}'$ porque de acordo com a adaptação da estrutura temporal da LTI, $before$ é único predicado temporal que relaciona uma variável temporal intervalar com uma variável temporal pontual, respectivamente. Assim, a matriz que representa o conjunto $\mathcal{T}'$ é a seguinte:

| $\mathcal{T}'$ | $e_1$ | $e_2$ | $f_3$ | $s_4$ |
|---|---|---|---|---|
| $e_1$ | | $before$ | $meets$ | $NULL$ |
| $e_2$ | | | $during$ | $NULL$ |
| $f_3$ | | | | $before$ |
| $s_4$ | | | | |

2. O conjunto $\mathcal{T}'$ gerado anteriormente é inserido em $ST$.

3. Da linha 8 até a 27:

   - Para $i = 2$ :

     - Para $\mathcal{T}'$, $r = before$ é o único predicado temporal que pode relacionar as variáveis temporais $e_2$ e $s_4$ conservando a consistência de $\mathcal{T}'$. Assim, é criado um conjunto $\mathcal{T}''$ tal que $\mathcal{T}'' = \mathcal{T}'$ e $\mathcal{T}''_{2,4} = before$. A matriz que representa o conjunto $T''$ é a seguinte:

| $\mathcal{T}''$ | $e_1$ | $e_2$ | $f_3$ | $s_4$ |
|---|---|---|---|---|
| $e_1$ | | $before$ | $meets$ | $NULL$ |
| $e_2$ | | | $during$ | $before$ |
| $f_3$ | | | | $before$ |
| $s_4$ | | | | |

- Para $i = 1$ :

  - Para $\mathcal{T}''$, $r = before$ é o único predicado temporal que pode relacionar as variáveis temporais $e_1$ e $s_4$ conservando a consistência de $\mathcal{T}'$. Assim, é criado um conjunto $\mathcal{T}''$ tal que $\mathcal{T}''' = \mathcal{T}''$ e $\mathcal{T}'''_{1,4} = before$. A matriz que representa o conjunto $T'''$ é a seguinte:

| $\mathcal{T}'''$ | $e_1$ | $e_2$ | $f_3$ | $s_4$ |
|---|---|---|---|---|
| $e_1$ | | $before$ | $meets$ | $before$ |
| $e_2$ | | | $during$ | $before$ |
| $f_3$ | | | | $before$ |
| $s_4$ | | | | |

  Os passos anteriores devem ser aplicados sobre todos os conjuntos de $ST$, caso exista mais de um, até que todas as entradas sejam definidas.

4. O conjunto $ST$ constituído de conjuntos de átomos temporais consistentes e completos é retornado.

Se a nova variável temporal incluída em $\mathcal{T}$ fosse intervalar, seríam gerados seis conjuntos $\mathcal{T}'$, um para cada $r \in \mathcal{P}_{temp}$, tal que, $\mathcal{T}' = \mathcal{T}$ e $\mathcal{T}'_{3,4} = r$ (da linha 2 até a 7). Seríam gerados seis conjuntos $\mathcal{T}'$ porque de acordo com a adaptação da estrutura temporal da LTI, para relacionar duas variáveis temporais intervalares, $r$ pode ser $before$, $meets$, $overlaps$, $starts$, $during$ e $finishes$. Os passos descritos anteriormente devem ser aplicados nos seis conjuntos gerados até que sejam gerados todos os conjuntos possíveis, completos e consistentes, de átomos temporais.

Com as funções $Match(\ )$ e $TExtension(\ )$ apresentadas, é possível descrever o algoritmo principal de especialização por extensão, apresentado na figura 5.7.

A função $Extension(P, \mathcal{R})$ especializa um $pth$ $P$ de acordo com os seguintes passos:

1. Primeiramente, constrói um autômato determinístico de estados finitos $\mathcal{A}_R$ associado a expressão regular $\mathcal{R}$, tal que, $q_0$ é o estado inicial de $\mathcal{A}_R$ (linha 2).
2. Encontra a palavra $W$ associada ao $pth$ $P$ com o auxílio da função $Match(P, \mathcal{R})$ (linha 3).

**Função** Extension($P$,$\mathcal{R}$)

1. $PE := \emptyset$;
2. Seja $\mathcal{A}_R$ se um autômato associado a $\mathcal{R}$, tendo $q_0$ como estado inicial;
3. Seja $W = Match(P, \mathcal{R})$;
4. Seja $W = w_1 \dots w_n$;
5. Seja $x = q_0$;
6. **For** $i = 1$ **to** $n$ **do**;
7. **If** $(x \stackrel{w_i}{\rightarrow} q_j) \in \mathcal{A}_R$ **then** $x = q_j$;
8. **For each** $w_{n+1}$ tal que $(x \stackrel{w_{n+1}}{\rightarrow} q)$ **do begin**
9. Seja $w_{n+1} = p_{n+1}(u_1^{n+1}, \dots, u_l^{n+1}, \#)$;
10. Seja $v = \{z_1, \dots, z_n\} \subseteq \mathcal{V}ar$;
11. **For** $i = 1$ **to** $l$ **do begin**
12. **If** $u_i^{n+1} \in \mathcal{V}ar$ **then** $y_i^{n+1} = u_i^{n+1}$;
13. **Else** $y_i^{n+1} = z_i$;
14. **End for**;
15. Seja $\mathcal{E}' = \{p_1(y_1^1, \dots, y_l^1, \tau_1), \dots, p_n(y_1^n, \dots, y_l^n, \tau_n), \dots, p_{n+1}(y_1^{n+1}, \dots, y_l^{n+1}, \tau_{n+1}\}$;
16. $ST = Textension(\mathcal{T}, w_{n+1})$
17. **For each** $\mathcal{T} \in ST$ **do begin**
18. Seja $P' = \{K, \mathcal{E}', \mathcal{T}'\}$
19. $PE := PE \cup \{P'\}$;
20. **End for**;
21. **End for**;
22. **Return** $PE$;

Figura 5.7: Função $Extension(P, \mathcal{R})$

3. Encontra um estado $q_j$ no autômato $\mathcal{A}_R$ atingido a partir do estado inicial $q_0$ seguindo seqüência de *strings* de $W$ ($q_0 \stackrel{W}{\rightarrow} q_j$) (da linha 5 até a 7).
4. Para cada modo $w_{n+1}$, tal que existe uma transição saindo de $q_j$ para qualquer estado $q$ ($q_j \stackrel{w_{n+1}}{\rightarrow} q$), realiza os seguintes passos:
   (a) Transforma o modo $w_{n+1}$ em um átomo de dados $p_{n+1}(y_1^{n+1}, \dots, y_l^{n+1}, \tau_{n+1})$ de acordo com $w_{n+1}$ (da linha 11 até a 14).
   (b) Constrói um novo conjunto de átomos de dados $\mathcal{E}'$ acrescentando o novo átomo de dados $p_{n+1}$ no fim de $\mathcal{E}$ (linha 15).
   (c) Constrói um conjunto $ST$, constituído por conjuntos de átomos temporais $\mathcal{T}'$, com o auxílio da função $Textension(\mathcal{T}, w_{n+1})$ (linha 16).
   (d) Da linha 17 até a 19, para cada $\mathcal{T}' \in ST$:

- Constrói um novo *pth* $P' = \{K, \mathcal{E}', \mathcal{T}'\}$ (linha 18).
- Insere $P'$ no conjunto $PE$ (linha 19).

5. Retorna o conjunto $PE$ de *pth's* obtidos pela extensão de $P$.

**Exemplo 5.1.5** Considere a expressão regular $\mathcal{R} = Hormônio(x, \$, \#)^* Fumo(x, \$, \#)$ $Birads(x, \$, \#)$ e o *pth* $P = (Paciente(x),\ \{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),$ $Fumo(x, z, f_3)\},\ \{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})$ pertencente a $F_3$, cujo conjunto de átomos temporais foi especializado no exercício anterior.

1. O autômato $\mathcal{A}_R$ associado a expressão regular $\mathcal{R}$ é mostrado na figura 5.8.

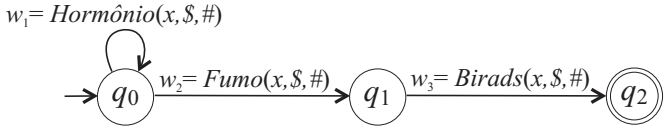



Figura 5.8: Autômato $\mathcal{A}_R$ associado a expressão regular $\mathcal{R}$

2. A palavra associada a $P$ é $W = Hormônio(x, \$, \#)Hormônio(x, \$, \#)Fumo(x, \$, \#)$.

3. O estado atingido a partir de $q_0$ seguindo a seqüência de *strings* de $W$ é $q_1$ ($q_0 \xrightarrow{W} q_1$ é uma transição de $\mathcal{A}_R$).

4. Como $q_1 \xrightarrow{w_3} q_2$ é uma transição de $\mathcal{A}_R$, $w_{n+1} = Birads(x, \$, \#)$, assim:

   (a) $Birads(x, \$, \#)$ é transformado no átomo de dados $Birads(x, h, s_4)$.

   (b) $\mathcal{E}' = \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, z, f_3), Birads(x, h, s_4)\}$.

   (c) $ST = Textension(\mathcal{T}, Birads(x, \$, \#))$ é o conjunto de $\mathcal{T}'$ gerados a partir de $\mathcal{T}$.

   (d) Para cada $\mathcal{T}' \in ST$ é gerado um $pthP' = \{K, \mathcal{E}', \mathcal{T}'\}$ e $PE = PE \cup \{P'\}$.

   Como, de acordo com o exemplo anterior, $ST$ é constituído por um único conjunto $\mathcal{T}'$ que foi renomeado para $\mathcal{T}'''$ para evitar confusão, $PE$ é constituído somente pelo *pth* $P'_5 = (Paciente(x), \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2),$ $Fumo(x, z, f_3),\ \ Birads(x, h, s_4)\},\ \ \{\mathcal{T}'''\}\ ),\ \ \text{onde},\ \ \mathcal{T}''' = \{before(e_1, e_2),$ $meets(e_1, f_3),\ before(e_1, s_4),\ during(e_2, f_3),\ before(e_2, s_4),\ before(f_3, s_4)\}$.

5. É retornado o conjunto $PE = \rho_E(P)$.

Como nos exemplos apresentados nesta seção foi utilizado um conjunto $F_3$ constituído por um único *pth* $P = (Paciente(x),\{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),\ Fumo(x, z, f_3)\},\ \{before(e_1, e_2),meets(e_1, f_3),during(e_2, f_3)\})$, no fim da etapa de geração são inseridos no conjunto $C_4$ os conjuntos $PI$ e $PE$, obtidos no exemplo 5.1.2 e 5.1.5, respectivamente. Portanto, $C_4$ é constituído pelos seguintes *pth's*:

$$
\begin{aligned}
C_4 = \{\ \ P'_1 &= (Paciente(x),\{Hormônio(x, estradiol, e_1),Hormônio(x, y, e_2),\\
&\quad\ Fumo(x, z, f_3)\},\{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})\\
P'_2 &= (Paciente(x),\{Hormônio(x, y, e_1),Hormônio(x, estradiol, e_2),\\
&\quad\ Fumo(x, z, f_3)\},\{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})\\
P'_3 &= (Paciente(x),\ \{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),\\
&\quad\ Fumo(x, 10, f_3)\},\{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})\\
P'_4 &= (Paciente(x),\ \{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),\\
&\quad\ Fumo(x, 20, f_3)\},\{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})\\
P'_5 &= (Paciente(x),\{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),\\
&\quad\ Fumo(x, z, f_3),\ Birads(x, h, s_4)\},\{before(e_1, e_2),\ meets(e_1, f_3),\\
&\quad\ before(e_1, s_4),\ during(e_2, f_3),\ before(e_2, s_4),\ before(f_3, s_4)\})\ \ \}
\end{aligned}
$$

Se existissem mais *pth's* em $F_3$, todos os conjuntos $PI$ e $PE$ gerados pela especialização desses *pth's* deveriam ser inseridos em $C_4$.

### 5.1.2 Etapa de Poda dos Candidatos

A propriedade antimonotônica é empregada na etapa de poda para eliminar os *pth's* que não são potencialmente freqüentes. Relembrando: *pela propriedade antimonotônica, um padrão freqüente não pode ser mais específico que um padrão não freqüente.* Portanto, em um nível $k$, são podados de $C_k$ todos os *pth's* $P$ que possuem pelo menos um $P' \succ P$ que satisfaz $\mathcal{R}$ (*p-válido*) e que não pertence a $F = F_1 \cup \ldots \cup F_{k-1}$ (não freqüente).

**Exemplo 5.1.6** Considere a expressão regular $\mathcal{R} = Hormônio(x, \$, \#)^* Fumo(x, \$, \#)$ $Birads(x, \$, \#)$, o conjunto $C_4$ obtido anteriormente na etapa de geração e o seguinte conjunto $F = F_1 \cup F_2 \cup F_3$. Observe que o *pth* $P$ que resultou nos *pth's* de $C_4$ também faz parte de $F$.

$$
\begin{aligned}
F = \{\ & P_1 = (Paciente(x),\ \{Hormônio(x, y, e_1)\},\{\emptyset\}),\\
& P_2 = (Paciente(x),\ \{Fumo(x, z, f_1)\},\{\emptyset\}),\\
& P_3 = (Paciente(x),\ \{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2)\},\ \{before(e_1, e_2)\}),\\
& P_4 = (Paciente(x),\ \{Hormônio(x, y, e_1),\ Fumo(x, z, f_2)\},\ \{before(e_1, f_2)\}),\\
& \boldsymbol{P} = (Paciente(x),\ \{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2), Fumo(x, z, f_3)\},\\
& \qquad \{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})
\end{aligned}
$$

Os *pth's* $P_1'$ e $P_2'$ são podados de $C_4$. O *pth* $P_1'$ possui dois sub-padrões: $(Paciente(x),$ $\{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, z, f_3)\}, \{before(e_1, e_2), meets(e_1, f_3),$ $during(e_2, f_3)\})$ e $(Paciente(x),\ \{Hormônio(x, estradiol, e_1),\ Hormônio(x, y_2, e_2)\},$ $\{before(e_1, e_2)\})$. Embora o primeiro *pth* seja freqüente, o segundo não é. Assim, de acordo com a propriedade antimonotônica, $P_1'$ não é potencialmente freqüente e pode ser eliminado. O *pth* $P_2'$ também possui dois sub-padrões: $(Paciente(x), \{Hormônio(x, y_1, e_1),$ $Hormônio(x, y_2, e_2),\ Fumo(x, z, f_3)\},\ \{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})$ e $(Paciente(x),\ \{Hormônio(x, y, e_1),\ Hormônio(x, estradiol, e_2)\},\ \{before(e_1, e_2)\})$. Embora o primeiro *pth* seja freqüente, o segundo não é. Assim, de acordo com a propriedade antimonotônica, $P_2'$ não é potencialmente freqüente e também pode ser eliminado.

Após a poda o conjunto $C_4$ é o seguinte:

$$
\begin{aligned}
C_4 = \{\ & P_3' = (Paciente(x),\ \{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),\\
& \qquad Fumo(x, 10, f_3)\},\{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})\\
& P_4' = (Paciente(x),\ \{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),\\
& \qquad Fumo(x, 20, f_3)\},\{before(e_1, e_2),\ meets(e_1, f_3),\ during(e_2, f_3)\})\\
& P_5' = (Paciente(x),\{Hormônio(x, y_1, e_1),\ Hormônio(x, y_2, e_2),\\
& \qquad Fumo(x, z, f_3),\ Birads(x, h, s_4)\},\{before(e_1, e_2),\ meets(e_1, f_3),\\
& \qquad before(e_1, s_4),\ during(e_2, f_3),\ before(e_2, s_4),\ before(f_3, s_4)\})\ \ \}
\end{aligned}
$$

### 5.1.3 Etapa de Avaliação (Cálculo do Suporte)

Na etapa de avaliação de um nível $k$ é calculado o suporte de todos os *pth's* que restaram em $C_k$ após a poda. Para isso, o banco de dados é percorrido e cada *pth* tem um contador incrementado em um cada vez que uma tupla do banco de dados o *suportar*. Ao terminar

de percorrer o banco de dados, o valor do contador de cada *pth* será a quantidade de tuplas que o suportou. Finalmente, este valor é dividido pela quantidade de tuplas da tabela de referência do banco de dados resultando no suporte do *pth*, como definido na seção 3.1.4 do capítulo 3. São considerados freqüentes, e portanto são inseridos em $F_k$, todos os *pth's* que possuem suporte superior a um suporte mínimo definido pelo usuário.

### Quando uma tupla suporta um *pth*?

Antes de mais nada, para verificar se uma determinada tupla de um banco de dados temporal suporta um determinado *pth*, é necessário que as seguintes definições estejam bem claras.

**Definição 5.1.2 (Seqüência do *pth*)** Seja $P = (K, \mathcal{E}, \mathcal{T})$ um *pth*, com $\mathcal{E} = \{p_1, \ldots, p_n\}$. A **seqüência de** $P$ definida por $SP(P)$, é uma estrutura da forma $< L_1, \ldots, L_n >$, onde, $n$ é o número de átomos de dados de $P$ e $L_i$ é uma lista $[s_1, \ldots, s_l]$ de variáveis (diferentes das variáveis de referência) ou constantes que aparecem no átomo de dados $i$.

**Exemplo 5.1.7** Seja um *pth* $P = (Paciente(x)$, $\{Hormônio(x, estradiol, e_1)$, $Fumo(x$, $z, f_2)$, $Birads(x, h, s_3)\}$, $\{during(e_1, f_2)$, $before(e_1, s_3)$, $before(f_2, s_3)\})$. A seqüência de $P$ é $SP(P) = <[estradiol], [z], [h]>$.

**Definição 5.1.3 (Seqüências da tupla)** Seja $BD$ um banco de dados temporal sobre o esquema $\mathbf{R} = \{K, R_1, ..., R_k\}$ e $U = ((w_1, ..., w_m), (y_1, ..., y_l, \tau_1)_1, \ldots, (y_1, ..., y_l, \tau_n)_n)$ uma tupla de $BD$, onde $w_i$ é um atributo de referência, $y_i$ é um atributo de dado e $\tau_i$ é um atributo temporal. A **seqüência de dados de** $U$, definida por $ST(U)$, é uma estrutura da forma $< T_1, \ldots, T_n >$, onde $n$ é o número de relações que não são de referência e $T_i$ é uma lista $[Y_1, \ldots, Y_l]$ de constantes não temporais. A **seqüência temporal de** $U$, definida por $St(U)$, é uma estrutura da forma $< t_1, \ldots, t_n >$, onde $t_j = \tau_j$, e $\tau_j$ é mapeado em um intervalo $[a, b]$, e onde $a$ e $b$ são inteiros mapeados em datas. Se $\tau_j$ representa um intervalo não nulo, então $a < b$, se $\tau_j$ representa um intervalo nulo (ponto), então $a = b$.

**Exemplo 5.1.8** Seja um banco de dados temporal $BD$ sobre o esquema $\mathbf{R} = \{Paciente(NomePac)$, $Hormônio(NomePac, NomeHorm, T_{it})$, $Fumo(NomePac, Qtde$,

$T_{it}), Birads(NomePac, Categoria, T_{pt})\}$ e a tupla $U$, mostrada na figura 5.9, que é o resultado da junção das tabelas *Hormônio*, *Fumo* e *Birads*.

| *NomePac* | *NomeHorm* | $T_{it}$ | *Qtde* | $T_{it}$ | *Categoria* | $T_{pt}$ |
|---|---|---|---|---|---|---|
| Maria | estradiol | [4,6] | 10 | [2,7] | 3 | 9 |

Figura 5.9: Tupla do banco de dados $BD$

A seqüência de dados de $U$ é $ST(U) = < [estradiol], [10], [3] >$ e a seqüência temporal de $U$ é $St(U) = < [4, 6], [2, 7], [9, 9] >$.

**Definição 5.1.4 (Seqüências compatíveis)** Seja a seqüência de dados $ST(U) = < T_1, \ldots, T_n >$ de uma tupla $U$, onde, $T_i = [Y_1, \ldots, Y_l]$ e $Y_j$ é uma constante. Seja também a seqüência $SP(P) = < L_1, \ldots, L_n >$ de um *pth* $P$, onde $L_i = [s_1, \ldots, s_l]$ e $s_j$ é uma variável ou uma constante. $ST(U)$ é **compatível** com $SP(P)$ se todos os elementos de $ST(U)$ forem compatíveis com todos os elementos de $SP(P)$ da seguinte forma:

(a) Se $s_j$ $\{j = 1, \ldots, l\}$ é uma variável, então qualquer valor de $Y_j$ é compatível com $s_j$;

(b) Se $s_j$ $\{j = 1, \ldots, l\}$ é uma constante, então $Y_j$ é compatível com $s_j$ se e somente se $s_j = Y_j$.

**Exemplo 5.1.9** Considere a tupla $U$ com $ST(U) = < [estradiol], [10], [3] >$ do exemplo anterior e o *pth* $P$ com $SP(P) = < [estradiol], [z], [h] >$ do exemplo 5.1.7. $ST(U)$ é compatível com $SP(P)$. Entretanto, $ST(U)$ não é compatível com a seqüência $SP(P') = < [progesterona], [10], [w] >$ de um outro *pth* $P'$, porque a condição (b) não é verificada.

Se for verificada compatibilidade entre a seqüência de dados de uma tupla e a seqüência de um *pth*, o próximo passo para verificar se esta tupla suporta o *pth* é analisar se a seqüência temporal da tupla satisfaz os átomos temporais do *pth*.

A seqüência temporal $St(U) =< t_1, t_2, t_3 >$ de uma tupla $U$ satisfaz os átomos temporais de um *pth* $P = (K, \mathcal{E}, \mathcal{T})$, com, $\mathcal{E} = \{p_1(x_1, \ldots, x_n, \tau_1), p_2(x_1, \ldots, x_n, \tau_2), p_3(x_1, \ldots, x_n, \tau_3)\}$ e $\mathcal{T} = \{r_1(\tau_1, \tau_2), r_2(\tau_1, \tau_3), r_3(\tau_2, \tau_3)\}$ se os valores da seqüência temporal da tupla não contradizem os átomos temporais do *pth*, ou seja, se para cada $r_i(\tau_j, \tau_k) \in \mathcal{T}$, existe $\tau_j = t_j$ e $\tau_k = t_k$, tal que, $r_i$ é satisfeito.

**Exemplo 5.1.10** Considere a tupla $U$ do exemplo 5.1.8, cuja seqüência temporal é $St(U) = <[4,6],\ [2,7],\ [9,9]>$ e o *pth* $P = (Paciente(x),\ \{Hormônio(x, estradiol, e_1),$ $Fumo(x, z, f_2),\ Birads(x, h, s_3)\},\ \{during(e_1, f_2), before(e_1, s_3), before(f_2, s_3)\})$. O átomo $during(e_1, f_2)$ é satisfeito, pois $e_1 = [4,6]$ e $f_2 = [2,7]$. Portanto, $e_1$ ocorre durante $f_2$. O átomo $before(e_1, s_3)$ é satisfeito, pois $e_1 = [4,6]$ e $s_3 = [9,9]$, e portanto $e_1$ ocorre antes de $s_3$. O átomo $before(f_2, s_3)$ é satisfeito, pois $f_2 = [2,7]$ e $s_3 = [9,9]$, e portanto $f_2$ ocorre antes de $s_3$. Com isso, $St(U)$ satisfaz $P$ no que diz respeito aos átomos temporais. Se pelo menos um átomo temporal de $P$ não for satisfeito por $St(U)$, então $St(U)$ não satisfaz os átomos temporais de $P$.

Como mostrado no exemplo 5.1.9, a seqüência de dados $ST(U)$ da tupla $U$ é compatível com a seqüência do *pth* $P$, e como mostrado no exemplo anterior, a seqüência temporal $St(U)$ da tupla $U$ satisfaz os átomos temporais de $P$. Portanto, a tupla $U$ suporta o *pth* $P$.

É importante chamar a atenção para a segunda condição necessária para que uma tupla suporte um *pth*. Como foi mostrado no exemplo anterior, esta condição trata dos relacionamentos temporais existentes entre os eventos de um *pth* e para sua verificação são considerados exatamente o início e o fim de cada intervalo. Por exemplo, o átomo temporal $before(e_1, e_2)$ é satifeito se existir uma instanciação $e_1 = [4,6]$ e $e_2 = [7,8]$, portanto, se a instanciação for $e_1 = [4,7]$ ou $e_1 = [4,8]$ e $e_2 = [7,8]$ o átomo temporal $before(e_1, e_2)$ não é mais satisfeito. Considerando agora átomo temporal $meets(e_1, e_2)$, ele é satifeito se existir uma instanciação $e_1 = [4,6]$ e $e_2 = [6,8]$, portanto, se a instanciação for $e_1 = [4,5]$ ou $e_1 = [4,7]$ e $e_2 = [6,8]$ o átomo temporal $meets(e_1, e_2)$ não é mais satisfeito. Cada predicado temporal interpreta os relacionamentos entre o início e o fim dos intervalos de maneira diferente como mostrado na adaptação da estrutura temporal da LTI.

Em muitas aplicações, como por exemplo na medicina, agronomía, agropecuária, etc., se forem considerados exatamente os limites (início e fim) dos intervalos para verificar se os átomos temporais de um *pth* são satisfeitos pela seqüência temporal de uma tupla, padrões bastante expressivos e conclusivos podem não ser minerados pelo simples fato dos eventos terem ocorrido em períodos diferentes, embora um especialista possa

considerar esta diferença desprezível. Por exemplo, considere o *pth* $P = (Paciente(x), \{Menopausa(x, sim, s_1),\ Antidepressivo(x, y, e_2)\}, \{before(s_1, e_2)\})$ e a tupla $U$, com $St(U) = < [2, 2], [3, 5] >$, onde, 2 é mapeado na data 15/10/2005, 3 é mapeado na data 15/11/2005 e 5 é mapeado na data 20/01/2006. Se forem considerados exatamente os limites dos intervalos, o átomo temporal $before(s_1, e_2)$ é satisfeito, embora, neste caso um especialista possa considerar os dias irrelevantes, pois, a paciente pode não lembrar exatamente o dia em que entrou na menopausa ou o dia em que iniciou o uso de um determinado antidepressivo. Assim, para uma análise mais correta o especialista pode utilizar, por exemplo, uma granularidade em meses, bimestres, semestres, etc.

Considerando os mapeamentos anteriores, uma granularidade definida em semestres sobre eles sería 2 = 2005/02, que significa que a paciente entrou na menopausa no segundo semestre de 2005, 3 = 2005/02 e 5 = 2006/01, que significa que a paciente usou um determinado antidepressivo do segundo semetre de 2005 até o primeiro semestre de 2006. Neste caso, o átomo temporal $before(s_1, e_2)$ do *pth* $P$ não é mais satisfeito, pois o especialista associou a qualquer data de um mesmo semestre um mesmo valor. Considerando agora um *pth* $P' = (Paciente(x),\ \{Menopausa(x, sim, s_1),\ Antidepressivo(x, y, e_2)\}, \{meets(s_1, e_2)\})$ e a granularidade em semestres, o átomo temporal $meets(s_1, e_2)$ é satisfeito pela seqüência temporal da tupla $U$.

Portanto, para realizar uma mineração bem sucedida, minerando *pth's* ricos em informações temporais, ou seja, com grande diversidade entre os predicados temporais, e com coerência com a aplicação, é fundamental que o usuário defina de forma adequada uma granularidade.

Para possibilitar que o usuário especifique o tipo de granularidade que ele deseja utilizar durante o processo de mineração foi desenvolvida a função *Granularity*( ), apresentada na figura 5.10. Esta função é chamada sempre que o suporte de um *pth* vai ser calculado e permite que sejam especificados sete tipos de granularidade ($\delta$), ou seja, $\delta$ pode ser $dia, semana, quinzena, mês, bimestre, trimestre, semestre$ ou $ano$.

O algoritmo anterior tem como entrada um intervalo cujo início e fim são datas, e uma granularidade. Na saída do algoritmo o mesmo intervalo de entrada é retornado, entretanto, mapeado na granularidade escolhida pelo usuário. Por exemplo, se o intervalo

```
Função Granularity(intervalo, δ-granularidade)
1.  If ( δ == dia )
2.     intervalo_gran = funcDia(intervalo);
3.  If ( δ == semana )
4.     intervalo_gran = funcSemana(intervalo);
5.  If ( δ == quinzena )
6.     intervalo_gran = funcQuinzena(intervalo);
7.  If ( δ == mes )
8.     intervalo_gran = funcMes(intervalo);
9.  If ( δ == bimestre )
10.    intervalo_gran = funcBimestre(intervalo);
11.  If ( δ == trimestre )
12.    intervalo_gran = funcTrimestre(intervalo);
13.  If ( δ == semestre )
14.    intervalo_gran = funcSemestre(intervalo);
15.  If ( δ == ano )
16.    intervalo_gran = funcAno(intervalo);
17.  return intervalo_gran;
```

Figura 5.10: Função $Granularity(intervalo, \delta - granularidade)$

de entrada for [10/02/2006, 15/08/2006] e a granularidade for definida em meses, o intervalo retornado pelo algoritmo será [2006/02, 2006/08]. O suporte de um *pth* é calculado somente após todos os intervalos que fazem parte de seus eventos terem sido mapeados de acordo com a granularidade especificada.

### Aumento da Performance da Etapa de Avaliação

Como já mencionado nesta dissertação, a etapa de avaliação é uma etapa dos algoritmos de mineração de dados importante e complexa e possui um custo computacional elevado. Por este motivo o MILPRIT* utiliza algumas técnicas para melhorar o desempenho desta etapa. Para um bom entendimento destas técnicas, é necessário que algumas definições seja introduzidas.

**Definição 5.1.5 (Matriz característica de um *pth*)** Seja $\Sigma = \{w_1, ..., w_n\}$ um alfabeto de modos sobre o esquema $\mathbf{R} = \{K, R_1, ..., R_n\}$. A **matriz característica de um *pth*** $P$ definida por $\mathcal{M}_{\mathcal{C}}(P)$, contém os modos $\{w_1, ..., w_n\}$ de $\Sigma$ e números $\{m_1, \ldots, m_n\}$,

onde, $m_i$ é a freqüência com que cada modo foi utilizado na geração dos átomos de $P$. A estrutura de uma matriz característica é mostrada na figura 5.11.

| *Modos* | *Freqüência* |
|---|---|
| $w_1$ | $m_1$ |
| $\vdots$ | $\vdots$ |
| $w_n$ | $m_n$ |

Figura 5.11: Estrutura da matriz característica de um *pth*

**Exemplo 5.1.11** Considere o alfabeto de modos $\Sigma = \{w_1 = Hormônio(x, *, \#),\ w_2 = Fumo(x, *, \#),\ w_3 = Birads(x, *, \#)\}$ e o conjunto $C_4$ obtido após a etapa de poda, mostrado na seção anterior.

Relembrando, o conjunto $C_4$ é constituído pelos *pth's* $P'_3 = \{Paciente(x), \mathcal{E}'_3, \mathcal{T}'_3\}$, $P'_4 = \{Paciente(x), \mathcal{E}'_4, \mathcal{T}'_4\}$ e $P'_5 = \{Paciente(x), \mathcal{E}'_5, \mathcal{T}'_5\}$, onde:

- $\mathcal{E}'_3 = \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, 10, f_3)\}$
- $\mathcal{E}'_4 = \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, 20, f_3)\}$
- $\mathcal{E}'_5 = \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, z, f_3), Birads(x, h, s_4)\}$

A figura 5.12 mostra as matrizes característica dos *pth's* $P'_3$, $P'_4$ e $P'_5$.

| *Modos* | *Ocorrências* |
|---|---|
| $w_1$ | 2 |
| $w_2$ | 1 |
| $w_3$ | 0 |

(a)

| *Modos* | *Ocorrências* |
|---|---|
| $w_1$ | 2 |
| $w_2$ | 1 |
| $w_3$ | 1 |

(b)

Figura 5.12: (a) matriz característica de $P'_3$ e $P'_4$, (b) matriz característica de $P'_5$

Os *pth's* $P'_3$ e $P'_4$ possuem dois átomos gerados pelo modo $w_1$, um átomo gerado do modo $w_2$ e nenhum do átomo gerado pelo modo $w_3$. Já o *pth* $P'_5$ possui dois átomos gerados pelo modo $w_1$, um átomo gerado pelo modo $w_2$ e um átomo gerado pelo modo $w_3$.

Antes de efetivamente calcular o suporte de todos os *pth's* que restaram em $C_k$ após a poda, o MILPRIT* particiona o conjunto $C_k$, ou seja, insere em uma mesma partição *pth's* com matrizes característica iguais. O fato de *pth's* com matrizes característica iguais serem inseridos em uma mesma partição permite que seja associada a cada partição uma

única expressão SQL (definida abaixo). Com isso, para calcular o suporte dos *pth's* de cada partição é realizada uma única consulta ao banco de dados (disco), agilizando a etapa de avaliação.

**Definição 5.1.6 (Expressão SQL associada a uma partição)** Seja o esquema de banco de dados $\mathbf{R} = \{K(at_1),\ R_1(at_1, at_2, T_{it}),\ R_2(at_1, at_2, T_{it}),\ R_3(at_1, at_2, T_p),\ \ldots,\ R_n\}$, onde $T_{it}$ representa dois atributos temporais $st$ e $en$ que representam o início e o fim de um intervalo, respectivamente, e $T_p$ representa o atributo temporal pontual $p$. Seja também uma partição $\partial$ constituída por uma único *pth* $P = (K(x),\ \{R_1(x, y, e_1),\ R_2(x, z, f_2),\ R_3(x, h, s_3)\},\ \{\mathcal{T}\})$. A **expressão SQL associada a** $\partial$ tem a seguinte estrutura:

**SELECT** $R_1.at_1, R_1.at_2, R_1.st, R_1.en, R_2.at_2, R_2.st, R_2.en, R_3.at_2, R_3.p$ **FROM** $K, R_1, R_2, R_3$ **WHERE** $R_1.at_1 = K.at_1$ **AND** $R_1.at_1 = R_2.at_1$ **AND** $R_1.at_1 = R_3.at_1$ **AND** $(R_1.en < R_2.en$ **OR** $(R_1.en = R_2.en$ **AND** $R_1.st > R_2.st))$ **AND** $(R_2.en < R_3.p)$

Observe que a expressão SQL é construída com base nos *pth's* de uma determinada partição, ou seja, a ordem com que as tabelas são selecionadas respeita a seqüência dos átomos de dados dos *pth's*. Isto simplifica a construção das seqüências das tuplas e evita que sejam consideradas tuplas que certamente não suportarão os *pth's*.

**Exemplo 5.1.12** Considere os conjunto $C_4$ constituído pelos *pth's* $P'_3$, $P'_4$ e $P'_5$ e o seguinte esquema de banco de dados:

$$\mathbf{R} = \{Paciente(NomePac), Hormônio(NomePac, NomeHorm, st, en),$$
$$Fumo(NomePac, Qtde, st, en), Birads(NomePac, Categoria, p)\}$$

Como os *pth's* $P'_3$ e $P'_4$ possuem matrizes característica iguais, eles são inseridos em uma mesma partição (Partição 1) e o *pth* $P'_5$ é inserido em outra partição (Partição 2).

Para a Partição 1 é construída a seguinte expressão $SQL_1$:

**SELECT** $H_1.NomePac$, $H_1.NomeHorm$, $H_1.st$, $H_1.en$, $H_2.NomeHorm$, $H_2.st$, $H_2.en$, $Fumo.Qtde$, $Fumo.st$, $Fumo.en$ **FROM** $Paciente$, $Hormônio$ **as** $H_1$, $Hormônio$ **as** $H_2$, $Fumo$ **WHERE** $H_1.NomePac = Paciente.NomePac$ **AND** $H_1.NomePac =$

$H_2.NomePac$ **AND** $H_1.NomePac = Fumo.NomePac$ **AND** $(H_1.en < H_2.en$ **OR** $(H_1.en = H_2.en$ **AND** $H_1.st > H_2.st))$ **AND** $(H_2.en < Fumo.en$ **OR** $(H_2.en = Fumo.en$ **AND** $H_2.st > Fumo.st))$

Para a Partição 2 é construída a seguinte expressão $SQL_2$:

**SELECT** $H_1.NomePac$, $H_1.NomeHorm$, $H_1.st$, $H_1.en$, $H_2.NomeHorm$, $H_2.st$, $H_2.en$, $Fumo.Qtde$, $Fumo.st$, $Fumo.en$, $Birads.Categoria$, $Birads.p$ **FROM** $Paciente$, $Hormônio$ **as** $H_1$, $Hormônio$ **as** $H_2$, $Fumo$, $Birads$ **WHERE** $H_1.NomePac = Paciente.NomePac$ **AND** $H_1.NomePac = H_2.NomePac$ **AND** $H_1.NomePac = Fumo.NomePac$ **AND** $H_1.NomePac = Birads.NomePac$ **AND** $(H_1.en < H_2.en$ **OR** $(H_1.en = H_2.en$ **AND** $H_1.st > H_2.st))$ **AND** $(H_2.en < Fumo.en$ **OR** $(H_2.en = Fumo.en$ **AND** $H_2.st > Fumo.st))$ **AND** $(Fumo.en < Birads.p)$

Portanto, a divisão do conjunto $C_4$ resulta em partições com estrutura semelhante a mostrada na figura 5.13.

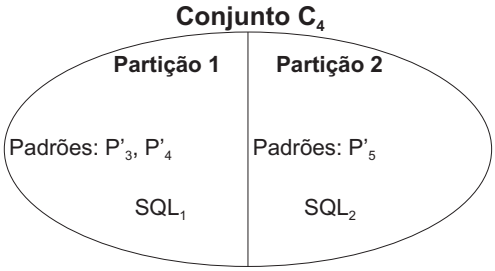



Figura 5.13: Conjunto $C_4$ após particionamento

Após particionar o conjunto $C_k$, o MILPRIT* ainda insere os *pth's* de cada partição em uma **árvore *hash*** similar a utilizada em [3]. Com isso, para verificar se os *pth's* são suportados por uma tupla do banco de dados, não é preciso analisar todos os *pth's* da partição, mas sim os *pth's* que são potencialmente suportados pela tupla, ou seja, os que estão em alguns ramos da árvore.

**Definição 5.1.7 (Função *hash*)** A **função *hash*** é definida por $h(x) = ((x \ mod \ (N - 1)) + 1)$, onde, $x$ é o elemento da seqüência do *pth* e $N$ é o número máximo de filhos em cada nó. Esta função *hash* possui as seguintes propriedades:

(a) se o elemento que está sendo analisado é uma constante, então $0 < h(c) \leq N$.

(b) se o elemento que está sendo analisado é uma variável, então $h(v) = 0$.

Para inserir um *pth* na árvore *hash*, a função *hash* é aplicada em cada elemento da seqüência do *pth* para decidir qual caminho seguir na árvore. Após a aplicação da função *hash* em todos os elementos da seqüência do *pth*, ele é inserido em um nó folha. É importante salientar que todos os *pth's* de uma determinada partição possuem a mesma quantidade de elementos em suas seqüências. Portanto, uma árvore *hash* possui profundidade igual a quantidade de elementos das seqüências dos *pth's* da partição que está sendo analisada.

**Exemplo 5.1.13** Considere a Partição 1 do conjunto $C_4$, constituída pelos *pth's* $P'_3$ e $P'_4$, e número máximo de filhos em cada $N = 3$.

$P'_3 = (Paciente(x), \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, 10, f_3)\}, \mathcal{T}'_2\})$
$P'_4 = (Paciente(x), \{Hormônio(x, y_1, e_1), Hormônio(x, y_2, e_2), Fumo(x, 20, f_3)\}, \mathcal{T}'_3\})$

Para possibilitar a aplicação da função *hash* sobre os elementos das seqüências dos *pth's*, considere a substituição das constantes das seqüências pela soma dos códigos ASCII de cada caracter. Assim:

$SP(P'_3) = < [y_1], [y_2], [10] >= < [y_1], [y_2], [97] >$
$SP(P'_4) = < [y_1], [y_2], [20] >= < [y_1], [y_2], [98] >$

Os valores retornados pela função hash aplicada sobre os elementos das seqüências dos *pth's* $P'_3$ e $P'_4$ são: $h(97) = 2$, $h(98) = 1$ e $h(y_1) = h(y_2) = 0$.

A figura 5.14 ilustra a árvore *hash* após a inserção dos *pth's*.

Para verificar se uma determinada tupla suporta os *pth's* de uma determinada partição, os quais estão inseridos em uma árvore *hash*, a função *hash* é aplicada sobre cada elemento da seqüência de dados da tupla. Com isso, são alcançados somente nós folhas que possuem *pth's* com possibilidade de serem suportados pela tupla, evitando analisar *pth's* que certemente não serão suportados por ela. Para garantir que sejam recuperados

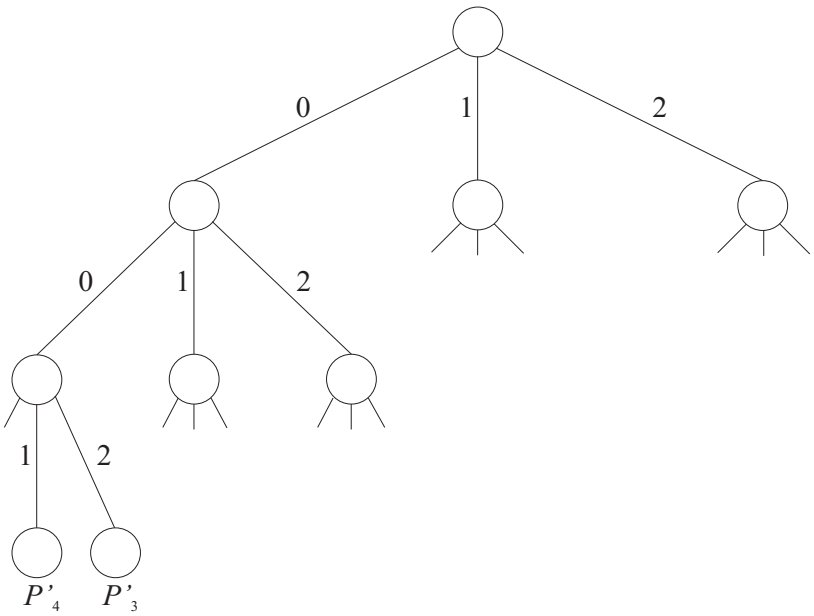



Figura 5.14: Árvore *hash* após a inserção dos *pth's*

todos os *pth's* com possibilidade de serem suportados pela tupla percorre-se o caminho informado pela função *hash* aplicada sobre cada elemento da seqüência de dados da tupla e o caminho zero. Assim, são recuperados todos os *pth's* cujas seqüências de dados possuem constantes e variáveis.

Somente após a construção de uma árvore *hash* para cada partição de um conjunto $C_k$ é que os *pth's* têm seu suporte calculado. No calculo do suporte somente uma árvore *hash* é analisada de cada vez, ou seja, é analisada uma partição de cada vez, possibilitando a execução de uma única consulta ao banco de dados para calcular o suporte de todos os *pth's* da partição, diminuindo assim o tempo gasto na etapa de avaliação.

### 5.1.4 Otimizações

O protótipo MILPRIT possui alguns problemas. Um deles é o fato de não realizar gerenciamento de memória, fazendo com que os padrões gerados ocupem rapidamente toda a memória principal disponível, o que faz com que o sistema aborte a execução logo nas primeiras iterações. Este é um problema que deve ser tratado em grande parte dos algoritmos de mineração de dados que utilizam a técnica *Apriori*, uma vez que, a cada iteração uma quantidade enorme de novos padrões são gerados. Outro problema do pro-

tótipo MILPRIT ocorre na etapa de geração, onde, determinados *pth's* não são gerados ou são gerados padrões que não são interessantes para o usuário. A seguir serão descritos os recursos que foram utilizados para solucionar estes problemas.

### Otimização da Ocupação da Memória Principal

Primeiramente, observa-se como é a relação entre as etapas de geração e poda do protótipo MILPRIT, etapas essas responsáveis diretas pela ocupação e liberação da memória principal.

#### MILPRIT: Execução das Etapas de Geração e Poda dos Padrões

Em uma iteração $k$ são gerados todos os possíveis padrões a partir dos padrões freqüentes do conjunto $F_{k-1}$, obtido na iteração anterior. Os novos padrões são inseridos diretamente no conjunto $C_k$. Somente após a geração de todos os possíveis padrões a partir de $F_{k-1}$ é que o conjunto $C_k$ é submetido a etapa de poda, onde grande parte dos padrões são eliminados, tendo como conseqüência uma considerável liberação da memória principal.

Ao acompanhar detalhadamente a execução do protótipo MILPRIT, um importante fato foi constatado: *a memória principal é completamente ocupada sempre na etapa de geração, não possibilitando sequer que um único padrão seja podado.*

#### O que fazer?

Em vista da constatação anterior surgiram dois caminhos dos quais foi escolhido apenas um para solucionar o problema da ocupação da memória. O primeiro caminho consiste em definir um limite de memória principal que pode ser ocupada pelos padrões gerados, se esse limite for ultrapassado, os padrões devem ser armazenados em disco. Após a geração, os padrões gerados devem ser recuperados do disco em quantidade limitada pela memória principal disponível para, ai sim, serem submetidos à etapa da poda. Este procedimento deve ser realizado até que todos os padrões gerados fossem recuperados do disco.

Esta idéia com certeza soluciona o problema da ocupação da memória principal, más leva a um outro problema: o tempo de execução aumenta, uma vez que o acesso a disco é mais lento que o acesso a memória principal. Assim, este caminho foi descartado, pois o

processo de mineração de *pth's* exige um tempo considerável e não seria viável considerar a hipótese de aumentar ainda mais este tempo.

Portanto, foi escolhido o segundo caminho, que consiste em intercalar as etapas de geração e poda e continuar trabalhando com todos os *pth's* na memória principal. Embora a escolha tenha sido por este caminho, é importante resaltar que ele também possui um problema. Como todos os *pth's* são mantidos na memória principal, se o MILPRIT* for executado sobre condições extremas como por exemplo domínios de dados muito grandes, expressões regulares muito pouco seletivas e nível mínimo de suporte muito baixo (próximo de zero), vai chegar um momento em que os *pth's* vão ocupar toda a memória principal.

Mesmo com este problema em potencial, a escolha por este caminho foi baseada em outro fato importante constatado durante exaustivos testes do protótipo MILPRIT: *o número de pth's podados é sempre muito elevado em relação ao número de pth's gerados.*

Após implementado, o algoritmo MILPRIT* foi testado em condições extremas e mesmo assim nunca foram atingidos grandes níveis de consumo da memória principal, pois ele gera um certa quantidade de *pth's* e imediatamente poda grande parte destes *pth's*, liberando a memória.

O modo como esta técnica foi implementada foi mostrada na visão geral do algoritmo MILPRIT* apresentado na figura 5.2. O laço responsável pela intercalação das etapas de geração e poda vai da linha 6 a 11. Detalhes desta técnica serão descritos a seguir.

### MILPRIT*: Execução das Etapas de Geração e Poda dos Padrões

Em uma iteração $k$ são gerados, atravéz da função $Gen(F_{k-1}[i], \mathcal{R})$, para cada *pth* $i$ de $F_{k-1}$, uma quantidade de *pth's* que depende diretamente do tamanho do domínio dos dados (instanciação) e do tipo da variável temporal do novo átomo de dados (extensão). Por exemplo, considere um conjunto $F_1$ com 10000 *pth's*, tal que, $P = (Paciente(x), \{Hormônio(x, y, e_1)\}, \{\emptyset\})$ é o primeiro *pth* de $F_1$. Considere também uma expressão regular $\mathcal{R} = Hormônio(x, \$, \#)Fumo(x, \$, \#)^*Birads(x, \$, \#)$ e um domínio de dados com 1000 constantes. Quando $P$ é submetido a $Gen(F_1[1], \mathcal{R})$ são gerados 1000 *pth's* $P'$ pela instanciação de $P$. De acordo com $\mathcal{R}$, $P$ pode ser estendido atravéz dos átomos de dados $Fumo(x, z, f_2)$ e $Birads(x, h, s_2)$. Para o átomo $Fumo(x, z, f_2)$ são gerados

6 *pth's* $P'$ pela extensão de $P$. Para o átomo $Birads(x, w, f_2)$ é gerado 1 *pth* $P'$ pela estensão de $P$. Portanto, $Gen(F_1[1], \mathcal{R})$ retorna 1007 *pth's* especializados a partir de $P$. Diferentemente do protótipo MILPRIT que especializa os 10000 *pth's* de $F_1$ para depois submeter os *pth's* especializados à etapa da poda, no MILPRIT* os 1007 *pth's* gerados a partir de $P$ são imediatamente submetidos a etapa da poda, onde a maioria deles são eliminados, resultando na liberação da memória principal. Este procedimento é realizado para cada *pth* de $F_1$. Observe que como cada *pth* de $F_1$ resulta em um conjunto de *pth's* que é imediatamente reduzido na etapa da poda, o nível de ocupação da memória se mantem sempre baixo durante todo o processo de mineração.

### Otimização da Etapa de Geração dos Padrões

O motivo pelo qual o protótipo MILPRIT minera determinados padrões que não interessam ao usuário ou não minera determinados padrões está diretamente relacionado a etapa de geração. Este problema será apresentado por meio de um exemplo.

Considere o *pth* $P = (Paciente(x), \{ Hormônio(x, var1, e_1) \}, \{ \emptyset \})$, o domínio $dom(2, Hormônio) = \{estradiol\}$, a expressão regular $\mathcal{R} = Hormônio(x, var1, \#)^* Fumo(x, \$, \#)$ e o alfabeto $\Sigma = \{w_1 = Hormônio(x, \$, \#), w_2 = Hormônio(x, var1, \#), w_3 = Fumo(x, \$, \#)\}$.

Para especializar $P$, o protótipo MILPRIT chama uma função de instanciação que primeiramente encontra as variáveis de $P$ que serão instanciadas. Relembrando, um padrão só é instanciado se não existe nenhuma variável já instanciada a direita da variável que vai ser instanciada e se a entrada do modo que gerou o átomo ao qual a variável que vai ser instanciada pertence for \$ (definição 4.1.1). É justamente neste segundo caso que se encontra um dos problemas da etapa de geração do MILPRIT. Para encontrar o modo que gerou um determinado átomo, à função de instanciação compara somente o predicado do átomo com o predicado de cada modo do alfabeto de modos, até encontrar o primeiro predicado de um modo que é igual o predicado do átomo que está sendo analisado. Quando isso acontece, a função encerra a busca e considera que o átomo é de acordo com o modo. O problema é que, quando o alfabeto tem mais de um modo com mesmo predicado, a função pode gerar um resultado incorreto. Neste

exemplo, a função de instanciação do protótipo MILPRIT considera (incorretamente) que a variável $var1$ pode ser instanciada, pois o primeiro modo com predicado igual ao do átomo $Hormônio(x, var1, e_1)$ encontrado pela função é o $Hormônio(x, \$, \#)$. Assim, como o modo tem um símbolo \$ na mesma posição de $var1$, $P'$ é instanciado resultando no padrão $P'_1 = (Paciente(x),\{Hormônio(x, estradiol, e_1)\},\{\emptyset\})$.

Observe na expressão regular que o usuário deseja obter padrões onde pacientes fizeram o uso de qualquer hormônio, não um hormônio específico como o *estradiol*. Portanto, o modo correto que deveria ser retornado pela função de instanciação é o $Hormônio(x, var1, \#)$, que faria com que $P$ não fosse instanciado, pois não existe um símbolo \$ na mesma posição de $y$.

Após a instanciação, como $P$ não contém constantes, o protótipo MILPRIT chama a função de extensão, que insere um novo átomo de dados no fim da seqüência de dados de $P$. Para isso, primeiramente é chamada a função que retorna a palavra associada a $P$ possibilitando descobrir através da expressão regular $\mathcal{R}$ qual átomo pode ser inserido em $P$. Outro problema da etapa de geração do protótipo MILPRIT está nesta função. Para encontrar a palavra associada a um determinado padrão, ele compara somente os predicados de cada átomo do padrão com os predicados de cada modo do alfabeto, de maneira similar à função de instanciação. Assim, uma palavra errada pode ser retornada. Para o *pth* $P$ a função retorna (incorretamente) a palavra $W = w_1 = Hormônio(x, \$, \#)$. Isto faz com que $P$ não seja estendido, pois o modo $Hormônio(x, \$, \#)$ não é considerado em $\mathcal{R}$.

A palavra correta associada ao padrão $P$ que deveria ser retornada é $W = w_2 = Hormônio(x, var1, \#)$. Assim, de acordo com a expressão regular $\mathcal{R}$, $P$ pode ser estendido pela inclusão dos átomos $Hormônio(x, y, e_2)$ e $Fumo(x, z, f_2)$ resultando nos seguintes padrões:

$P'_2 = (Paciente(x),\{Hormônio(x, estradiol, e_1),\ Hormônio(x, y, e_2)\},\{\mathcal{T}'_2\})$

$P'_3 = (Paciente(x),\{Hormônio(x, estradiol, e_1),\ Fumo(x, z, f_2)\},\{\mathcal{T}'_3\})$

No algoritmo MILPRIT*, a função de instanciação, chamada $Instantiation(\ )$, e a que encontra a palavra associada a um determinado *pth*, chamada $Match(\ )$, foram implementadas de modo que sempre que elas forem verificar se um átomo é de acordo com

um modo considerem, além dos predicados, todas as entradas, como definido na seção 4.1.1 do capítulo anterior. Assim, o problema da etapa de geração foi solucionado no algoritmo MILPRIT*.

## 5.2 Considerações Finais do Capítulo

Neste capítulo o algoritmo MILPRIT* foi apresentado. Foram detalhadas e explicadas com exemplos as principais funções utilizadas na etapa de geração dos *pth's*, como por exemplo as funções de especialização por instanciação e por extensão. Na etapa de avaliação foram apresentadas todas as estruturas e condições necessárias para verificar se uma determinada tupla suporta um determinado *pth*, condição fundamental para o calculo do suporte.

Ainda na etapa de avaliação, foram apresentadas algumas técnicas, como por exemplo o particionamento de um conjunto $C_k$ de *pth's* candidatos e a inserção dos *pth's* de cada partição em uma árvore *hash*, desenvolvidas e implementadas com o objetivo agilizar o calculo do suporte.

Finalizando o capítulo, foram apresentados os problemas existentes no protótipo MILPRIT e que foram solucionados no algoritmo MILPRIT*. O primeiro deles, a ocupação da memória principal, foi solucionado mantendo sempre com todos os *pth's* na memória principal, por motivos de performance, mas intercalando as etapas de geração e poda, fazendo com que grande parte da memória ocupada após a especialização de cada *pth* seja imediatamente liberada. O problema da etapa de geração foi solucionado criando conceitos para verificar de forma correta se um átomo é de acordo com um modo. Estes conceitos foram implementados na função de instanciação e na função que encontra a palavra associada a um determinado *pth*.

# Capítulo 6

# Resultados Experimentais

Neste capítulo serão apresentados os resultados de vários experimentos realizados com o MILPRIT* em bancos de dados sintéticos, com o intuito de analisar sua performance e escalabilidade. Serão apresentados também resultados de experimentos realizados em um banco de dados real.

Os experimentos com o algoritmo MILPRIT* foram realizados em um Pentium Core 2 Duo de 2.4 GHz com 4 GB de memória RAM executando o sistema operacional Linux.

## 6.1 Dados Sintéticos

Foi desenvolvido um gerador de dados sintéticos que produz bancos de dados temporais, onde *pth's* que satisfazem expressões regulares $\mathcal{R}$, também produzidas pelo gerador, podem aparecer. Os bancos de dados e as expressões regulares são produzidas de acordo com alguns parâmetros de entrada que são mostrados na tabela 6.1.

Nos testes apresentados neste capítulo, foi usada a seguinte notação para os bancos de dados gerados: $I$i-$P$p-$M$m-$U$u-$B$b-$N$n-$V$v-$Q$q, onde, i, p, m, u, n, v e q são os valores definidos para os parâmetros de entrada $I, P, M, U, B, N, V$ e $Q$, respectivamente. Os parâmetros de entrada $S, D$ e $T$, são definidos como: $S = \frac{U}{2.5}$, $D = \frac{U}{5}$ e $T = \frac{U}{5}$.

As expressões regulares $\mathcal{R}$ produzidas pelo gerador tem o formato $(B_1^* \ldots B_k^*)$, onde cada $B_i^*$ é um *bloco* com o formato $(N_1 \& \ldots \& N_l)$ e cada $N_j$ é um *termo* com o formato $(m_1^j | \ldots | m_h^j)$ e $m_g^j$ são modos. Assim, o gerador produz uma grande variedade de

| Parâmetros | |
|---|---|
| $I$ | Número de tabelas com intervalos |
| $P$ | Número de tabelas com pontos |
| $M$ | Número de atributos por tabela (não temporal) |
| $U$ | Número de tuplas por tabela - em '000s |
| $S$ | Tamanho do domínio dos atributos de referência - em '000s |
| $D$ | Tamanho do domínio dos atributos que não são de referência - em '000s |
| $T$ | Tamanho do domínio dos atributos temporais - em '000s |
| $B$ | Número de blocos da expressão regular |
| $N$ | Número máximo de termos por bloco da expressão regular |
| $V$ | Número máximo de modos por termo da expressão regular |
| $Q$ | Número de *pth's p-válidos* - em '000s |

Tabela 6.1: Parâmetros usados pelo Gerador de Dados Sintéticos

expressões regulares quando são variados o número de blocos ($b$), o número máximo de termos em cada bloco ($n$) e o número máximo de modos em cada termo ($v$). Por exemplo, uma expressão regular $\mathcal{R}$ que poderia ser gerada pelo gerador de dados sintéticos é $\mathcal{R} = ((Hormônio(x, \$, \#) \mid Fumo(x, \$, \#)) \ \& \ (Fumo(x, \$, \#) \mid Birads(x, \$, \#))^*$. Essa expressão regular tem um bloco, dois termos, e cada termo tem dois modos.

A configuração dos bancos de dados utilizados nos testes é mostrada na tabela 6.2.

| Nome | $I$ | $P$ | $M$ | $U$ | $B$ | $N$ | $V$ | $Q$ |
|---|---|---|---|---|---|---|---|---|
| $I2$-$P2$-$M2$-$U5$-$B2$-$N2$-$V1$-$Q1$ | 2 | 2 | 2 | 5 | 2 | 2 | 1 | 1 |
| $I2$-$P2$-$M2$-$U5$-$B$b-$N2$-$V1$-$Q1$ | 2 | 2 | 2 | 5 | 1-5 | 2 | 1 | 1 |
| $I2$-$P2$-$M2$-$U5$-$B2$-$N$n-$V1$-$Q1$ | 2 | 2 | 2 | 5 | 2 | 1-5 | 1 | 1 |
| $I2$-$P2$-$M2$-$U5$-$B2$-$N2$-$V$v-$Q1$ | 2 | 2 | 2 | 5 | 2 | 2 | 1-5 | 1 |
| $I2$-$P2$-$M2$-$U$u-$B2$-$N2$-$V1$-$Q1$ | 2 | 2 | 2 | 2-10 | 2 | 2 | 1 | 1 |
| $I2$-$P2$-$M2$-$U5$-$B2$-$N2$-$V1$-$Q$q | 2 | 2 | 2 | 5 | 2 | 2 | 1 | 1-4 |
| $I2$-$P2$-$M$m-$U5$-$B2$-$N2$-$V1$-$Q1$ | 2 | 2 | 2-7 | 5 | 2 | 2 | 1 | 1 |

Tabela 6.2: Bancos de Dados Sintéticos

O banco de dados apresentado na primeira linha da tabela 6.2 ($I2$-$P2$-$M2$-$U5$-$B2$-$N2$-$V1$-$Q1$) contém 2 tabelas cujo atributo temporal é intervalar, 2 tabelas cujo atributo temporal é pontual, cada uma das quatro tabelas possui 2 atributos não temporais, 5000 tuplas e foram inseridos 1000 *pth's p-válidos*. A expressão regular gerada possui 2 blocos com 2 termos em cada bloco e 1 modo em cada termo. Na segunda linha da tabela 6.2 a cofiguração $I2$-$P2$-$M2$-$U5$-$B$b-$N2$-$V1$-$Q1$ representa vários bancos de dados, onde apenas

o parâmetro *número de blocos* ($B$) varia de um banco de dados para outro.

### 6.1.1 Análise de Performance

A figura 6.1 mostra o tempo de execução do algoritmo MILPRIT* para os quatro primeiros bancos de dados da tabela 6.2.

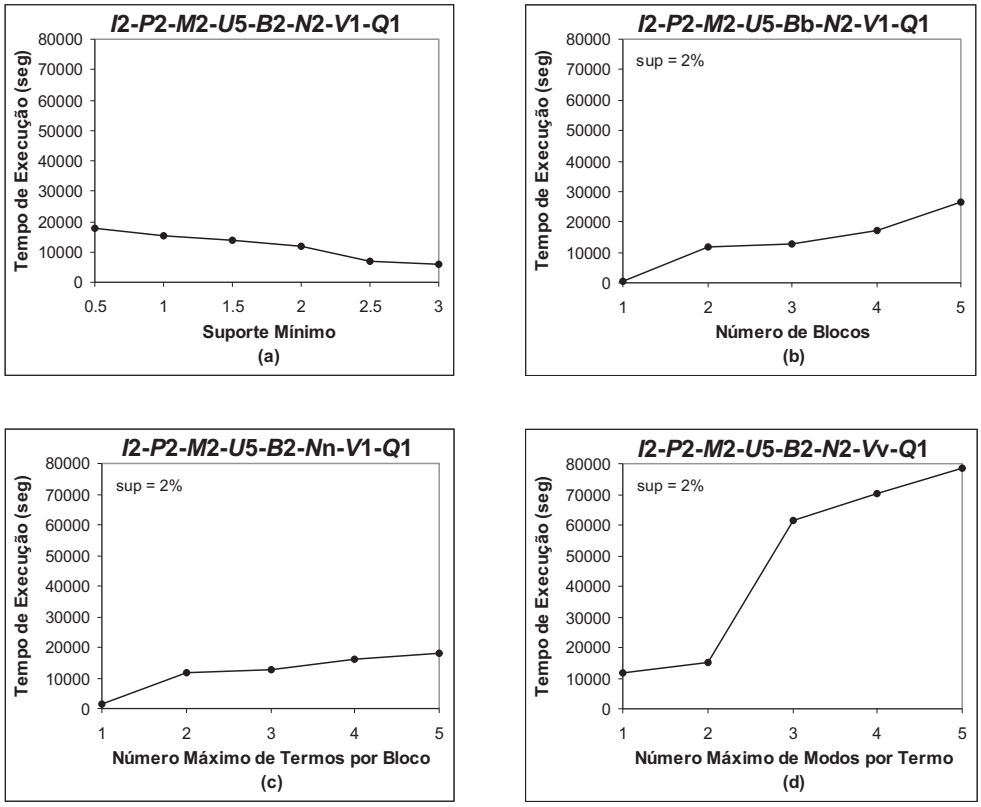



Figura 6.1: Análise de performance variando alguns parâmetros

A figura 6.1(a) mostra a performance do algoritmo à medida que o suporte mínimo é incrementado de 0.5% a 3%. Como esperado, o tempo de execução do algoritmo diminui, pois poucos candidatos são potencialmente freqüentes para altos valores de suporte mínimo, e portanto, quanto menor for o suporte, mais padrões são encontrados.

As figuras 6.1(b), 6.1(c) e 6.1(d) mostram a performance do algoritmo enquanto os parâmetros da expressão regular variam. Nestes testes o suporte mínimo foi mantido em 2%. A figura 6.1(b) ilustra o tempo de execução à medida que o número de blocos na expressão regular ($b$) é incrementado de 1 a 5. Quando o número de blocos aumenta,

a seletividade da restrição diminui, em conseqüência disso, o número de *pth's p-válidos* aumenta, resultando no aumento do tempo de execução do MILPRIT*. A figura 6.1(c) ilustra o tempo de execução à medida que o número máximo de termos por bloco na expressão regular ($n$) é incrementado de 1 a 5. Pode ser observado que o tempo de execução aumenta. Isto ocorre porque quando a expressão regular tem poucos termos por bloco, os *pth's* que a satisfaz tem comprimento pequeno. Conseqüentemente, os *pth's* são encontrados pelo MILPRIT* em níveis (iterações) baixos, entretanto, quando a expressão regular tem muitos termos por bloco os *pth's* que a satisfaz tem comprimento grande. Conseqüentemente, os *pth's* são encontrados pelo MILPRT* em níveis (iterações) mais elevados, resultando no aumento do tempo de execução. A figura 6.1(d) mostra a performance do algoritmo à medida que o número máximo de modos por termo na expressão regular ($v$) é incrementado de 1 a 5. Neste caso, a análise é similar à realizada para a figura 6.1(b) (variação do número de blocos), pois quando o número máximo de modos por termo aumenta também há uma diminuição da seletividade da restrição.

A figura 6.2 ilustra a relação entre a quantidade de *pth's* gerados e podados durante a execução do MILPRIT* sobre o banco de dados $I2$-$P2$-$M2$-$U5$-$B2$-$N2$-$V1$-$Q1$. As figuras 6.2(a) e 6.2(b) mostram a quantidade de *pth's* que foram gerados e podados quando o MILPRIT* foi executado com suporte de 0.5% e 2%, respectivamente. Note que nos dois casos a quantidade de *pth's* podados é muito elevada. Com isso, uma quantidade pequena de *pth's* em relação a quantidade de *pth's* gerados é mantida na memória principal e levada para a etapa de avaliação. Este comportamento foi observado em todos os testes realizados.

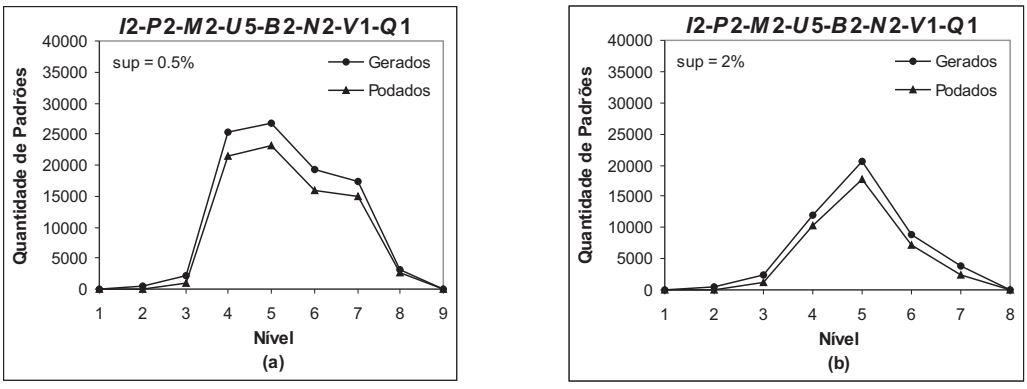



Figura 6.2: Relação entre a quantidade de *pth's* gerados e podados

Os resultados dos testes apresentados na figura 6.3 foram realizados com o objetivo de mostrar o tempo que o MILPRIT* gasta na execução de cada etapa em cada nível. Os testes foram realizados sobre o banco de dados $I2$-$P2$-$M2$-$U5$-$B2$-$N2$-$V1$-$Q1$ com suporte de 0.5% (figura 6.3(a)) e 2% (figura 6.3(b)). O tempo gasto na etapa de avaliação dos dois casos, principalmente nos níveis (iterações)mais elevados, é extremamente alto em relação ao tempo gasto na etapa de geração/poda. Isso acontece porque nos níveis (iterações) mais elevados é calculado o suporte de uma grande quantidade de padrões extensos (com grande comprimento), resultando na execução de grandes expressões SQL's sobre o banco de dados. Neste caso fica claro o motivo de tanto esforço para otimizar a etapa de avaliação.

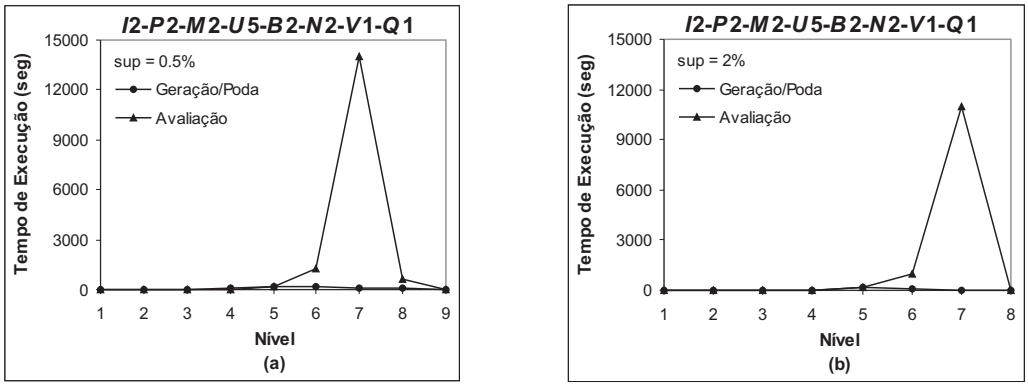



Figura 6.3: Tempo de execução de cada etapa em cada nível

### 6.1.2 Análise de Escalabilidade

Nesta seção serão apresentados os resultados dos principais experimentos realizados para análise da escalabilidade do algoritmo MILPRIT*. O suporte mínimo utilizado para estes testes foi fixado em 2%.

Na figura 6.4(a) é mostrado como o MILPRIT* se comportou quando foi avaliado sobre o quinto banco de dados da tabela 6.2 ($I2$-$P2$-$M2$-$U$u-$B2$-$N2$-$V1$-$Q1$), quando o número de tuplas ($u$) foi incrementado de 20000 a 100000. Claramente, quanto maior o número de tuplas nas tabelas do banco de dados maior é o tempo de execução do algoritmo. Isso ocorre porque a fase de avaliação, como mostrado na figura 6.3, é a responsável por grande parte do tempo de execução, assim, quanto mais tuplas mais tempo o MILPRIT* gasta no cálculo do suporte dos *pth's*.

A figura 6.4(b) mostra o tempo de execução quando o MILPRIT* foi executado sobre o sexto banco de dados da tabela 6.2 ($I2$-$P2$-$M2$-$U5$-$B2$-$N2$-$V1$-$Q$q). Observa-se que conforme a quantidade de *pth's* freqüentes ($q$) foi incrementada de 1000 a 4000 no banco de dados, o tempo de execução do algoritmo aumentou.

A figura 6.4(c) mostra como o MILPRIT* se comportou quando foi executado sobre o sétimo banco de dados da tabela 6.2 ($I2$-$P2$-$M$m-$U5$-$B2$-$N2$-$V1$-$Q1$) e o número de atributos não temporais em cada tabela ($m$) foi incrementado de 2 a 6. Quando o número de atributos não temporais aumenta, o número de elementos que fazem parte das seqüências de dados dos *pth's* também aumenta. Conseqüentemente o número de atributos escaneados durante a fase de avaliação aumenta, resultando no aumento do tempo de execução.

Note que nos três testes mostrados nas figuras 6.4(a), 6.4(b) e 6.4(c), o tempo de execução do MILPRIT* aumentou linearmente quando os parâmetros correspondentes (número de tuplas, *pth's* freqüentes e atributos não temporais) foram incrementados.

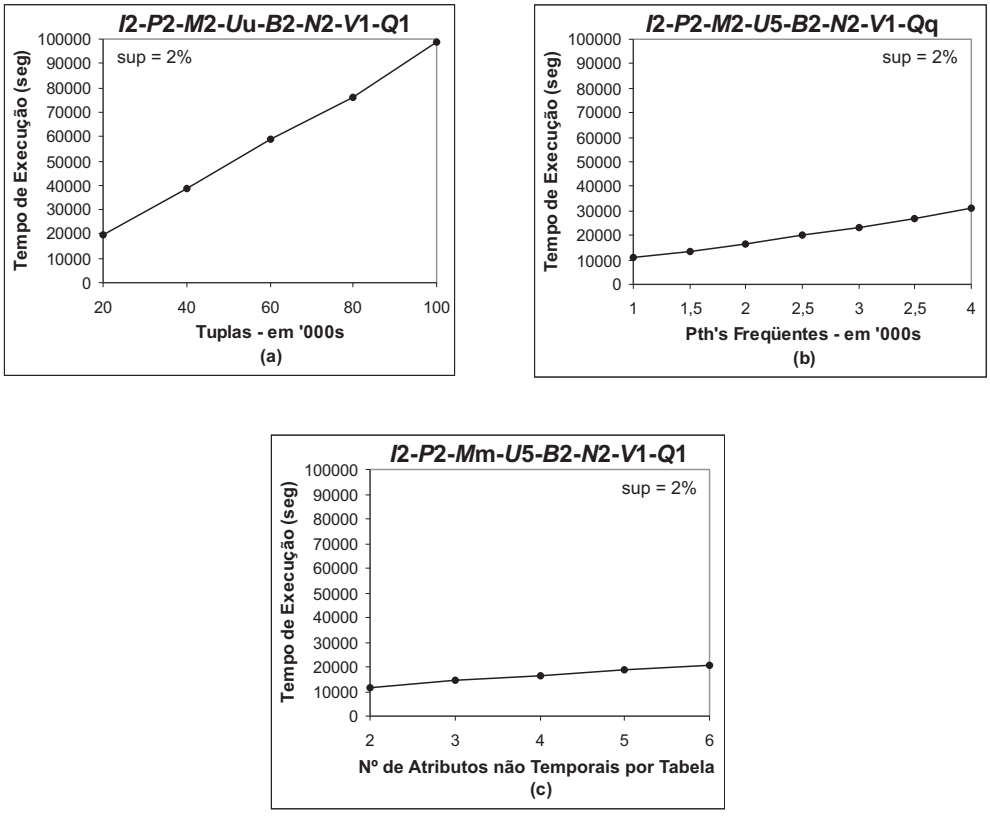



Figura 6.4: Análise de escalabilidade variando alguns parâmetros

## 6.2 Dados Reais

Em nosso laboratório está em desenvolvimento um projeto, chamado MIMED, que integra um sistema de mineração de padrões temporais, chamado MILPRIT*, e um atlas indexado de mamografias digitais, chamado AMDI[1] [43]. Esses sistemas são designados para serem usados por radiologistas e estudantes de radiologia, com o intuído de auxiliá-los na prevensão e no diagnóstico de câncer de mama.

Para a realização dos testes em dados reais foi utilizado um banco de dados[2] cuja modelagem foi baseada na classificação Bi-Rads, padrão publicado pela ACR (Colégio Americano de Radiologia) [18]. A classificação Bi-Rads tem como objetivo padronizar os laudos mamográficos levando em consideração a evolução do diagnóstico e recomendar uma conduta. Esta padronização evita laudos extensos, sem objetividade, e não conclusivos, além de facilitar a descrição de eventuais lesões [18].

O banco de dados real é um banco de dados relacional gerenciado pelo Sistema de Gerenciamento de Banco de Dados POTGRESQL e contém informações sobre pacientes do sexo feminino, como mamografias digitais; estilo de vida e hábitos: por exemplo, período de tempo em que fumou, ingeriu bebida alcoólica, etc; histórico clínico: por exemplo, período de tempo em que usou medicamentos como hormônios, antidepressivos, anticoncepcionais, data da primeira gravidez, data da menopausa, resultados dos exames mamográficos baseados na classificação Bi-Rads, etc. A classificação Bi-Rads divide os laudos em sete categorias, quanto maior for a categoria maiores são as chances da paciente ter câncer de mama. A tabela 6.3 mostra como as categorias são divididas.

Os dados registrados no banco de dados real foram obtidos junto ao Instituto Victório Valeri de Diagnósticos Médicos de Ribeirão Preto - SP. Esses dados foram coletados durante um período de três meses e totalizam 1365 casos.

[1] O sistema *AMDI* está disponível em www.lcc.ufu.br/amdi.

[2] O banco de dados utilizado neste trabalho está disponível em www.lsi.ufu.br.

| Categorias | Avaliação | Conduta |
|---|---|---|
| 0 | Incompleta | Outras incidências de mamografia são necessárias |
| 1 | Negativa (nada encontrado) | Rastreamento normal |
| 2 | Achados benignos (calcificações) | Rastreamento normal |
| 3 | Provavelmente benignos | Segmento 6 meses (às vezes indica-se biópsia) |
| 4 | Anomalias suspeitas<br>**A** - menor suspeita<br>**B** - média suspeita<br>**C** - maior suspeita | Biópsia deve ser avaliada |
| 5 | Alta suspeita de malignidade | Necessita esclarecimento definitivo |
| 6 | Já existe diagnóstico do câncer | |

Tabela 6.3: Categorias definidas pela classificação Bi-Rads [18]

O esquema do banco de dados real é o seguinte:

$$\mathbf{R} = \{Paciente(NomePac),\ Anticoncepcional(NomePac, NomeAntConcep, T_{it}),$$
$$Antidepressivo(NomePac, NomeAntDep, T_{it}),\ Hormônio(NomePac, NomeHorm, T_{it}),$$
$$Fumo(NomePac, Qtde, T_{it})\ Bebida(NomePac, Tipo, T_{it})$$
$$PrimeiraMenstruacao(NomePac, Status, T_{pt}),\ PrimeiraGravidez(NomePac, Status, T_{pt}),$$
$$Menopausa(NomePac, Menopausa, T_{it}),\ Birads(NomePac,\ Categoria,\ T_{pt})\}$$

Nos testes foi utilizada uma expressão regular que permitiu gerar qualquer configuração de *pth's*, restringindo o átomo *Birad* a aparecer somente no final, após qualquer outro átomo.

Inicialmente foi utilizada a granularidade em dias, o que levou a constatação de que, para esta aplicação, este tipo de granularidade resulta na mineração de *pth's* pobres em informações temporais. A maioria dos *pth's* minerados continham apenas o predicado temporal $before$.

Com o objetivo de minerar *pth's* mais conclusivos foram realizados testes considerando também a granularidade em semestres, que resultou na mineração de *pth's* com uma maior variedade de predicados temporais. Alguns *pth's* que foram encontrados no banco de dados real considerando a granularidade em semestres são:

$$P_1 = (Paciente(x), \{Menopausa(x, SIM, e_1), Antidepressivo(x, FLUOXETINA, f_2)\},$$
$$\{starts(e_1, f_2)\})$$

$$P_2 = (Paciente(x), \{Primeiragravidez(x, SIM, s_1), Fumo(x, 20, f_2)\}, \{during(s_1, f_2)\})$$

$$P_3 = (Paciente(x), \{Fumo(x, 20, e_1), Birads(x, 2, s_2)\}, \{before(e_1, s_2)\})$$

$$P_4 = (Paciente(x), \{Primeiragravidez(x, SIM, e_1), Menopausa(x, SIM, f_2), Birads(x, 2, s_3)\}, \{before(e_1, f_2), before(e_1, s_3), before(f_2, s_3)\})$$

O *pth* $P_1$ foi verificado em 25 pacientes do banco de dados e significa que pacientes começaram a tomar o antidepressivo $FLUOXETINA$ no momento em que entraram na menopausa. O *pth* $P_2$ foi verificado em 50 pacientes e significa que pacientes tiveram filhos durante o tempo em que fumavam (ou ainda fumam) 20 cigarros por dia. O *pth* $P_3$ foi verificado em 14 pacientes e significa que pacientes que fumavam 20 cigarros por dia, apresentaram Bi-Rads categoria 2 no futuro. O *pth* $P_4$ foi verificado em 177 pacientes e significa que pacientes que já tiveram filhos apresentaram Bi-Rads categoria 2 após entrarem na menopausa.

Pelo fato do banco de dados real sobre o qual foram realizados os testes ser relativamente pequeno, não foram minerados *pth's* bastante expressivos e conclusivos. Apesar do grande trabalho despendido na coleta das informações e na construção do banco de dados real, não foi reunida uma grande variedade de informações, fundamental para que um banco de dados voltado para este tipo de aplicação seja confiável e consistente, possibilitando a mineração de *pth's* com mais informações relevantes.

Futuramente, quando o sistema AMDI estiver concluído, poderão ser realizados novos testes mais elaborados, tendo em vista que este sistema permite que radiologistas e estudantes de radiologia insiram ilimitadamente novos casos em seu banco de dados. Assim, é esperado que seja obtido um banco de dados maior, mais confiável e com uma grande variedade de informações, possibilitando a mineração de *pth's* mais conclusivos.

# Capítulo 7

# Conclusão e Trabalhos Futuros

A mineração de padrões temporais freqüentes em grandes bancos de dados constitui um problema importante em mineração de dados. A distinção existente entre os padrões temporais pontuais proposicionais ou de primeira ordem, conhecidos também como padrões seqüenciais proposicionais ou de primeira ordem, e os padrões temporais intervalares proposicionais ou de primeira ordem fazem com que fatos importantes não sejam expressos. Nesta dissertação foi proposto um novo padrão temporal chamado *padrão temporal híbrido* (*pth*) e foi mostrado que ele necessita da expressividade da Lógica Temporal de Primeira Ordem para ser especificado.

**Lógica Temporal de Intervalos - LTI**. Foi apresentada a LTI criada por Allen, que é uma lógica temporal de primeira ordem, que foi adaptada e utilizada para especificar os *pth's*. A base da LTI considera períodos de tempo, relacionamentos entre estados e seus efeitos. Como os períodos de tempo são representados na LTI original em termos de *intervalos* (períodos com duração) e *pontos* não são considerados períodos, a LTI foi adaptada de forma que *pontos* são considerados períodos sem duração. Portanto, como *intervalos* e *pontos* são considerados períodos, podem ser aplicados relacionamentos entre eles.

O problema de mineração dos *pth's com* restrições de expressões regulares também foi explorado nesta pesquisa.

**Mineração de *pth's* com restrições de expressões regulares**. Foram desenvolvidas técnicas para a mineração de *pth's* freqüentes em um banco de dados $BD$ com relação a

um suporte mínimo e a uma expressão regular definida pelo usuário. O objetivo do uso das restrições de expressões regulares é reduzir o número de *pth's* minerados e fornecer ao usuário um maior controle do processo de mineração, afim de que sejam encontrados somente os *pth's* que o interessam. Foi proposto então o algoritmo MILPRIT*, que minera *pth's* que satisfazem uma expressão regular.

As idéias principais do MILPRIT* são relembradas a seguir:

- **MILPRIT*** (***M****ining* ***I****nterval* ***L****ogic* ***P****atterns with* ***R****egular expressIons* cons***T****raints*). Este algoritmo realiza a mineração dos *pth's* nível por nível incorporando restrições de expressões regulares (*não-antimonotônicas*) nas etapas de geração e poda dos *pth's* candidatos. Ele intercala as etapas de geração e poda com o objetivo de controlar a ocupação da memória principal. Na etapa de geração, ele emprega operadores de especialização, garantindo que sejam gerados todos os *pth's* possíveis e evitando que sejam gerados *pth's* duplicados. Na etapa de avaliação, ele utiliza algumas técnicas, como por exemplo a divisão dos *pth's* em partições e uma árvore *hash* para torná-la mais eficiente.

Foram realizados vários experimentos em dados sintéticos para avaliar a performance e a escalabilidade do algoritmo MILPRIT*. Foram realizados também experimentos em dados reais. Os resultados obtidos nos dados sintéticos confirmam a viabilidade de, além de se usar um suporte mínimo, também incorporar restrições de expressões regulares durante o processo de mineração.

### Trabalhos Futuros

Com o objetivo de orientar estudos futuros, algumas possíveis extensões e otimizações deste trabalho podem ser destacadas. Entre elas:

- **Considerar o predicado temporal *equals***. Para simplificar a representação e facilitar a exploração da essência do problema de mineração padrões temporais híbridos, a formulação proposta neste trabalho não inclui padrões temporais onde *vários* átomos de dados são relacionados a uma mesma variável temporal. A inclusão

do predicado temporal *equals* permitiria essa nova representação, mas para isso novas teorias devem ser formuladas.

- **Testes em bancos de dados reais maiores**. Como especialistas afirmam e os testes em dados reais demonstraram, no caso de bancos de dados mamográficos, para se obter informações conclusivas é necessário uma grande quantidade de casos. A construção de um grande banco de dados mamográfico poderia resultar na descoberta de interessantíssimos *pth's* pelo algoritmo MILPRIT*.

- **Comparação entre o armazenamento dos *pth's* na memória principal e no disco**. Todos os *pth's* gerados são armazenados na memória principal e as etapas de geração e poda são intercaladas. Embora o problema da ocupação da memória principal tenha sido suprido por esta abordagem, ela é um problema em potencial. Testes poderiam ser feitos para verificar a viabilidade de se armazenar os *pth's* em disco. Assim, o problema da ocupação da memória principal estaria definitivamente resolvido.

- **Desenvolvimento de um novo algoritmo para mineração de *pth's* baseado na técnica do algoritmo *PrefixSpan***. Existem diversos algoritmos de mineração de padrões temporais e diferentes técnicas nas quais esses algoritmos se baseiam. Os algoritmos AprioriAll e GSP são exemplos de algoritmos de mineração de padrões seqüenciais baseados na técnica *Apriori*, assim como o MILPRIT* é um algoritmo de mineração de padrões temporais híbridos também baseado na técnica *Apriori*.

  Como mencionado nesta dissertação, o GSP apresenta melhor performance que o AprioriAll. Recentemente, foram desenvolvidos outros algoritmos para mineração de padrões seqüenciais que se mostraram mais rápidos que o GSP. Entre eles, podem ser destacados o algoritmo SPADE [10] e o PrefixSpan [31]. Testes mostraram que este último tem desempenho melhor, se comparado com o GSP e SPADE.

  Diferentemente da técnica *Apriori*, a técnica empregada no algoritmo PrefixSpan procura melhorar a eficiência diminuindo o tamanho do banco de dados a ser percorrido e diminuindo o número de padrões candidatos gerados. O banco de dados

de seqüências é recursivamente projetado em bancos de dados menores e todos os padrões seqüenciais são estendidos levando em conta somente informações contidas nos bancos de dados projetados. Assim, são gerados somente padrões seqüenciais freqüentes, evitando o tratamento de padrões candidatos gerados aleatoriamente, sem vinculos com o banco de dados, como é o caso dos algoritmos baseados na técnica *Apriori*.

Um algoritmo que minera os mesmos tipos de padrões que o MILPRIT* poderia ser desenvolvido utilizando a técnica empregada no algoritmo PrefixSpan. Certamente este novo algoritmo será mais rápido que o MILPRIT*, uma vez que neste último a etapa computacionalmente mais cara é a de avaliação (percorrer o banco de dados para calcular o suporte dos *pth's*) e um dos objetivos da técnica empregada no PrefixSpan é justamente diminuir o tamanho do banco de dados que será percorrido.

# Referências Bibliográficas


[1] G. Piatetsky-Shapiro, U. Fayyad, and P. Smith. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[2] J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. Technical report, Journal of Logic and Computation, 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *11th International Conference on Data Engineering*, pages 3–14. IEEE Computer Society Press, 1995.

[4] M. Joshi, G. Karypis, and V. Kumar. A universal formulation of sequential patterns. Technical report, Department of Computer Science, University of Minnesota, Minneapolis, 1999.

[5] O. Heierman, G. Youngblood, and D. Cook. Mining temporal sequences to discover interesting patterns. *International Conference Knowledge Discovery and Data Mining*, 2004.

[6] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[7] D.A. Furtado S. de Amo. First-order temporal pattern mining with regular expression constraints. In *20th Brazilian Symposium on Databases*, 2005.

[8] F. Höppner and F. Klawonn. Finding informative rules in interval sequences. *Lecture Notes in Computer Science*, 2189:123–132, 2001.

[9] A. Lattner and O. Herzog. Unsupervised learning of sequential patterns. In *ICDM Workshop on Temporal Data Mining: Algorithms, Theory and Applications TDM*, Brighton, UK, 2004.

[10] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

[11] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *5th International Conference Extending Database Technology*, volume 1057, pages 3–17. Springer, 1996.

[12] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *The VLDB Journal*, pages 223–234, 1999.

[13] S. D. Lee and L. D. Raedt. Constraint based mining of first order sequences in seqlog (extended abstract), 2005.

[14] C. Masson and F. Jacquenet. Mining frequent logical sequences with SPIRIT-LoG. In *12th International Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 166–181. Springer, 2003.

[15] D. A. Furtado D. Laurent S. de Amo, A. Giacometti. An apriori-based approach for first-order temporal pattern mining. In *19th Brazilian Symposium on Databases*, 2004.

[16] S. A. Amo, A. Giacometti, and M. S. Santana. Milprit: Mining interval logic patterns with regular expression constraints. In *1st Workshop de Algoritmos de Mineração de Dados*, Uberlândia, Brasil, 2005.

[17] N. Jacobs and H.Blockeel. From shell logs to shell scripts. In *11th International Conference on Inductive Logic Programming*, pages 80–90, 2001.

[18] A. C. of Radiology. Breast imaging reporting and data system birads. In *American College of Radiology*, 2004.

[19] S. A. Amo, W. J. Pereira, A. Giacometti, and T. G. Clemente. Mining first-order temporal interval patterns with regular expression constraints. In *DaWaK*, pages 459–469, Resgensburg, Alemanha, 2007.

[20] S. A. Amo, A. Giacometti, and W. J. Pereira. Mining temporal relational patterns over databases with hybrid time domains. In *22th Simpósio Brasileiro de Banco de Dados*, pages 347–361, João Pessoa, Brasil, 2007.

[21] C. J. Matheus, P. K. Chan, and G. Piatetsky-Shapiro. Systems for knowledge discovery in databases. *IEEE Trans. On Knowledge And Data Engineering*, 5:903–913, 1993.

[22] R. Agrawal, T. Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, 1993.

[23] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *20th International Conference on Very Large Data Bases*, pages 144–155, Los Altos, USA, 1994.

[24] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36(1–2):105–139, 1999.

[25] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th Internation Conference Very Large Data Bases, VLDB*, pages 487–499, 1994.

[26] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *3rd Internation Conference Knowledge Discovery and Data Mining, KDD*, pages 67–73. AAAI Press, 1997.

[27] R. Srikant and R. Agrawal. Mining generalized association rules. *Future Generation Computer Systems*, 13(2–3):161–180, 1997.

[28] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2–3):131–163, 1997.

[29] A. Srivastava, E. H. Han, V. Kumar, and V. Singh. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery*, 3(3):237–261, 1999.

[30] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[31] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *17th International Conference on Data Engineering*, pages 215–224. IEEE Computer Society, 2001.

[32] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *6th ACM SIGKDD international Conference on Knowledge Discovery and Data Mining*, pages 355–359. ACM Press, 2000.

[33] J. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In *Conference ACM CIKM*, 2002.

[34] H. Albert-Lorineza an J. F. Boulicaut. ining frequente sequential patterns under regular expressions. In *nternational Conference on Data Mining*, 2003.

[35] C. Antunes and A. L. Oliveira. Constraint relaxations for discovering unknown sequential patterns. In *Knowledge Discovery in Inductive Databases: 3th International Workshop, KDID*, pages 11–32, 2004.

[36] J.-F.Boulicaut, L. De Raedt, and H. Mannila. Constraint-based mining and inductive databases. In *European Workshop on Inductive Databases*, volume 3848, 2006.

[37] S. D. Lee. Constrained mining patterns in large databases, 2005.

[38] J. F. Allen. Maintaining knowledge about temporal intervals. Technical report, Communications of the ACM, 1983.

[39] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics*. North-Holland Pub. Co./MIT Press, 1991.

[40] F. Höppner. Discovery of temporal patterns - learning rules about the qualitative behavior of time series. In *5tn European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 192–203, 2001.

[41] A. Lattner and O. Herzog. Mining temporal patterns from relational data. In *LWA*, pages 184–189, 2005.

[42] P. A Flach. The logic of learning: a brief introduction to inductive logic programming. In *CompulogNet Area Meeting on Computational Logic and Machine Learning*, pages 1–17. University of Manchester, 1998.

[43] D. Guliato, E. V. Melo, R. S. Bôaventura, and Rangaraj M. Rangayyan. AMDI - indexed atlas of digital mammogram that integrates case studies, e-learning, and research systems via the web. *In: Jasjit S. Duri; Rangayyan M. Rangayyan. (Eds.). Recent Advances in Breast Imaging, Mammography, and Computer-aided Diagnosis of Breast Cancer. Bellingham, WA USA: SPIE PRESS*, pages 535–562, 2006.

## APÊNDICE A

Para testar a fórmula que garante a consistência de um conjunto de átomos temporais com a inclusão do novo predicado temporal $r$, foram criadas tabelas distribuídas em oito casos. É importante que os tipos (intervalar ou pontual) das variáveis temporais sejam considerados, pois isso restringe o uso de alguns predicados, como descrito na adaptação da estrutura temporal da LTI. Observe que no primeiro caso, onde todas as variáveis temporais são intervalares, todos os predicados temporais de $\mathcal{P}_{temp}$ podem ser utilizados, o que não acontece nos outros sete casos.

A fórmula $\mathcal{T}'_{i,j}(\tau_i, \tau_j) \ \wedge \ \mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1}) \ \wedge \ r(\tau_i, \tau_{n+1})$ é válida somente se a seqüência $\mathcal{T}'_{i,j}(\tau_i, \tau_j) \ \mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1}) \ r(\tau_i, \tau_{n+1})$ for igual a alguma linha de alguma tabela.

**1º caso:** Se $\tau_i$ representa um intervalo,$\tau_j$ representa um intervalo e $\tau_{n+1}$ representa um intervalo, então:

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 1 | $before(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 2 | $before(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 3 | $before(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 4 | $before(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 5 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 6 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 7 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 8 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 9 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 10 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 11 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 12 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 13 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 14 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 15 | $meets(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 16 | $meets(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 17 | $meets(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 18 | $meets(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 19 | $meets(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 20 | $meets(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 21 | $meets(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 22 | $meets(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 23 | $meets(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 24 | $meets(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 25 | $overlaps(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 26 | $overlaps(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 27 | $overlaps(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 28 | $overlaps(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 29 | $overlaps(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 30 | $overlpas(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 31 | $overlaps(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 32 | $overlaps(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 33 | $overlaps(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 34 | $overlaps(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 35 | $overlaps(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 36 | $overlaps(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 37 | $starts(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 38 | $starts(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 39 | $starts(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 40 | $starts(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 41 | $starts(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 42 | $starts(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 43 | $starts(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 44 | $starts(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 45 | $during(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 46 | $during(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 47 | $during(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 48 | $during(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 49 | $during(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 50 | $during(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 51 | $during(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 52 | $during(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 53 | $during(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 54 | $during(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 55 | $finishes(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 56 | $finishes(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 57 | $finishes(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 58 | $finishes(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 59 | $finishes(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 60 | $finishes(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 61 | $finishes(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 62 | $finishes(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $finishes(\tau_i, \tau_{n+1})$ |

**2º caso:** Se $\tau_i$ representa um intervalo,$\tau_j$ representa um intervalo e $\tau_{n+1}$ representa um ponto, então:

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 1 | $before(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 2 | $meets(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 3 | $overlaps(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 4 | $starts(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 5 | $during(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 6 | $during(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 7 | $finishes(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |

**3º caso:** Se $\tau_i$ representa um intervalo,$\tau_j$ representa um ponto e $\tau_{n+1}$ representa um intervalo, então:

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 1 | $before(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 2 | $before(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 3 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 4 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 5 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 6 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 7 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 8 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 9 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $meets(\tau_i, \tau_{n+1})$ |
| 10 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $overlaps(\tau_i, \tau_{n+1})$ |
| 11 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 12 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |

**4º caso:** Se $\tau_i$ representa um ponto,$\tau_j$ representa um intervalo e $\tau_{n+1}$ representa um intervalo, então:

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 1 | $before(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 2 | $before(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 3 | $before(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 4 | $before(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 5 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 6 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 7 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 8 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 9 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 10 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 11 | $starts(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 12 | $starts(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 13 | $starts(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 14 | $starts(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 15 | $starts(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 16 | $starts(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 17 | $during(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 18 | $during(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 19 | $during(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 20 | $during(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 21 | $during(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 22 | $during(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 23 | $during(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 24 | $during(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 25 | $finishes(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 26 | $finishes(\tau_i, \tau_j)$ | $meets(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 27 | $finishes(\tau_i, \tau_j)$ | $overlaps(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 28 | $finishes(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 29 | $finishes(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 30 | $finishes(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $finishes(\tau_i, \tau_{n+1})$ |

**5º caso:** Se $\tau_i$ representa um ponto,$\tau_j$ representa um ponto e $\tau_{n+1}$ representa um ponto, então:

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 1 | $before(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |

**6º caso:** Se $\tau_i$ representa um ponto,$\tau_j$ representa um ponto e $\tau_{n+1}$ representa um intervalo, então:

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 1 | $before(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 2 | $before(\tau_i, \tau_j)$ | $starts(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 3 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 4 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 5 | $before(\tau_i, \tau_j)$ | $during(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |
| 6 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 7 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $starts(\tau_i, \tau_{n+1})$ |
| 8 | $before(\tau_i, \tau_j)$ | $finishes(\tau_j, \tau_{n+1})$ | $during(\tau_i, \tau_{n+1})$ |

**7º caso:** Se $\tau_i$ representa um ponto,$\tau_j$ representa um intervalo e $\tau_{n+1}$ representa um ponto, então:

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 1 | $before(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 2 | $starts(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 3 | $during(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |
| 4 | $finishes(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |

**8º caso:** Se $\tau_i$ representa um intervalo,$\tau_j$ representa um ponto e $\tau_{n+1}$ representa um ponto, então:

| | $\mathcal{T}'_{i,j}(\tau_i, \tau_j)$ | $\mathcal{T}'_{j,n+1}(\tau_j, \tau_{n+1})$ | $r(\tau_i, \tau_{n+1})$ |
|---|---|---|---|
| 1 | $before(\tau_i, \tau_j)$ | $before(\tau_j, \tau_{n+1})$ | $before(\tau_i, \tau_{n+1})$ |

# Livros Grátis

( http://www.livrosgratis.com.br )

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo