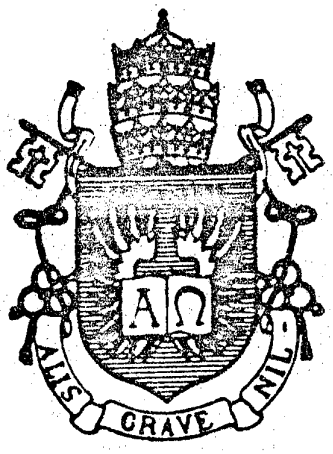


PUC

Série : Monografias em Ciência da Computação

No. 25/89

BANCOS DE DADOS DEDUTIVOS TEMPORAIS: UMA REVISÃO

José de Jesús Pérez Alcázar

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453

RIO DE JANEIRO - BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Séries: Monografias em Ciência da Computação, No 25/89

Editor: Paulo A. S. Veloso Novembro, 1989

BANCOS DE DADOS DEDUTIVOS TEMPORAIS: UMA REVISÃO *

José de Jesús Pérez Alcázar **

* Trabalho parcialmente financiado pela FINEP e CNPQ

** Cursando o programa de doutoramento do DI

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC RIO, Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
BRASIL

Tel.: (021) 529-9386 TELEX: 31078 FAX: (021) 274-4546
BITNET: userrtlc@lncc.bitnet

RESUMO

Um dos objetivos para integrar programação lógica e bancos de dados é fornecer mecanismos mais expressivos para a construção e gerenciamento de sistemas de informação. A integração se torna particularmente poderosa se for adicionado o aspecto temporal. Este relatório analisa alguns trabalhos a respeito, dando especial atenção ao Cálculo de Eventos proposto por Kowalski.

PALAVRAS-CHAVE

Bancos de dados temporais, bancos de dados dedutivos, bancos de dados históricos, bancos de dados de repetição, cálculo de eventos, cálculo de situações, semântica de caso, programação em lógica, Prolog.

ABSTRACT

One of the objectives to integrate logic programming and databases is to provide more expressive mechanisms for the construction and management of information systems. The integration is particularly powerful if the temporal aspect is added. This report examines a number of works on the subject, giving special attention to the Event Calculus proposed by Kowalski.

KEYWORDS

Temporal databases, deductive databases, historical databases, rollback databases, event calculus, situation calculus, case semantics, logic programming, Prolog.

ÍNDICE

1. INTRODUÇÃO

Uma das tendências que mais tem chamado a atenção aos pesquisadores é o fato de integrar a programação lógica e os bancos de dados. Através da lógica é possível fornecer mecanismos de modelagem mais expressivos e ricos com os quais é possível capturar um maior significado dos dados [PCGJ86]. A maioria dos trabalhos nesta área tem sido enfocados no papel da programação lógica na extensão de bancos dos dados relacionais com aspectos como dedução, armazenamento de informação não orientada a registros e facilidades de combinar esquema (metadados e restrições) com fatos do banco de dados [BoKo82]. A programação lógica também fornece uma maneira uniforme e elegante de implementar visões, linguagens de consulta e valores nulos [PCGJ86].

Um apecto de vital importância que tem sido esquecido na maioria das novas propostas de modelagem de dados é a consideração do tempo. Livros mais ou menos recentes [TsLo82] pretendem considerar o tempo como um aspecto secundário que deve ser manipulado de maneira ad-hoc. Não obstante, o tempo é uma parte esencial da informação relacionada com os aspectos dinâmicos do mundo real e já que os bancos de dados são modelos do mundo real, estes deveriam permitir mecanismos para registrar e processar aspectos do mundo real que mudam com o tempo. Por exemplo, deveriam suportar consultas históricas, registrar mudanças proativas e retroativas, facilitar a análise de tendências (esencial nos sistemas de apoio à decisão) e facilitar o suporte de versões (de grande utilidade em CAD, CASE, etc) [Snodg86].

Este relatório pretende mostrar algumas das pesquisas na área de Bancos de Dados temporais e principalmente o trabalho de Kowalski e seu grupo do Imperial College, chamado "Cálculo de eventos", que relaciona aspectos da programação lógica ao tratamento do tempo [KoSe86].

Na Segunda Seção deste relatório, tenta-se dar um marco conceitual a este trabalho apresentando os conceitos básicos da programação lógica aplicada aos bancos de dados. Será feita uma análise das pesquisas relacionadas com o tempo e bancos de dados, explicando os conceitos básicos utilizados nesta área.

Na Terceira Seção será analisada a proposta de Kowalski et alii, para o raciocínio temporal, onde pretende-se misturar os conceitos descritos na seção anterior. O cálculo de eventos, trata com a representação dos eventos em atualizações de bancos de dados e representação narrativa [Kowa86]. Portanto, nesta seção, são analisadas desde este ponto de vista, algumas propostas que originaram o cálculo de eventos. Para finalizar, será descrita uma extensão do cálculo de eventos desenvolvida pelo grupo do Imperial College.

Na Quarta Seção serão estudadas outras propostas de

raciocínio temporal e em geral de suporte para bancos de dados dedutivos temporais.

Na Quinta e Última Seção serão listadas algumas críticas ao cálculo de eventos e especificamente ao uso de PROLOG em bancos de dados. Algumas propostas de trabalhos futuros e de possíveis melhoras a esta abordagem, serão analisadas.

2 CONCEITOS BÁSICOS

Nesta seção pretende-se dar uma base conceitual para a leitura das seções seguintes. Estudan-se os conceitos básicos da programação lógica e cos bancos de dados temporais.

2.1. Lógica e Bancos de Dados

A lógica em bancos de dados tem sido usada como um sistema de inferência e como uma linguagem de representação, isto é, para fornecer capacidades dedutivas nos SGBD's e também para fornecer um formalismo a banco de dados convencionais (hierárquico, rede e relacional) [GaMN84]. No segundo caso, a lógica fornece o suporte formal para estudar problemas clássicos de bancos de dados e em alguns casos a lógica pode ajudar ao entendimento destes e portanto na sua solução.

No primeiro caso, a lógica estende os bancos de dados convencionais para permitir dedução. Este novo tipo de bancos de dados são chamados de dedutivos. Melo [Melo88] define um banco de dados dedutivo, como aquele em que estão presentes todas as características funcionais dos bancos de dados convencionais, acrescidas de um mecanismo de inferência, que possibilita a dedução de novos fatos a partir dos dados (chamada parte explícita do banco de dados) e de regras (chamada parte implícita do banco de dados) Não obstante, esta definição ser clara, necessita-se de uma definição mais formal para apresentar alguns conceitos que serão de utilidade mais na frente.

Quando um banco de dados é projetado ou utilizado, são feitas as seguintes hipóteses [Fros86, GaMN84]:

- Hipótese do domínio fechado. Ela define que não existe nenhum outro indivíduo além dos existentes no banco de dados.

- Hipótese de nomes únicos. A qual define que indivíduos com nomes diferentes, são diferentes.

- Hipótese do mundo fechado chamada também convenção para informação negativa, a qual define que fatos não conhecidos são assumidos como falsos. Por exemplo, se o relacionamento de casamento entre João e Maria não é representado explicitamemte no banco de dados, então a hipótese do mundo fechado implicaria que João e Maria não estão casados.

É possível então definir uma formulação teórica de um banco de dados convencional a partir de axiomas ou regras que consideram as hipóteses anteriores. Estes axiomas são de três tipos [Reit84]:

1. **Assertivas**. Símbolos predicativos para cada relação n-ária do banco de dados.

2. **Axiomas de particularização**.

- Axiomas de completude (hipótese do mundo fechado);
- Axiomas de nomes únicos,
- Axiomas do fecho de comínio.

3 **Axiomas de igualdade**. Especificam as propriedades comuns de igualdade, que são necessárias nos axiomas de particularização.

Com estes conceitos pode-se partir para uma definição mais formal de um Banco de Dados Dedutivo (BDD). Entretanto, os BDD's podem ser divididos em definidos e indefinidos. Os fatos em um BDD indefinido, armazenados ou deduzidos, podem ser da forma "P(a) ou P(b)", onde sabe-se que o fato completo é verdadeiro mas existe incerteza sobre o valor de verdade de P(a) ou de P(b). Em um banco de dados dedutivo definido, isto não é permitido.

Neste trabalho trataremos os BDDs definidos, não obstante pesquisas estão sendo feitas no estudo de BDDs indefinidos [GrMi83]. Com estes conceitos podemos dizer que um BDD definido consiste de:

1) Uma teoria cujos axiomas próprios são:
   a) Axiomas de particularização.
   b) Fatos elementares;
   c) Regras de dedução. Conjunto de cláusulas definidas que não envolvem funções (por questões de segurança).

2) Um conjunto de restrições de integridade que consistem de fórmulas fechadas (sem variáveis livres).

Entretanto, a definição acima não pode ser usada diretamente como uma base para a implementação de SGBDs dedutivos. A complexidade combinatorial dos axiomas de particularização conduziria a sistemas ineficientes. O número de axiomas cresceria na medida em que novas entidades fossem adicionadas no BDs. Por exemplo, a hipótese do fecho de comínio é definida por extensão da seguinte maneira,

$$\forall x\ ((x = e1) \vee (x = e2) \vee \ldots \vee (x = e\ )),$$

onde ei representa uma entidade do Banco de Dados. Portanto, para cada entidade inserida no banco de dados será necessário estender o axioma. O mesmo deve ser feito para os outros axiomas. Sendo assim, é necessário uma definição operacional. Esta definição pode ser conseguida, tratando com fórmulas de escopo limitado ("range restricted") para a formulação de consultas, restrições de integridade e regras de dedução (evitando a hipótese do fecho de comínio), e através da negação por falha (evitando as hipóteses de completude e de nomes únicos). A negação por falha define que se uma teoria falha na prova de um predicado P, isto implica que não P é certo. Com estes conceitos um banco de dados dedutivo pode ser definido como [GaMN83].

1 Um conjunto de axiomas. Fatos elementares e regras de dedução,

2 Um conjunto de regras de integridade,

3 Uma meta-regra : Negação por falha.

As duas definições acima, a formal e operacional, são equivalentes, já que geram as mesmas respostas para as mesmas consultas. Não obstante, elas não são estritamente equivalentes. A definição formal está baseada na lógica de primeira ordem padrão a qual é monotónica e a definição operacional é não-monotónica.

Uma lógica monotónica segundo [GaMN83] é definida de tal maneira que dada uma teoria T (um conjunto de axiomas) na qual a fórmula x pode ser provada, então a adição a T de um outro axioma continuará permitindo a prova de x.

De acordo com a negação por falha, "não P(b)" pode ser inferido de {P(a), Q(b)} mas não de {P(a), Q(b)} U {P(b)}, implicando no uso de uma lógica não-monotónica quando é considerada a negação por falha.

2.2. O Tempo e os Bancos de Dados

A informação temporal tem sido sempre de grande importância nos sistemas de informação. Um exemplo claro disso são os sistemas contábeis e as folhas de pagamento. Não obstante, nestes sistemas, os programas de aplicação se encarregam de manipular os atributos que envolvem o tempo e o SGBD trata estes como qualquer outro valor (literal ou numérico).

A necessidade de fornecer um suporte para o processamento de informação temporal dentro de SGBDs tem sido amplamente reconhecida. Entretanto, é muito difícil modelar as atividades humanas dentro de um contexto temporal (complexidade conceitual). Além disso, tentativas feitas para implementar bancos de dados historicamente ricos sofreram da ausência de meios adequados para manipular grandes volumes de memória secundária, necessários para manter esta informação temporal [Aria86].

Apesar dos problemas acima descritos , o estudo em bancos de dados é neste momento um "tópico quente" cujo interesse cresce cada dia mais. Snodgrass [Snod86] relaciona 24 projetos com pesquisas ativas nesta área e em 1982, uma revisão bibliográfica continha mais ou menos 70 artigos que relacionavam o tempo e o processamento de informação [BADW86]. Desde então e até 1986 mais de 150 artigos apareceram. Uma relação bibliográfica destes pode ser encontrada em [Mcke86]. O anterior mostra um crescimento quase exponencial de trabalhos feitos nesta área durante estes últimos anos.

As causas desse crescimento são devidas à rápida diminuição do custo de memória secundária ligada ao nascimento de novas tecnologias de armazenamento tais como discos óticos e á

utilização de bancos de dados em novas áreas de aplicação chamadas "não convencionais" (Engenharia, Automação de escritórios, Sistemas de Apoio à Decisão, etc), que têm exigido uma série de requisitos como manipulação de versões e suporte temporal. Snodgrass [Snod85,87] classifica estas atividades de pesquisa de acordo com a ênfase em: a formulação da semântica do tempo a nível conceitual [Schi83, Bube77, HaMc81], o desenvolvimento de um modelo para bancos de dados temporais análogo ao modelo relacional [ClWa83, Codd79, Sern80] e o projeto de linguagens de consulta temporais [Snod87, Aria86, BenZ82]. Todos estas pesquisas incorporam um ou mais atributos de tempo (um só é insuficiente). Entretanto, tem existido uma certa confusão na definição destes atributos e na terminologia utilizada (tempo físico, lógico, de transação, etc) [SnAh85]. Para evitar esta confusão Snodgrass & Ahn definem uma taxonomia de tempo que consiste de três conceitos temporais diferentes para o uso em bancos de dados. Usando essa taxonomia é possível definir quatro tipos de bancos de dados classificados de acordo com a capacidade deles para suportar esses conceitos e em geral a informação temporal.

A maioria dos bancos de dados que incorporam o tempo suportam apenas um só aspecto do tempo, o tempo quando a informação é válida. Este aspecto é chamado de "tempo válido" [Snod87] ou "tempo de evento" [CoMa84]. Dois outros aspectos do tempo deveriam ser suportados, o "tempo de transação" também chamado "tempo físico" [CoMa84] e o tempo definido pelo usuário. O tempo de transação representa o tempo no qual os dados são armazenados no banco de dados. O tempo definido pelo usuário representa aqueles dados do tipo temporal de interesse do usuário que o SGBD trata como qualquer outro dado do banco de dados.

De acordo com os conceitos anteriores podemos classificar os bancos de dados em:

- Bancos de dados instantâneos ou estáticos;
- Bancos de dados de **repetição** ("Rollback");
- Bancos de dados históricos; e
- Bancos de dados temporais.

A seguir, serão estudados cada um dos tipos de bancos de dados separadamente. Os conceitos mencionados nessas seções estão baseados no modelo relacional, mas podem ser estendidos a outros modelos

2.2.1 Bancos de Dados instantâneos ou estáticos.

Neste caso a dinâmica do mundo real é modelada como um fato (instantâneo) ou um ponto particular do tempo. Um estado ou instante do banco de dados é seu conteúdo atual, o qual não necessariamente reflete o estado atual do mundo real, já que mudanças no banco de dados sempre estão atrasadas com respeito a mudanças no mundo real. Geralmente, um banco de dados, neste caso, representa o estado mais recente do mundo real sendo

modelado. No processo de atualização do banco de dados, estados passados são esquecidos completamente.

Os SGBDs, neste caso, não fornecem facilidades para a interpretação ou manipulação da informação temporal; estas operações são feitas pelos programas de aplicação escritos especialmente para isso. Portanto, o tipo de tempo que é suportado neste caso é o tempo definido pelo usuário. Um exemplo deste tipo de bancos de dados são os bancos de dados convencionais.

2.2.2. Bancos de dados de "repetição" "rollback"

Os bancos de dados instantâneos tem uma série de desvantagens, por exemplo, não é possível responder perguntas sobre estados passados. Uma forma de solucionar estes problemas é armazenar todos os estados passados do banco de dados, indexados por tempo, na medida que este evolui (muda). O anterior implica a representação do tempo de transação por parte do SGBD. É possível obter dados de um instantâneo (foto) de uma relação em qualquer tempo no passado, selecionando o estado do banco de dados mais próximo (<=) ao tempo definido. Atualizações só podem ser feitas sobre o último estado (instantâneo/foto) do banco de dados.

NOME	CARGO	TEMPO DE TRANSAÇÃO	
		início	fim
Maria	auxiliar	25/08/85	15/12/88
Maria	titular	15/12/88	∞
João	adjunto	01/06/85	∞
Sandra	auxiliar	01/03/80	01/06/85

Fig. 1. Uma Relação de um Banco de Dados de "Repetição"

Para representar um banco de dados de "repetição, utilizaremos como base o modelo relacional e como exemplo, o seguimento histórico dos professores de um departamento de alguma universidade. Neste caso, os dados são armazenados da forma descrita na Fig. 1. A partir da figura podem ser inferidos, entre outros, os seguintes dados:

- A informação de que Maria era professor auxiliar foi inserida no BD em 25/08/85.
- A informação sobre a promoção de Maria de professor adjunto para titular foi inserida no BD em 15/12/88 e ainda é válida.

Esta última informação é parte do último instantâneo do banco de dados. Cada transação define um novo instantâneo do banco de dados que é armazenado e indexado (o índice é o atributo tempo de transação) para ser consultado posteriormente.

Um exemplo de um SGBD que utiliza esta idéia é o POSTGRES [StRo86]. Estes tipos de banco de dados facilitam a implementação de versões.

2.2.3 Bancos de dados históricos

Os bancos de dados "rollback", não obstante, não preenchem todos os requisitos necessários para uma boa manipulação do tempo. Estes bancos suportam unicamente o tempo de transação o que implica no armazenamento da história das atividades do banco de dados e não da história do mundo real. Portanto, quando um dado é armazenado no banco de dados, ele é efetivado e pode ser acessível para atualização unicamente se pertence ao último instantâneo do banco de dados (igual que nos bancos de dados estáticos). Desta maneira, erros do passado não podem ser corrigidos (nunca podem ser esquecidos) e mudanças retroativas e proativas não podem ser feitas. Mais na frente estudaremos estes conceitos detalhadamente.

Uma outra abordagem são os bancos de dados históricos. Este tipo de bancos de dados armazena um só estado histórico para cada relação, no lugar de uma série de fotos (estados) do banco de dados. Estes bancos de dados armazenam a história sem erros, isto é, como ela é melhor conhecida. Dados errados são corrigidos e ao igual que os bancos de dados estáticos, nenhum registro destes dados é mantido. Portanto, não é possível ter uma visão do passado do banco de dados. Entretanto, o "tempo válido" é suportado o que permite ter uma história do mundo real.

NOME	CARGO	TEMPO VÁLIDO	
		início	fim
Maria	auxiliar	01/07/85	01/12/88
Maria	titular	01/12/88	∞
João	adjunto	30/05/85	∞
Sandra	auxiliar	29/02/80	30/05/85

Fig. 2. Uma Relação de um Banco de Dados Histórico.

Utilizando o mesmo exemplo anterior, um banco de dados histórico, no caso do modelo relacional, seria representado como na Fig. 2. Neste caso, por exemplo, sabe-se que Maria foi

contratada como professor auxiliar em 01/07/85 e que foi promovida para professor titular no dia 01/12/88. Se acontecer algum erro, por exemplo, suponha que Maria não foi promovida de professor auxiliar para titular e sim para professor adjunto, então a informação errada será removida e a nova tupla será inserida com o mesmo tempo válido associado.

Snodgrass & Ahn [SnAh85,86] enumeram duas diferenças entre bancos de dados históricos e de "rollback":

- As operações dos bancos de dados históricos devem ser mais sofisticadas já que a semântica do tempo válido é mais complexa.

- Os bancos de dados históricos permitem atualizações arbitrária enquanto que os bancos de dados "rollback" unicamente suportam a adição de novos fatos ou instantes do banco de dados.

2.2.4 Bancos de Dados Temporais

Enquanto, os bancos de dados de repetição suportam o tempo de transação, isto é, eles representam os dados válidos como são conhecidos em cada um dos instantes do passado; os bancos de dados históricos suportam o tempo válido, isto é, os dados válidos como são conhecidos agora e não como eram conhecidos no passado (qualquer erro do passado pode ser corrigido e perdido). Porquê, então, não tentar juntar as duas facilidades e suportar os dois tipos de tempos? Desta maneira será possível ver os dados válidos como são conhecidos em qualquer momento.

NOME	CARGO	TEMPO VÁLIDO		TEMPO DE TRANSAÇÃO	
		início	fim	início	fim
Maria	auxiliar	01/07/85	∞	25/08/85	15/12/88
Maria	auxiliar	01/07/85	01/12/88	15/12/88	∞
Maria	titular	15/12/88	∞	15/12/88	∞
João	adjunto	30/05/85	∞	01/06/85	∞
Sandra	auxiliar	29/02/80	∞	01/03/80	01/06/85
Sandra	titular	29/02/80	30/05/85	01/06/85	∞

Fig. 3. Uma Relação de um Banco de Dados Temporal

Os bancos de dados temporais suportam os dois tempos. Na Fig. 3, mostra-se a tabela de professores para este caso. Se acontecer algum erro, semelhante ao do exemplo do banco de dados

histórico, nenhuma tupla será removida, pelo contrário será inserida uma tupla com o tempo de transação e o mesmo tempo válido que a tupla errada.

3 CÁLCULO DE EVENTOS

Nesta seção tentaremos juntar as idéias descritas acima introduzindo um novo conceito, os bancos de dados dedutivos temporais.

Na primeira seção viu-se como a lógica pode ser usada em bancos de dados (especialmente a lógica de primeira ordem). Mas a lógica também é importante no processamento de linguagem natural, em aplicações legais, na formalização de programas, etc. Em geral, para todas essas aplicações uma representação do tempo é também importante.

Diferentes tipos de lógicas tem sido usadas para representar o tempo, por exemplo, a lógica modal [Gols86] ou outros tipos de lógicas diferentes da lógica de primeira ordem [HaMc85]. A lógica de primeira ordem tem sido pouco utilizada no caso do raciocínio temporal. Kowalski & Sergot apresentam um formalismo, baseado em cláusulas de Horn aumentado com a negação por falha, como uma alternativa. Este formalismo é chamado de cálculo de eventos [KoSe86] e pode ser executado em um sistema de programação lógica (DATALOG [GaVa88], PROLOG [Kowa79], etc). Pode-se pensar que esta é uma maneira simples e clara de integrar o raciocínio temporal com bancos de dados. Além disso, sabe-se que a lógica de primeira ordem pode ser um mecanismo uniforme e simples de representar e manipular bancos de dados [Kowa79].

3.1 Cálculo de eventos, uma descrição informal

O cálculo de eventos como já foi dito, foi desenvolvido como uma teoria para raciocinar sobre eventos em uma base de programação lógica. Os relacionamentos (propriedades dos objetos ou relacionamentos entre eles) e os períodos de tempo nos quais eles são verdadeiros são derivados (definidos) a partir de uma descrição dos eventos que ocorrem no mundo real. Esta teoria é baseada em parte no cálculo de situações [McHa69] mas enfocando o conceito de eventos como é definido nas representações em rêdes semânticas da "semântica de casos". O principal interesse do cálculo de eventos é a representação destes na atualização de bancos de dados e no entendimento narrativo [KoSe86]. Esta tentativa é similar aos trabalhos de Lee et al. [LeCC85] e a lógica temporal baseada em intervalos de Allen [Alle83,84]. Mais na frente serão descritos estes trabalhos e serão comparados com o cálculo de eventos. Esta descrição e comparação será feita baseada no trabalho de Sadri [Sadr87].

A seguir será utilizado um exemplo para descrever o cálculo de eventos. O aspecto de atualização em bancos de dados servirá de base para mostrar outras propostas de manipulação temporal e

para compará-las com o cálculo de eventos. O cálculo de eventos herdou destas propostas, aquelas características mais interessantes, sem ter em conta algumas de suas falhas. Depois da descrição destas propostas, o cálculo de eventos será definido formalmente. Finalmente, será mostrada uma proposta de extensão do cálculo de eventos que considera o tempo de transação

3.1.1 Um pequeno exemplo

Através do exemplo do histórico dos professores, serão reveladas algumas das características mais importantes do cálculo de eventos. Seja o seguinte texto:

- João foi contratado como professor auxiliar a 10 de maio de 1980.

- Sandra deixou de ser professora auxiliar a 1 de Junho de 1985.

- João deixou de ser professor adjunto a 1 de outubro de 1988.

- João foi promovido de professor auxiliar adjunto a 1 de junho de 1985.

Ao serem armazenados os dados acima no banco de dados, cada frase do texto equivale a uma atualização que adiciona novas informações na base de dados. A idéia do cálculo de eventos é descrever o texto através de eventos. Por exemplo, o texto acima pode ser descrito assim:

- e1 é um evento tal que,
  João foi contratado como professor auxiliar e
  a sua data é 10 de maio de 1980.

- e2 é um evento tal que,
  Sandra deixa a posição de professor auxiliar e
  a sua data é 1 de junho de 1985.

- e3 é um evento tal que,
  João deixa a posição de professor adjunto e
  a sua data é 1 de outubro de 1988

- e4 é um evento tal que,
  João é promovido de professor auxiliar a adjunto e
  a sua data é 1 de junho de 1985.

Através de uma representação gráfica é possível mostrar a evolução do banco de dados na medida que as sentenças vão sendo processadas. Por exemplo o evento e1 pode ser descrito como na Fig 4.

Como pode ser visto um evento inicia um intervalo de tempo no qual um relacionamento se mantem. Em geral, um evento pode iniciar mais de um período de tempo. No caso da Fig. 4, o evento

gera um período de tempo no qual o relacionamento João é professor auxiliar, é verdade. Se for representado o período gerado pelo evento e1 através do termo **depois(e1)**, este fato poderia ser generalizado por meio da seguinte regra:

X tem posição Y no período **depois(E) Se** E representa um evento no qual X é contratado como Y

Quando for inserido e2, um caso contrário acontece. O evento e2 termina um período de tempo. Veja Fig. 5.

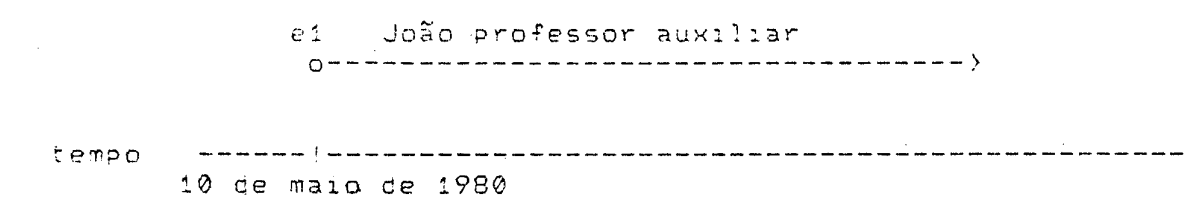


Fig. 4. Banco de Dados após a primeira atualização.

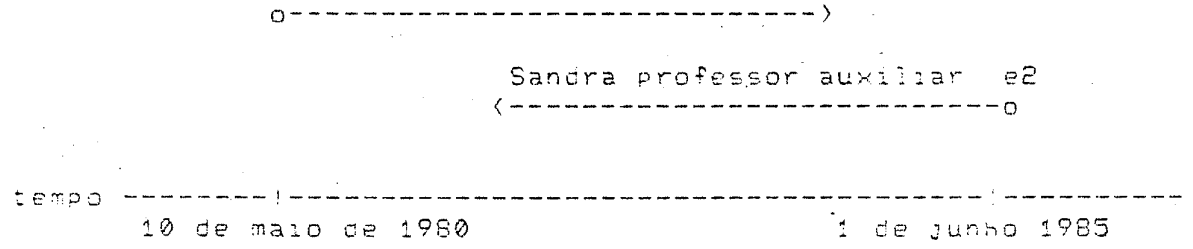


Fig. 5. Banco de Dados após a segunda atualização.

Este novo período identificado, pode ser representado pelo termo **antes(e2)** e podemos descrever este fato pela seguinte regra:

X tem posição Y no período **antes(E) Se** E representa um evento no qual X deixa a posição Y.

Depois de serem processadas todas as sentenças, isto é, depois de serem feitas todas as atualizações sobre o banco de casos, o banco de dados poderá ser representado pela Fig. 6.

Com a narrativa anterior é possível inferir que o período **depois (e1)**, finaliza com o evento e4 e que o período **antes(e4)** é exatamente, **depois(e1)**. No caso do período **antes(e3)**, é possível chegar a conclusões similares. Por exemplo, é possível inferir que o período **antes(e3)** inicia com o evento e4 e que este

período é exatamente igual a **depois(e4)**. Este raciocínio pode mudar se alguma outra atualização for feita no banco de dados. Por exemplo, se for definido um evento e5 declarando que João deixou de ser professor adjunto o 10 de agosto de 1986. Portanto, o raciocínio é não-monotónico.

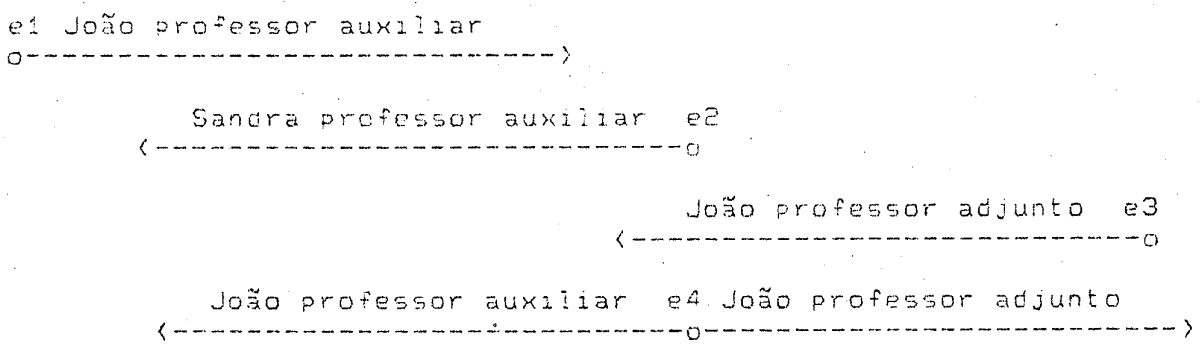


Fig. 6 Banco de Dados após a quarta atualização.

O cálculo de eventos está formado por uma série de regras ou axiomas que regem este raciocínio. Mais tarde mostraremos estas regras. Por enquanto, este exemplo foi definido para ter uma visão informal do cálculo de eventos e para mostrar antecipadamente algumas de suas características:

- Os eventos podem ser processados em uma ordem diferente daquela na qual eles acontecem.
- As atualizações são aditivas; não é necessário fazer nenhuma remoção de dados.
- Embora os eventos podem ser tratados como sendo efetuados instantaneamente, o cálculo de eventos deixa aberta a possibilidade de que eles possam ter uma duração como define Allen [Alle83].
- Relacionamentos são assumidos para serem verdade durante períodos de tempo, que têm duração. Um período de tempo persiste tanto no futuro, até que um evento termina este, quanto no passado, até que um evento inicia este. Portanto, existe um tratamento simétrico do tempo. Esta simetria, segundo Kowalski [Kowa86], é muito poderosa no caso geral, mas eleva a complexidade em alguns casos onde ela não é necessária.

3.2. Atualizações em Bancos de Dados.

Nesta seção, falaremos da maneira como é feito o tratamento das atualizações em bancos de dados relacionais, bancos de dados históricos, lógica modal, cálculo de situações e semântica de

caso. Depois serão comparadas estas abordagens com a proposta do cálculo de eventos.

NOME	CARGO
Maria	Titular
João	Auxiliar
Sandra	Adjunto

Fig. 7. Tabela de Professores

3.2.1. Atualizações em Bancos de Dados Relacionais

Em bancos de dados relacionais as atualizações são feitas mudando valores de campos das tuplas ou adicionando e removendo tuplas (na realidade todas as atualizações podem ser reduzidas a adições e remoções de tuplas). A vantagem desta abordagem é a eficiência tanto em tempo de execução quanto em espaço físico utilizado. Não obstante esta vantagem gera uma série de desvantagens enumeradas abaixo:

- Estes bancos de dados confundem a simulação de eventos com a descrição do mundo real. Por exemplo, seja a tabela da Fig. 7, a atualização,

  ```
  Remova posição(joão, professor_auxiliar)
  Adicione posição(joão, professor_adjunto)
  ```

  pode significar ou uma mudança do estado do mundo devido a um evento de "promoção" ou uma mudança de conhecimento sobre a posição de João na universidade.

- Atualizações devem ser feitas na mesma ordem em que os eventos ocorrem. Esta restrição está relacionada ao fato discutido na seção 2.2.1, na qual era definido que fatos passados são totalmente esquecidos, isto é, a informação sobre estados passados é destruida quando são feitas atualizações.

- Atualizações sintaticamente aceitáveis podem não ter validez semântica. Por exemplo, a atualização,

  ```
  Remova posição(joão, professor_auxiliar)
  Adicione posição(maria, professor_adjunto)
  ```

  não corresponde a nenhum evento do mundo real.

3.2.2 Atualizações em Bancos de Dados Históricos

As duas primeiras desvantagens consideradas na abordagem anterior podem ser eliminadas, se for incorporado o tempo como parte integrante do modelo. Já foi visto que os bancos de dados históricos suportam o tempo válido, repetiremos algumas de suas vantagens (e dos bancos de dados temporais também) bem como algumas de suas desvantagens.

Neste tipo de bancos de dados, cada tupla das relações é estendida para conter o tempo inicial no qual o relacionamento virou válido e o tempo final no qual este deixou de sê-lo (Veja Fig. 2). Um tempo "∞" representando o tempo final, indica que o relacionamento persiste no futuro, e um tempo "∞" representando o tempo inicial, indica que não é conhecido o tempo no qual o relacionamento virou válido. O anterior implica que é possível manter informação incompleta sobre o passado e que esta pode ser atualizada mais tarde quando seja conhecida. Portanto, uma das vantagens dos bancos de dados históricos é que os relacionamentos podem ser registrados independente da ordem em que acontecem.

Outra vantagem desta abordagem é a possibilidade de diferenciar entre uma atualização que representa uma mudança do mundo real de uma que representa uma mudança do conhecimento deste mundo. Por exemplo, se João for promovido de professor auxiliar a adjunto a 1 de junho de 1985, esta atualização pode ser feita no banco de dados histórico da seguinte maneira:

Remova posição(joão, professor_auxiliar, 10/5/80, ∞)
Adicione posição(joão, professor_auxiliar, 10/5/80, 1/6/85)
Adicione posição(joão, professor_adjunto, 1/6/85, ∞)

Uma mudança de conhecimento é feita de outra forma. Por exemplo, se João não era professor auxiliar em 10/5/80, e sim professor adjunto, então esta atualização é feita da seguinte maneira:

Remova posição (joão, professor_auxiliar, 10/5/80, ∞)
Adicione posição(joão, professor_adjunto, 10/5/80, ∞)

A principal desvantagem desta abordagem (inclusive nos bancos temporais) é a perda da estrutura semântica dos eventos que terminam e finalizam os relacionamentos, devido a que as atualizações são feitas registrando só o início e o fim dos relacionamentos. Portanto, devem ser implementadas restrições de integridade de difícil implementação para não permitir atualizações arbitrárias, isto é que não correspondem a eventos reais do mundo.

3.2.3 Atualizações em Lógica Temporal

A lógica temporal está relacionada ao raciocínio sobre o tempo. Ela evita a representação explícita do tempo através de

operadores modais tais como **passado**, **futuro**, **sempre**, etc. Portanto, a principal vantagem desta abordagem é sua naturalidade de expressão. Por exemplo, um banco de dados em lógica modal estaria representado com cláusulas da forma:

```
passado(posição(joão, professor_auxiliar))
futuro(posição(joão, professor_adjunto))
```

Outras vantagens desta abordagem é que a informação sobre estados passados pode ser preservada e a informação pode ser assimilada em uma ordem diferente à ordem na qual os eventos realmente ocorrem.

Não obstante, estes operadores modais geram as seguintes desvantagens:

- Eles são sensíveis ao contexto. Portanto, mudanças complexas podem ser feitas para preservar as consistências no caso de atualizações de bancos de dados quando o contexto muda). Por exemplo, seja o banco de dados:

  ```
  passado(posição(joão, professor_auxiliar))
  presente(posição(sandra, professor_auxiliar))
  presente(posição(joão, professor_adjunto))
  futuro(posição(joão, professor_titular))
  ```

  No caso de registrar o evento, "João foi promovido de professor adjunto a professor titular ", seriam necessárias as seguintes operações:

  ```
  Remova(presente(posição(joão, professor_adjunto)))
  Adicione(passado(posição(joão, professor_adjunto)))
  Remova(futuro(posição(joão, professor_titular)))
  Adicione(presente(posição(joão, professor_titular)))
  ```

- Estes operadores não permitem distinguir (facilmente) entre diferentes ocorrências do mesmo evento ou relacionamento. Por exemplo, se João foi contratado duas vezes como professor auxiliar, por que foi demitido e reintegrado novamente, esta abordagem não permitirá diferenciar estes dois eventos.

- A mistura destes operadores com referencias explicitas ao tempo não é conveniente já que os operadores deixam de ser essenciais.

- Como no caso de bancos de dados relacionais, manter a integridade semântica é um processo complexo. Isto é devido à complexidade dos operadores.

- Procedimentos de prova para a lógica modal são muito complexos comparados aos procedimentos da lógica clássica.

3.2.4 Atualizações ao Cálculo de Situações

O cálculo de situações foi desenvolvido por McCarthy & Hayes [McHa69] como um fundamento lógico para o raciocínio sobre ações. Esta abordagem tem sido a base de outros estudos como o do Hanks & Mcdermott [HaMc87, HaMc85] que será analisado mais na frente. Neste relatório será analisada uma versão do cálculo de situações formulado em programação lógica e definida por Kowalski [Kowa79, Kowa86].

Nesta abordagem, por motivos de generalidade, um relacionamento é representado através de um predicado que tem como parâmetros o nome do relacionamento e o nome do estado global no qual o relacionamento é válido,

é_válido(R, S).

Por exemplo, o fato de que a posição de João seja professor auxiliar no estado zero, seria descrita da seguinte maneira,

é_válido(profissão(joão, professor_auxiliar), e0).

Esta notação é melhor que profissão(joão, professor_auxiliar, e0). As duas são válidas dentro da lógica de primeira ordem, mas a primeira permite que uma variável unifique com termos que representam relacionamentos e a segunda não (é_válido é um meta-predicado).

Nesta abordagem um estado é definido em função do anterior estado e do evento que gerou a transição do estado. O estado inicial é a exceção, já que ele é definido como uma constante e0. Assim os outros estados seriam definidos como:

```
e1 = resultado(contrato(maria, professor_auxiliar),e0)
e2 = resultado(promoção(joão, professor_adjunto),
               resultado(contrato(maria, professor_auxiliar),e0))
```

O primeiro parâmetro do termo resultado(a,s) define um tipo de evento e já que no cálculo de situações, somente um evento de um tipo dado pode ocorrer em um dado estado. A combinação tipo de evento e estado constitui uma única ocorrência de um evento. Desta maneira, o cálculo de situações associa um estado global (como se fosse um tipo de contexto) a todo relacionamento que pode ser mudado por uma atualização.

Da mesma maneira que a lógica temporal modal, o cálculo de situações tem a vantagem de que a informação sobre o passado pode ser mantida quando são feitas atualizações (todos os fatos são rotulados pelo estado no qual aconteceram). Entretanto, ao contrário da lógica modal, as atualizações são aditivas, isto é, as atualizações são feitas, unicamente, adicionando sentenças. Existem regras gerais para derivar informação sobre estados a partir da descrição dos eventos, isto permite definir uma estrutura semântica para os eventos. Por exemplo,

é_válido(posição(joão, professor_adjunto),
resultado(promoção(joão, professor_adjunto), e1),

pode ser derivado da descrição do evento,

aconteceu(promoção(joão, professor_adjunto), e1),

através da regra geral do cálculo de situações,

é_válido(R, resultado(A, S)) **Se** aconteceu(A, S), inicia(A, R).

A regra inicia(A, R) é específica da aplicação e define quando um relacionamento começa a ser válido. Por exemplo,

inicia(promoção(A, X), posição(A, X)).

As atualizações são feitas através de remoções implícitas de relacionamentos que já não são válidas. Isto é feito através de outra regra geral chamada "axioma de marco" ("frame axiom") e que definiremos a seguir.

é_válido(R, resultado(A, S)) **Se** é_válido(R, S),
não termina(A, R).

O predicado **termina** é também uma regra específica da aplicação. Por exemplo,

termina(renuncia(X, Y), posição(X, Y)).

As regras que definem **inicia** e **termina** descrevem a semântica dos eventos. Pre-condições de eventos podem ser expressas como restrições de integridade.

O cálculo de situações tem uma série de desvantagens:

- A principal e maior desvantagem é o chamado "problema de marco". Kowalski [KowaB6] enumera três aspectos associadas a este problema. O aspecto **epistemológico**, o aspecto computacional e o problema de ramificação. O primeiro aspecto está relacionado ao problema de formalizar, de uma forma natural todos aqueles relacionamentos não terminados por um evento e que são preservados. Este problema acontece quando diferentes relacionamentos que mudam com o tempo são representados por diferentes símbolos predicativos e é resolvido pelo uso do predicado **é_válido** e negação por falha no "axioma de marco". O aspecto computacional está relacionado ao excessivo "overhead" computacional associado com o uso do "axioma de marco" para raciocinar que virtualmente todos os relacionamentos são preservados de estado a estado. O "problema de ramificação" define que um relacionamento poderia persistir quando a base para sua derivação é terminada.

- Já que estados (globais) não iniciais são definidos de estados anteriores, a adição de novos eventos deve ser na

ordem em que eles acontecem no mundo real.

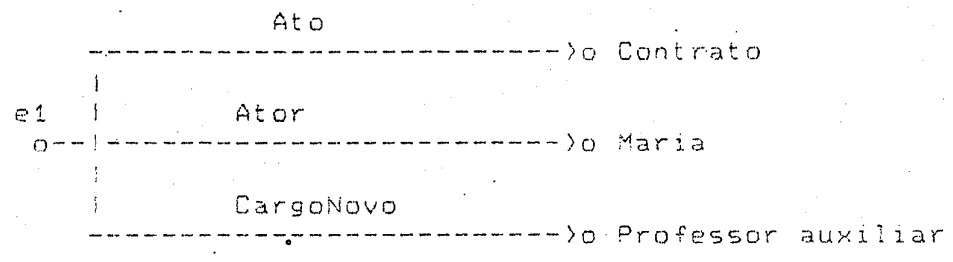
- Devido ao uso da notação funcional para identificar estados, sem igualdade, não é possível a adição de novos dados sobre o passado.

3.2.5 Atualizações na Semântica de Caso

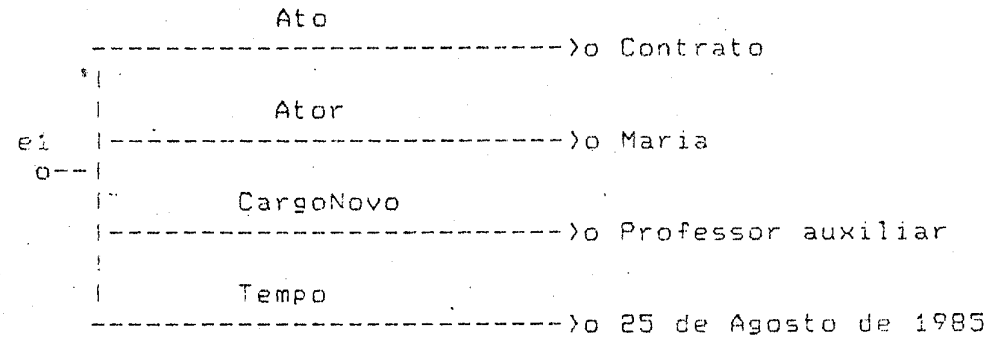
A semântica de caso é usada para o processamento de linguagens naturais e a representação gráfica usada nestes casos é a rêde semântica [Qui68]. Por exemplo, a frase,

"Maria foi contratada como professora auxiliar"

seria representada como:



Esta representação facilita a adição posterior de informação sobre o evento, por exemplo, se este evento aconteceu a 25 de agosto de 1985 então a adição desta nova informação seria feita adicionando um novo arco, da seguinte maneira:



Esta notação é vantajosa porque facilita a adição de informação sobre os eventos em qualquer ordem e as atualizações podem ser feitas sem recorrer a remoções. Portanto, informação do passado pode ser preservada e os eventos podem ser adicionados em uma ordem diferente da ordem em que acontecem no mundo real.

Outra característica desta abordagem é que os eventos podem

se relacionar temporalmente sem ter tempos específicos associados. Por exemplo, através do relacionamento **antes**, **depois**, etc. que podem ser representados como arcos ligando os eventos. Também diferentes ocorrências do mesmo tipo de evento podem ser diferenciados sem a necessidade deles terem um tempo associado.

A semântica de caso não utiliza estados globais o que permite representar eventos que ocorrem paralelamente, facilitando assim, a representação de situações semanticamente complexas. As desvantagens desta representação são:

- Regras gerais não podem ser representadas convenientemente.
- Não existem mecanismos de inferência para a avaliação de consultas.
- As atualizações não têm estrutura semântica porque a representação das ligações dos eventos com os relacionamentos que eles iniciam ou terminam, é complicada.

3.3. Descrição Formal do Cálculo de Eventos.

Em geral as desvantagens da semântica de caso surgem pela falta de uma base de programação lógica. A reformulação da semântica de caso em uma base de programação lógica tem sido uma das maiores influências do cálculo de eventos. Outras influências tem sido a lógica temporal de Allen [Alle83, Alle84], que será descrita mais na frente. Desta lógica, o cálculo de eventos herdou o uso de períodos de tempo no lugar de instantes de tempo. Do cálculo de situações herdou o uso de regras gerais para ligar eventos com relacionamentos que os iniciam e terminam.

O uso de períodos de tempo no lugar de estados globais para associar com os relacionamentos evita a necessidade de raciocinar explicitamente sobre relacionamentos que não são terminados e são preservados de estado a estado. Assim, os relacionamentos persistem até que eles são terminados.

No cálculo de eventos cada período de tempo é identificado unicamente pelo relacionamento que é válido neste período e pelo evento que o inicia ou termina. Isto porque cada evento pode iniciar ou terminar mais de um período e o relacionamento pode ser válido em vários períodos de tempo (nos bancos de dados históricos os períodos de tempo são identificados pelos tempos de início e fim, o que acarretava a falta de estrutura semântica). O termo,

depois(E, R)

representa o período de tempo iniciado pelo evento E e no qual R é válido. Por exemplo,

depois(e1, posição(maria, professor_auxiliar))

Da mesma maneira, o termo,

antes(E, R)

representa o período de tempo terminado pelo evento E e no qual R é válido. Por exemplo,

antes(e1, posição(maria, desempregado)).

O uso de termos para representar períodos, no lugar de tempos explícitos de início e fim como nos bancos de dados históricos, permite a representação de fins e inícios desconhecidos, sem a necessidade de introduzir tempos fictícios como "∞".

Da mesma maneira que o cálculo de situações, o cálculo de eventos usa o predicado geral,

é_válido(depois(E, R))

para expressar que o relacionamento R é válido no período depois(E, R). Um predicado similar pode ser definido para o termo **antes**. Para facilitar o entendimento do leitor, as regras gerais do cálculo de eventos serão descritas, a partir de um exemplo. Seja a descrição de eventos dada na seção 3.1. Fazendo uma análise desta descrição é possível deduzir os seguintes fatos:

- Existem dois períodos, um iniciado pelo evento e1 e outro iniciado pelo evento e4.
- Existem também três períodos, um terminado pelo evento e2, outro pelo evento e3 e finalmente outro pelo evento e4.

Como no cálculo de situações, o cálculo de eventos tem regras para deduzir tais períodos de tempo. Estas regras são chamadas regras de iniciação e terminação [Kowa86] e são definidas da seguinte maneira:

é_válido(depois(E, R)) **Se** inicia(E, R) **Regra de iniciação (CE1)**
é_válido(antes(E, R)) **Se** termina(E, R) **Regra de terminação (CE2)**

Junto a estas regras são definidas regras específicas da aplicação que definem os predicados inicia e termina. Para o caso do exemplo tem-se:

```
inicia(E, posicao(X, Y)) Se ato(E, contrato),
                            ator(E, X),
                            cargonovo(E, Y).

inicia(E, posicao(X, Y)) Se ato(E, promocao),
                            ator(E, X),
                            cargonovo(E, Y)

termina(E, posicao(X, Y)) Se ato(E, renuncia),
                             ator(E, X),
                             cargovelho(E, Y)
```

termina(E, posicao(X, Y)) **Se** ato(E, promocao),
ator(E, X),
cargovelho(E, Y)

As regras acima definem os relacionamentos que cada tipo de evento inicia ou termina. A definição dos tipos de eventos depende da aplicação sendo implementada. Assim, das regras anteriores é possível deduzir que os seguintes relacionamentos são válidos:

é_válido(depois(e1, posição(joão, professor_auxiliar)))
é_válido(antes(e2, posição(sandra, professor_auxiliar)))
é_válido(antes(e3, posição(joão, professor_adjunto)))
é_válido(antes(e4, posição(joão, professor_auxiliar)))
é_válido(depois(e4, posição(joão, professor_adjunto)))

Os relacionamentos anteriores determinam períodos de tempo, entretanto, como determinar o início e o fim destes períodos de tempo? No cálculo de eventos são definidas as seguintes regras para esse fim:

começo(depois(E, R), E) (CE3)
fim(antes(E, R), E) (CE4)
começo(antes(E1, R), E) **Se** depois(E, R) = antes(E1, R) (CE5)
fim(depois(E, R), E1) **Se** depois(E, R) = antes(E1, R) (CE6)

O predicado **inicio(X, Y)** define que o evento Y inicia o período X. O predicado fim é definido de maneira similar. Estas regras permitem entre outras coisas a dedução, no caso do exemplo, dos seguintes fatos:

- O início do período depois(e1, posição(joão, professor_auxiliar)) é o evento e1 (utilizando CE3).

- O fim do período depois(e1, posição(joão, professor_auxiliar)) é o evento e4 (utilizando CE4).

Entretanto, existe um problema na definição de igualdade de dois períodos de tempo. Esta definição precisa de um raciocínio por falha partindo do fato de que não existe nenhum evento que inicie ou termine um período de tempo que crie conflitos. O seguinte axioma define esta igualdade:

depois(E, R) = antes(E1, R) **Se** é_válido(depois(E, R)),
é_válido(antes(E1, R)),
E < E1,
não(quebra(E, R, E1)). (CE7)

Este axioma é baseado no "axioma de persistência" [Kowa86] o qual define que uma vez inicializado um relacionamento, este se mantem indefinidamente até que ele é terminado (o caso contrário de persistência no passado pode ser definido por um axioma similar). Não obstante este axioma não é o suficientemente geral para considerar todos os casos [Kowa86].

No caso do CE7 o operador "(" é definido sobre a ordem cronológica dos eventos e o conectivo **não** é considerado como a negação por falha. O predicado "quebra" é definido como:

```
quebra(E, R, E1) Se é_válido(depois(E2, R2)),
                    exclusivo(R, R2),
                    E ( E2,
                    E2 ( E1.                (CE8)

quebra(E, R, E1) Se é_válido(antes(E2, R2)),
                    exclusivo(R, R2),
                    E ( E2,
                    E2 ( E1.                (CE9)
```

O predicado exclusivo define que dois relacionamentos não podem acontecer simultaneamente. Isto é, os dois ou são iguais ou são incompatíveis. Portanto, "exclusivo" é definido como:

```
exclusivo(R, R).
exclusivo(R, R1) Se incompativel(R, R1).
```

O predicado "incompatível" depende do domínio da aplicação e para o exemplo tratado aqui, será definido como,

```
incompativel(posição(X, Y), posição(X, Y1)) Se não(Y == Y1),
```

que indica que uma pessoa não pode ter duas posições ao mesmo tempo

Até aqui, as regras ou axiomas consideravam só os casos onde os pontos extremos dos períodos são eventos definidos como entradas ao sistema. Entretanto, existem alguns casos onde é necessário inferir a existência de pontos extremos sem o conhecimento dos eventos correspondentes. Kowalski & Sergot [KoSe86] consideram três casos com os quais eles cobrem todos os casos onde os períodos de tempo interagem para implicar a existência de pontos extremos. Em [KoSe86] são amostrados os princípios gerais para a geração destes casos.

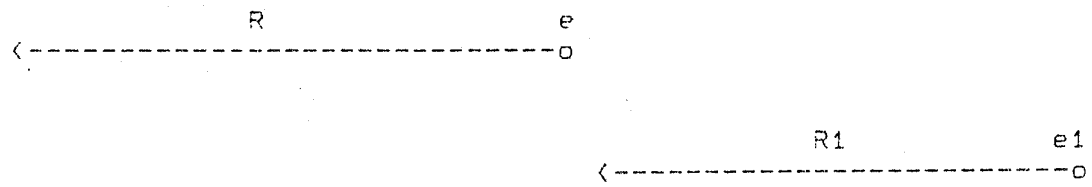


Fig. 8. Caso 1.

O primeiro caso acontece quando existem dois eventos que terminam relacionamentos que são mutuamente exclusivos e o seguinte relacionamento não tem um evento que o inicie. Assim é

possível então deduzir, por falha ("default"), um evento que inicie este relacionamento. O caso é mostrado na Fig. 8. O novo evento pode ser definido como uma função do período de tempo, por exemplo, **começo(antes(E1,R1))**, de acordo com a seguinte regra:

```
começo(antes(E1,R1),começo(antes(E1,R1))),
E <= começo(antes(E1, U1))        Se é_válido(antes(E,R)),
                                     é_válido(antes(E1,R1)),
                                     exclusivo(R, R1),
                                     E < E1,
                                     não(quebra(E, R1, E1)).
```

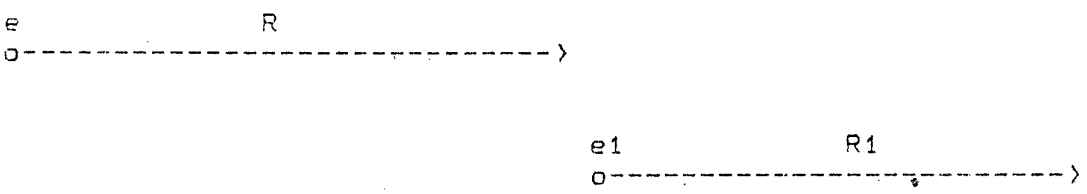


Fig. 9. Caso 2.

O segundo caso é simétrico ao anterior, o evento que é deduzido é aquele que finaliza o primeiro relacionamento como mostra a Fig. 9. Este evento pode ser identificado através de uma função do período, por exemplo, **final(depois(E, R))**. A seguinte regra é utilizada na dedução deste fato.

```
fim(depois(E, R), final(depois(E, R))),
fim(depois(E, R)) <= E1            Se é_válido(depois(E, R)),
                                      é_válido(depois(E1, R1)),
                                      exclusivo(R, R1),
                                      E < E1,
                                      não(quebra(E, R, E1)).
```

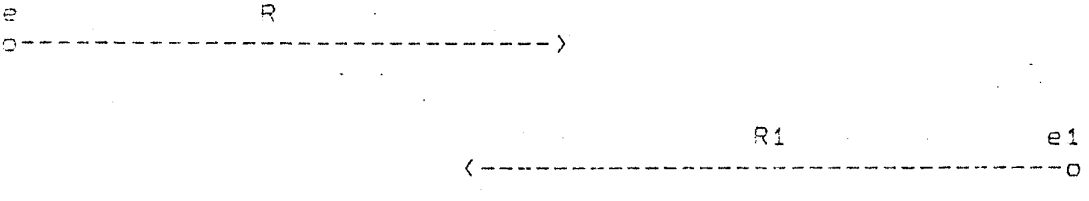


Fig. 10. Caso 3.

O último caso é mais complicado que os descritos acima. Nesse caso existem dois eventos, um que inicia um relacionamento e outro que termina um outro relacionamento, os dois relacionamentos são incompatíveis. Desta maneira, devem ser deduzidos dois eventos, um que termine o primeiro relacionamento e outro que inicie o segundo, isto é, a regra é quase uma

combinação das regras definidas acima. Esse caso é descrito na Fig. 10 e a regra associada é a seguinte:

```
final(depois(E,R)) (= inic(antes(E1,R1)),
começo(antes(E1,R1)), inic(antes(E1,R1))),
fim(depois(E, R), final(depois(E, R))). Se é_válido(depois(E,R)),
                                          é_válico(antes(E1,R1)),
                                          incompativel(R, R1),
                                          E < E1,
                                          não(quebra(E, R, E1)).
```

As regras ou axiomas definidos(as) até o momento podem ser usados para determinar os períodos de tempo nos quais um relacionamento é válido mas às vezes é necessário saber se um relacionamento também é válido em um determinado instante de tempo. Para isso, são úteis regras adicionais que facilitam esse tipo de dedução. Essas regras são:

```
é_válidoem(R, T). Se é_válido(depois(E, R)),
                     em(T, depois(E, R)).

é_válido(R, T) Se é_válido(antes(E, R)),
                  em(T, antes(E, R)).

em(T, P) Se começo(P, E1), fim(P, E2),
            tempo(E1, T1), tempo(E2, T2),
            T1 < T, T < T2.
```

No apêndice A será listada uma implementação do cálculo de eventos, escrito em ARITY/PROLOG [Arit86, Marc86]. Este programa foi testado para exemplo do histórico dos professores. O programa utiliza uma rotina de comparação de datas desenvolvida por A. L. Furtado.

3.3.1 Uma Simplificação do Cálculo de Eventos

Para facilitar a comparação com outras propostas de raciocínio temporal e para ressaltar o seu poder de generalização, mostraremos um caso especial do cálculo de eventos no qual são consideradas as seguintes condições:

- Os eventos são associados a tempos definidos;
- Os eventos são registrados na ordem na qual eles ocorrem;
- O banco de dados tem um registro completo de todos os eventos passados relevantes (a relação "(=" define uma ordem total)

Neste caso as regras definidas para o cálculo de eventos podem ser simplificadas. Por exemplo, todos os relacionamnetos 'U' podem ser derivados do termo **é_válido(antes(E,U))** - o

predicado é_válido(antes(E, U)) já não será mais necessário. As regras para derivar pontos extremos explícitos também não serão mais úteis.

O cálculo de eventos fica portanto resumido às seguintes regras:

Regra 1
é_válido(depois(E, R)) **SSe** inicia(E, R) **Regra de iniciação**

Regra 2
é_válidoem(R, T) **Se** é_válido(depois(E, R)),
em(T, depois(E, R)).

Regra 3
em(T, depois() **Se** tempo(E1, T1),
T1 < T,
não(fim(depois(E, R), E1),
tempo(E2, T2),
T <= T2).

S3 é utilizada no caso do intervalo não ter um evento que o termine.

Regra 4
começo(depois(E, R), E).

Regra 5
fim(depois(E, R), E1) **Se** é_válido(depois(E, R)),
termina(E1, R),
E < E1,
não(termina(E2, R)
E < E2 <E1).

Regra 6
em(T, P) **Se** começo(P, E1), fim(P, E2),
tempo(E1, T1), tempo(E2, T2),
T1 < T < T2.

3.3.2 Uma Extensão do Cálculo de Eventos para Tratar o Tempo de Transação

Como já foi analisado o cálculo de eventos suporta o tempo válido, isto é, o tempo no qual os eventos acontecem. Portanto, o cálculo de eventos aplicado a bancos de dados formaliza um banco de dados histórico no qual erros do passado não podem ser corrigidos.

Para que em um SGBD possam ser definidos bancos de dados temporais, o cálculo de eventos deve ser estendido para suportar o tempo de transação. Sripada [Srip88] define uma maneira de estender o cálculo de eventos, que será descrita nesta seção.

Todo evento é registrado no banco de dados através de uma

transação. Sendo assim, uma transação pode ser considerada um **meta-evento**. No momento em que um evento **e** for registrado no banco de dados, através de uma transação **tr**, será iniciado um período de tempo no qual acredita-se que esse evento aconteceu na vida real. Este período é chamado de **período de crença** e termina quando uma mudança de conhecimento acontece, isto é, quando já não se acredita mais que esse evento aconteceu. Portanto, esse conhecimento deve ser corrigido.

O predicado **base(TR, E)** é utilizado para expressar que a base para acreditar que o evento **E** aconteceu foi a transação **TR**.

O predicado **corrige(E, E1)** é utilizado para terminar um período de crença e define que a informação associada ao evento **E** corrige a informação associada ao evento **E1**. Para fazer essa correção não foi necessário a remoção do conhecimento anterior, o que permite a definição de consultas retroativas.

Associado ao símbolo predicativo **base** sempre deve ser utilizado o símbolo predicativo **tempo**, que define a data em que foi registrada a transação. Por exemplo, se um evento **e10** for registrado, então além das **asserções** que explicam os eventos, as seguintes asserções devem ser adicionadas ao banco de dados:

```
base(tr5, e10)
tempo(tr5, 21/10/89).
```

Com a adição do suporte do tempo de transação no cálculo de eventos é possível responder a consultas do tipo: "Qual era o cargo de João a 5 de Janeiro de 1989 segundo o conhecimento existente no banco de dados a 10 de Janeiro do mesmo ano?". Esta consulta segundo Sripada deve ser feita em dois passos:

- Quais eventos eram válidos a 10 de Janeiro? (Aqueles cujo período de crença não tinham terminado).
- Do conjunto de eventos anterior, o que pode ser inferido sobre o cargo de João a 5 de Janeiro?.

O segundo passo pode ser solucionado através do tratamento do tempo feito pelo cálculo de eventos (utilizando as regras originais). Mas, para poder dar resposta ao primeiro passo são necessárias novas regras que considerem o tempo de transação e as regras originais do cálculo de eventos devem ser mudadas.

Estas regras novas devem formalizar a dedução de períodos de crença, da mesma maneira que as regras originais deduzem períodos de tempo onde os relacionamentos são válidos. Portanto, as novas regras podem ser deduzidas de maneira similar.

Já que cada transação inicia ou termina mais de um período de crença, é possível utilizar o predicado,

```
depois(Tr, E)
```

para representar o período de crença iniciado pela transação **Tr** e no qual acredita-se que o evento **E** aconteceu. O predicado **antes(Tr, E)** é definido de maneira similar.

Por motivos de generalidade é definido um novo símbolo predicativo, **bé_válido** **bé_válido(E, P)**, expressa o fato do evento **E** ser válido no período de crença **P**. O prefixo b é utilizado para diferenciá-lo do símbolo utilizado originalmente no cálculo de eventos.

Sripada encontra duas diferenças entre o tratamento do tempo válido e o tempo de transação:

- As transações são registradas em ordem cronológica ascendente, o que implica que em qualquer ponto do tempo existe um registro completo das transações que tem acontecido. Este não é o caso dos eventos.

- Os períodos de crença não podem ser terminados sem ter sido definidos anteriormente. No caso de períodos definidos pelos eventos isto não acontece.

Desta maneira, uma só regra pode ser definida para deduzir períodos de crença (as regras, devidas às condições acima, são semelhantes ao caso simplificado do cálculo de eventos),

Regra 1

```
bé_válido(E, depois(Tr, E)) Se base(Tr, E),
                               não(corrige(E1, E),
                                   base(Tr1,E1),
                                   Tr1 (= Tr).
```

É suficiente considerar períodos de crença persistindo em direção ao futuro.

Para derivar que um evento **E** é válido em um instante **T** é necessário que **T** esteja dentro de um período de crença de **E**. Este raciocínio pode ser formalizado pelas regras.

Regra 2

```
bé_válido(E, T) Se bé_válido(E, depois(Tr, E)),
                   dentro_de(T, depois(Tr, E)).
```

Regra 3

```
dentro_de(T, P) Se bComeço(P, Tr1),
                   bFim(P, Tr2),
                   tempo(Tr1, T1),
                   tempo(Tr2, T2),
                   T1 ( T ( T2.
```

Os símbolos predicativos **bComeço** e **bFim** definem o início e o fim de um período de crença. No caso de não ser conhecido o fim do período de crença, a regra 3 deve ser mudada da seguinte

maneira:

Regra 4

```
dentro_de(T, P) Se bComeço(P,Tr1),
                   tempo(Tr1, T1),
                   T1 < T,
                   não(bFim(P,Tr2))
```

A definição do **Começo** e o **fim** de um período de crença é dado pelas seguintes regras:

```
                bComeço(depois(Tr, E), Tr).
bFim(depois(Tr, E), Tr1) Se base(Tr1, E1), corrige(E1,E).
```

Com as regras anteriores é possível fazer tanto atualizações proativas quanto retroativas. Entretanto, já que a crença em um evento e as inferências relacionadas, podem mudar com o tempo, a crença na validez de um relacionamneto R no período **depois(E,R)** depende do período de crença do evento. Desta maneira, todos os símbolos predicativos do cálculo de eventos devem ser mudados. Por exemplo, o símbolo predicativo **é_válido** deve ter um parâmetro adicional que represente o tempo de transação que registrou o evento asssociado.

No apêndice B temos uma listagem do cálculo de eventos estendido, escrito em ARITY/PROLOG [Arit86, Marc86]. Este programa foi testado em sua totalidade.

3.3.3 Um Exemplo da utilização do Cálculo de Eventos Aumentado

Para mostrar a utilização do cálculo de eventos aumentado, será utilizado o exemplo descrito por Snodgrass & Ahn [SnAh86]. Neste exemplo é feito um seguimento histórico dos cargos pelos quais passou uma professora chamada "Merrie". A sequência de eventos é a seguinte:

- Em setembro de 1973 Merrie foi contratada como professora auxiliar (evento e1). Para registrar este evento devem ser adicionados os seguintes fatos ao banco de dados:

```
base(tr1, e1).
tempo(tr1, 1/9/73).     /* Tempo de transação */
tempo(e1, 1/9/73).      /* Tempo válido */
ato(e1, contrato).
ator(e1, merrie)
cargoNovo(e1, auxiliar).
```

Dos axiomas anteriores podem ser deduzidos os seguintes fatos:

```
bé_válido(e1, depois(tr1, e1)).
bComeço(depois(tr1, e1), tr1) e
```

bé_válido(e1, t) para qualquer tempo t, após acontecer e1 (o período de crença estende-se indefinidamente)

Uma consulta, como, "Qual é o cargo de Merrie?", feita em outubro de 1973, deveria ser tracuzida no cálculo de eventos, como

? pé_válicoem(depois(E,posição(merrie,Y)), 1/10/1973).

O resultado seria E = e1, Y = auxiliar

Para facilitar este tipo de consultas pode ser definido um novo predicado, **consulta**, da seguinte maneira

consulta(R, T) **Se** pé_válido(depois(E,R),T).

Merrie foi promovida para professora titular em dezembro de 1973

base(tr2, e2).
tempo(tr2, 1/12/73).
tempo(e2, 1/12/73).
ato(e2, promoção).
ator(e2, merrie)
cargoVelho(e2, auxiliar).
cargoNovo(e2, titular).

Neste caso o predicado, bé_válido(e1,t), continuará tendo um período de crença infinito. Também pode ser inferido que,

pé_válido(depois(e2, posição(merrie, professor_titular)),
depois(tr2, e2)),

isto é, o cargo de Merrie é professor titular para o intervalo infinito que começa em 1/12/73.

Se em Janeiro de 1974 fosse acessado o banco de dados, através da consulta, "Qual é o cargo de Merrie?". Esta deveria ser tracuzida no cálculo de eventos como

? consulta(posição(merrie,X), 1/1/74).

O resultado seria: X = titular.

Entretanto, se em janeiro de 1974, fosse feita a seguinte: "Qual foi o cargo de Merrie no último mes de outubro?", seria necessária a definição de um novo predicado, **Consultativa**, que facilite a definição de consultas retroativas e proativas. **Consultativa** foi definido em ARITY/PROLOG, da seguinte maneira:

```
/* Acha eventos válidos em Tr */

consultativa(R, Tr, T) :- findall(E, bé_válidoem(E,Tr), L),
                          processa(L,T,R).
```

```
/* Do conjunto de eventos, o que pode ser inferido em T */

processa([], T, R) :-
        write("Nao existe relacionamento valido"), !.

processa([E|L],T,R) :- pe_validoem(depois(E,R),T), !.

processa([E|L],T,R) :- processa(L,T,R).
```

Se Tr >= T a consulta definida é retroativa e se Tr < T a consulta é proativa. Desta maneira a consulta acima seria definida da seguinte maneira:

```
? consultativa(posição(merrie,X), 1/1/74, 1/10/73).
```

A resposta seria: X = titular.

Suponha que foi encontrado um erro no banco de dados e que Merrie não foi promovida para professor titular e sim para professor adjunto. O erro foi corrigido em Fevereiro de 1974.

```
corrige(e3, e2).
base(tr3, e3).
tempo(tr3, 1/2/74).
tempo(e3, 1/12/73).
ato(e3, promoção).
ator(e3, merrie).
cargoVelho(e3, auxiliar).
cargoNovo(e3, adjunto).
```

Neste caso, o predicado, bé_válido(e2,t), deixará de ser válido devido ao predicado **corrige**. Ele termina o período de crença associado ao evento e2.

Se fosse feita a pergunta, "Qual era o cargo do merrie em janeiro de 1974?". Traduzida para o cálculo de eventos,

```
? consulta(posição(merrie,X),1/1/74)
```

A resposta seria: X = adjunto.

Mas se a pergunta fosse, "Qual, pensava-se, que era o cargo de Merrie em janeiro de 1974?". Traduzida para o cálculo de eventos,

```
? consultativa(posição(merrie, X), 1/1/74, 1/1/74).
```

A resposta seria: X = titular.

Em dezembro de 1978, Merrie é promovida a professor titular, retroativo desde junho de 1978.

```
base(tr4, e4)
tempo(tr4, 1/12/78).
tempo(e4, 1/06/78).
ato(e4, promoção).
ator(e4, merrie).
cargoVelho(e4, adjunto).
cargoNovo(e3, titular).
```

Se depois desta atualização retroativa, fosse feita a seguinte pergunta, "Qual era o cargo de Merrie em Outubro de 1978, como era conhecido em novembro de 1978?". Traduzida para o cálculo de eventos,

```
? consultativa(posicao(merrie, X), 1/11/78, 1/10/78).
```

A resposta seria: X = adjunto.

Se a pergunta fosse mudada para, "Qual era o cargo de Merrie em outubro de 1978?". Traduzida para o cálculo de eventos,

```
? consulta(posição(merrie, X), 1/10/78).
```

A resposta seria: X = titular.

4. OUTRAS FORMAS DE RACIOCÍNIO TEMPORAL E SUA COMPARAÇÃO COM O CÁLCULO DE EVENTOS

Nesta seção serão analisados alguns dos trabalhos da área de raciocínio temporal. Estes servirão de base para a busca de possíveis extensões a serem feitas no cálculo de eventos e também para poder avaliar o verdadeiro potencial deste. Alguns destes trabalhos são semelhantes ao cálculo de eventos. Por exemplo, o cálculo de Lee et alii [LeCC85] e a lógica de Allen [Alle83, Alle84].

Além dos trabalhos que serão mencionados nesta seção, existem uma série de trabalhos, bastante interessantes que deveriam ser mencionados, por exemplo, o trabalho do projeto PROBE na Computer Corporation of America [DBGH85] e alguns outros relacionados em [Snod86].

4.1. Lógica do Tempo e Eventos de Lee, Coelho e Cotta [LeCC85]

Este trabalho está orientado ao desenvolvimento de sistemas que permitam representar e raciocinar acerca de informações dependentes do tempo. Os sistemas tratados por eles eram especificamente comerciais [LeCC85]. O projeto faz parte de um projeto maior, chamado CANDID, que está relacionado com a modelagem de bancos de dados administrativos. Ele foi implementado em PROLOG (lógica de primeira ordem) mas está baseado na lógica temporal de Rescher e Urquhart [ReUr71] e a lógica de mudanças de von Wright [vonW65].

A característica mais importante desta proposta é a separação das mudanças genéricas, de tempos absolutos ou definidos. Por exemplo, é possível processar eventos do tipo,

João foi promovido de professor auxiliar a adjunto em 1980

onde o tempo, não está totalmente definido.

Na seção seguinte será descrita a notação utilizada por este modelo.

4.1.1. Notação

Lee et ali, para ganhar generalidade, evitam representar os relacionamentos temporais através de um predicado no qual um dos argumentos é o tempo em que este é válido. Para isso, é utilizada notação,

R(t):p

para indicar que o relacionamento p é válido no tempo t. Esta representação é equivalente à notação, é_válido-em(p, t) do cálculo de eventos.

Para representar mudanças independentes do tempo em que elas ocorrem é utilizado o seguinte termo:

(R1 ! R2)

O operador "!" indica que o relacionamento R2 acontece após a terminação do relacionamento R2. Por exemplo:

(posição(joão, professor_auxiliar) !
posição(joão, professor_adjunto)

Esta notação é importante para representar a ocorrência de um evento específico. Por exemplo, se associamos um tempo ao tipo de evento anterior,

R(data(1/1/76)): (posição(joão, professor_auxiliar) !
posição(joão, professor_adjunto)

estaria sendo representado o evento que aconteceu nessa data e que representa a promoção de João.

O termo (R1 ! R2) poce ser abreviado para representar outro tipo de fatos, por exemplo, o termo,

(* ! R)

descreve o tipo de evento que inicia R e termina "não R". A notação "*" representa a negação do outro argumento neste caso R. Por exemplo,

(* ! posição(joão, professor_auxiliar))

descreve o evento no qual joão foi contratado como professor auxiliar, mas sem conhecer a sua posição anterior.

Da mesma maneira, o termo,

(R ! *)

descreve o tipo de evento que termina R e inicia "não R".

A forma anterior de ver o tempo como um conjunto discreto de momentos não se ajusta a aplicações mais realistas onde um conceito de tempo contínuo seria mais adequado [LeCC85]. Entretanto, muitas das atividades do mundo real consistem da execução de eventos discretos. Portanto, Lee et alii assumem que a visão básica do tempo é continua consistindo de uma linha continua de pontos (instantes). Esta visão é semelhante à definida no cálculo de eventos.

Desta maneira, os intervalos de tempo consistem de um número infinito de momentos, onde os pontos de maior interesse são os pontos extremos (inicio e fim). Sendo assim, o predicado,

RD(d):(R1!R2)

representa um evento do tipo R1 ! R2 que ocorreu em algum momento durante o intervalo d.

4.1.2. Regras gerais do Cálculo de Lee et alii

Estas regras descrevem as propriedades dos relacionamentos e da persistência destes.

Regra 1
R(T):P **Se** R(T0):P, T0 .(=. T, não_muda(T0, T, P)

.(=. é o equivalente ao (=, mas é utilizado somente para comparar pontos no tempo. O predicado **não_muda(T0, T, P)** é utilizado para afirmar que P não muda no intervalo de T0 a T.

Esta regra enuncia que um relacionamento é válido no tempo T se P for válido em um tempo anterior T0 e ele não for terminado no intervalo de T0 a T.

Regra 2
R(T):P **Se** R(T0):(Q!P),
T0 (= T,
não_muda(T0, T, P)

Esta regra define que um relacionamento P é válido no tempo T, se P for iniciado no tempo T0 e T0 for um tempo anterior a T e P não for terminado no intervalo definido de T0 a T.

Regra 3
R(T) P **Se** RD(D):(Q!P),
fim(D) .(= T,
não_muda(fim(D), T, P)

Esta regra define que um relacionamento P é válido no tempo T, se P for iniciado em algum tempo durante um intervalo D e o fim de D for antes ou igual a T e P não for terminado no intervalo que vai do fim de D a T.

A Regra 4 descreve o predicado **não_muda**

Regra 4
não_muda(T0, T1, P) **Se** não(RD(D):(P!Q), P <> Q,
T0 .(início(D),
início(D) .(T1),
não(R(T):(P!Q), P <> Q,
T0 (= T, T .(= T1)

Aqui **não** é interpretado como negação por falha

4.1.3. Problemas Relacionados ao uso do Cálculo de Lee et alii

Sadri [Sadr87], enumera alguns problemas encontrados no cálculo de Lee e que estão relacionados com as regras 3 e 4

A regra 3, por exemplo, não considera a possibilidade da propriedade p terminar antes do fim do intervalo d. Isto é, dada a seguinte informação,

RD(data(1988):(Q ! P)) (1)

e usando a regra 3 e regra 4 é possível concluir que:

R(data(1/1/89)): P (2)

Agora se for processada a informação de que o predicado P termina o 27/12/88. Portanto,

R(data(27/12/88)):(P ! *) (3)

ainda é válido, sabendo que P é válido em 1/1/89, o que é uma contradição.

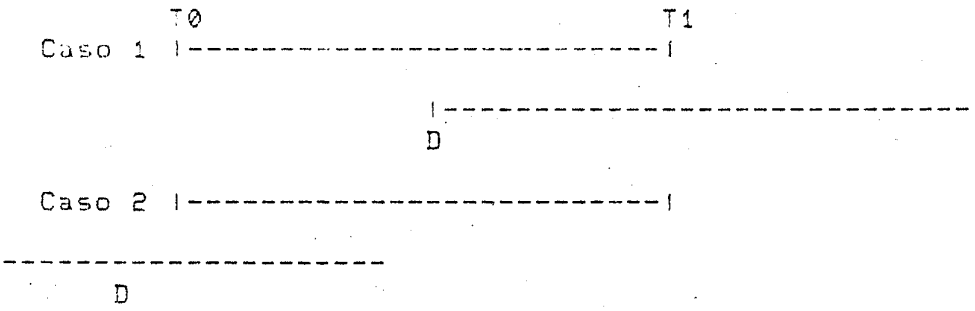
Além do anterior, a regra 3 não permite que um novo conhecimento mais preciso da data de ocorrência de um evento para o qual inicialmente existia uma data indefinida associada, possa ser asimilado. Por exemplo, suponha que o evento "Q ! P" ocorreu exatamente na data 1/4/88, então é possível representar este fato como

R(data(1/4/88)) (Q ! P) (4)

Não obstante, o cálculo não permite inferir que as duas

ocorrências (1) e (4) referenciam o mesmo evento.

No caso da regra 4, ela trata dois casos bastante similares de maneiras diferentes. Por exemplo, sejam as seguintes situações



e sejam as seguintes afirmações:

RD(D) : (P ! Q), P <> Q.

Usando a regra 4 é possível provar que **não-Muda(T0,T1,P)** no Caso 2, mas isto é impossível no Caso 1.

4.1.4 Considerações finais sobre o Cálculo de Lee et alii

O cálculo de Lee at ali é muito próximo ao caso especial do cálculo de eventos definido na seção 3.2.7. Sadri [Sadr87] mostra muito bem essas semelhanças. Elas permitem deduzir algumas das características do cálculo de Lee et alii.

- Ele é baseado no conceito de eventos e mudanças.
- Os eventos devem ser registrados na ordem em que eles acontecem. Esta característica pode ser relaxada definindo algumas extensões ao cálculo.
- Os eventos têm tempos associados que podem ser definidos ou indefinidos.
- As propriedades (relacionamentos) persistem no futuro a menos que seja provado que foram terminadas (Prova por falha)
- Lee et alii [LeCC85] não discutem a possibilidade da descrição incompleta de eventos. Sadri [Sadr87] mostra uma maneira de estender este cálculo para permitir esta descrição.

4.2 Lógica Temporal de Allen [Alle84]

Este formalismo da mesma forma que o cálculo de eventos é definido na lógica clássica de primeira ordem e é baseado nos

intervalos de tempo. Mas ao contrário dos outros, ele foi implementado em LISP. Este formalismo é utilizado como ferramenta para análise, entre outros conceitos, de eventos, ações, crença, intenção e causalidade.

Nesta seção será analisada só aquela parte da abordagem de Allen que é utilizada para representar e raciocinar acerca de eventos, intervalos de tempo e propriedades dependentes do tempo.

4.2.1. Notação

Para descrever a notação utilizada por Allen, além de algumas de suas características, utilizaremos o exemplo da seção 3.1.

Um evento na lógica de Allen pode ser descrito através do predicado **Ocorre**. Por exemplo, a primeira sentença do exemplo da seção 3.1. pode ser representado pelo termo,

ocorre(contrato(João, professor_auxiliar), T1)

O predicado Ocorre(E, T) é interpretado como indicando que o evento E ocorre sobre todo o intervalo T, isto é, não existe algum sub-intervalo próprio de T no qual E acontece. O anterior pode ser representado pela seguinte regra:

Regra 1
não(ocorre(E, T1)) **Se** ocorre(E, T), T1 em T.

Como pode ser visto esta abordagem não trabalha com o conceito de pontos no tempo. Em todos os casos a manipulação é feita a nível de intervalos de tempo. Para efeitos desta manipulação ele define, uma série de operadores relacionais como por exemplo, (,), =, "Encosta" ("Meets"), "Durante", etc.

Como no cálculo de eventos, na lógica de Allen as descrições dos eventos são usadas para deduzir a validade de um relacionamento em um intervalo de tempo. Por exemplo, se um evento de contratar X em um cargo Y ocorre no intervalo T, então a posição de X é Y em um intervalo T1 tal que T **encosta em** T1. Mais formalmente,

Regra 2
∃ T1[T encosta em T1,
é_válido(posição(X, Y), T1)] **Se** ocorre(contrato(X, Y), T)

T **encosta em** T1, significa que T ocorre exatamente antes que T1 e portanto não se interceptam. Graficamente seria,

De uma maneira similar podem ser definidas outras regras específicas para a terminação e iniciação de relacionamentos.

Se um evento de demissão ou renuncia de X do cargo Y ocorre no intervalo de tempo T, então há um intervalo T1 que encosta em T no qual X está na posição Y. Formalmente:

Regra 3

∃ T1[T1 encosta em T,
é_válico(posição(X, Y), T1)] **Se** ocorre(renuncia(X, Y), T)

Se um evento de promoção de X do cargo Y para Z, ocorre no intervalo T então existe um intervalo T1 que encosta em T no qual X está na posição Y, e T encosta em um intervalo T2 no qual X esta na posição Z. Formalmente

Regra 4

∃ T1, T2[T1 encosta em T,
é_válido(posição(X, Y), T1),
T encosta em T2,
é_válido(posição(X,Z),T2)] **Se**
ocorre(promoção(X,Y,Z),T2)]

4.2.2 Algumas Características da lógica de Allen

Sadri [Sadr87] enumera uma série de características encontradads na lógica de Allen:

1) Como no cálculo de eventos, o passaco e o futuro são tratados simetricamnete. A partir da descrição de um evento é possível deduzir a terminação de um relacionamento que foi válido em um intervalo anterior ou a iniciação de um relacionamento que será válido no intervalo seguinte.

2) Descrições de eventos podem ser registrados em uma ordem diferente à ordem em que realmente estes aconteceram. As descrições de eventos podem ser vistas como atualizações ao banco de dacos e portanto as atualizações são aditivas.

3) Ao contrário do cálculo de eventos, se é_válido(R, T) for verdadeiro na lógica de Allen, então T não será necessariamente um período maximal para o qual R for válido, isto é, um relacionamento R é válido em um intervalo T sse R for válido para todos os sub-intervalos próprios de T. Formalmente:

Regra 5

é_válido(R, T) **SSe** ∀ T1[não(T1 "Em" T) v é_válido(R, T1)]

4) Não existem regras que permitem inferir se dois intervalos de tempo são iguais. Entretanto Sadri [Sadr87] mostra uma extensão da lógica do Allen para tratar este tipo de casos.

5) Existe uma diferença entre eventos e relacionamentos como foi definido nas regras 1 e 2. Se um evento E ocorre em um

intervalo T, então E não pode ocorrer em qualquer sub-intervalo de T. Mas se um relacionamento for válido sobre T, então R será válido em todos os sub-intervalos de T.

Para evitar problemas com a regra 1 e certo tipo de eventos que expressam continuidade (por exemplo João está caminhando), Allen introduz o conceito de processo. Este conceito não é necessário no cálculo de eventos.

6) Tanto no cálculo de eventos quanto no cálculo de Lee os relacionamnetos persistem através do raciocínio por falha. Allen acha tal persistência muito importante, mas não define nenhuma regra para a formulação de persistência. Sadri [Sadr87] mostra como a lógica de Allen, pode ser estendida para suportar persistência de relacionamentos.

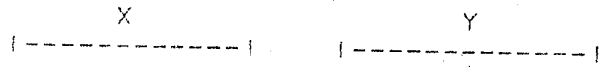
7) Eventos com uma descrição parcial são permitidos na lógica de Allen. Entretanto, não existem regras para completar por falha descrições parciais de eventos. Sadri [Sadr87] estende a lógica de Allen incorporando regras para o tratamento destes casos.

4 2 3 Uma Descrição Detalhada da Lógica de Allen

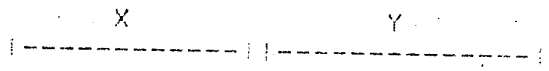
Nesta seção serão descritas regras mais gerais da lógica de Allen, e os operadores utilizados para relacionar intervalos de tempo.

Na lógica de Allen é considerada uma só linha de tempo. Dois intervalos sobre essa linha podem ser relacionados através de 13 operadores relacionais descritos a seguir:

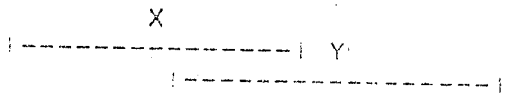
X antes Y (X (Y)



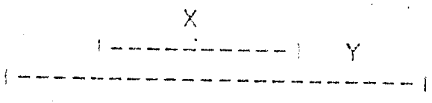
X encosta em Y



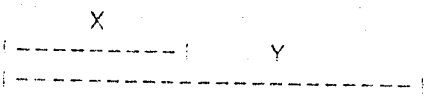
X intercepta Y



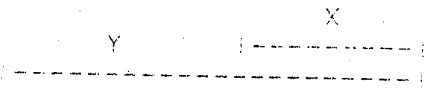
X durante Y



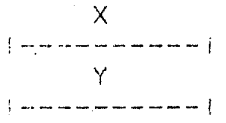
X inicia Y



X termina Y



X igual Y



Além dos operadores descritos acima existem operadores inversos para cada um dos seis primeiros operadores. Associadas a estes operadores existem uma série de regras que descrevem o seu comportamento transitivo. Por exemplo:

Regra 6

[T1 interseta T3 ou
T1 durante T3 ou
T1 inicia T3] **Se** T1 encosta T2, T2 durante T3

Regra 7

T1 antes T3 **Se** [T1 encosta T2, T2 durante T3]

Relacionamentos são definidos para serem válidos dentro de intervalos de tempo. Allen permite a representação de relacionamentos através de expressões lógicas complexas e para isso usa uma série de símbolos funcionais que representam os conectivos lógicos "e", "ou", "não" e os quantificadores ∀ e .

A validez de relacionamentos negativos para intervalos de tempo é definida pela seguinte regra:

Regra 8

é_válido(não(R), T) **SSe** ∀T1 [T1 em T **Se** não(é_válido(R, T1))]

onde,

T em T1 **SSe** [T inicia T1 ou
T durante T1 ou
T finaliza T1]

Relacionamentos disjuntivos são definidos como:

Regra 9

é_válido(ou(P, Q), T) **SSe**
∀T1 [T1 em T -> S [S em T1,
[é_válido(P, S) ou
é_válido(Q, S)]]

3 Trabalhos do Hanks & McDermott [HaMc85, HaMc87]

Um trabalho recente de Hanks & Mcdermott [HaMc85] formaliza, por falha, conceitos de raciocínio temporal, similares a esses formalizados no cálculo de eventos e na Lógica de Lee et alii. Os conceitos tratados são :

- A ocorrência de eventos no tempo;
- A iniciação de relacionamentos através de eventos;

- A validez de relacionamentos em intervalos de tempo, e
- A persistência de relacionamentos na ausência de informação que determine o contrário.

Nesse trabalho Hanks & Mcdermott [HaMc85] utilizam três lógicas diferentes de raciocínio para formalizar o caso de persistência por falha dos relacionamentos: A lógica não-monotônica de Mcdermott, a Lógica por falha de Reiter e a circunscrição de McCarthy.

Eles concluem que todas as três são inadequadas para representar o raciocínio por falha necessário, porque permitem resultados incompatíveis com suas regras.

Portanto, a solução deles é representar as regras através de procedimentos. Não obstante, uma das alternativas não considerada é expressar estas regras de raciocínio através de programação lógica (cláusulas de Horn), aumentada com negação por falha.

Sadri [Sadr87] considera interessante retomar o trabalho de Hanks & McDermott, formulando as regras temporais deles através da lógica de primeira ordem estendida com negação por falha. Ele considera que desta maneira seria possível resolver os problemas identificados por Hanks & Mcdermott.

4.4 Outros trabalhos

Nesta seção serão revisados outros trabalhos feitos na área de raciocínio temporal e principalmente que relacionam estas características com bancos de dados.

4.4.1 Modelagem de Dados Históricos e a Lógica do Tempo Preciso e Impreciso

Este trabalho sendo desenvolvido na Universidade Federal da Paraíba e dirigido pelo Prof. Ulrich Schiel [Mack86, Schi85a, 85b] pode ser dividido em três partes:

Análise de Dados Históricos e Modelagem de Eventos

O principal resultado desta parte foi o desenvolvimento do modelo semântico chamado THM. Este modelo é classificado como um modelo de dados binário [Schi83].

THM suporta o tempo válido, portanto, pode ser considerado como um modelo para BD's históricos. Neste modelo o tempo é tratado de forma simétrica e é possível inserir e deletar entidades tanto do passado quanto do futuro. A história é introduzida associando intervalos de tempo a entidades e relacionamentos. Mais detalhes deste modelo podem ser encontrados em [Schi83, FuNe86].

- Raciocínio Temporal

Esta parte da pesquisa utiliza como elemento básico o intervalo de tempo, seguindo a idéia da lógica de Allen.

Os intervalos são relacionados pelos símbolos predicativos **antes(I, J)** e **durante(I, J)** para os quais são definidas uma série de regras.

Com os símbolos predicativos mencionados acima e os símbolos funcionais **união(I + J)** e **diferença(I - J)**, são definidos novos símbolos predicativos semelhantes àqueles definidos por Allen, como por exemplo **intercepta(I, J)**, **encosta(I, J)**, **termina(I, J)**, etc.

Ao contrário da lógica de Allen, esta lógica de tempo define um símbolo funcional **duração(I)** que associa um número real a cada intervalo. Também é definida a distância entre dois intervalos e é definida uma nova classe de eventos chamados periódicos, para os quais são estendidos os símbolos predicativos **antes** e **durante**.

A lógica temporal é usada para a modelagem de eventos e seu interrelacionamento. Mais detalhes deste trabalho podem ser encontrados em [Schi85b].

- Tempo Impreciso

No caso da descrição parcial de eventos, este trabalho utiliza o conceito de tempo impreciso. Esta imprecisão é classificada em três categorias:

1) Completa ignorância. Quando não for conhecida nenhuma data dos pontos extremos de um intervalo de tempo.

2) Conhecimento parcial. Quando existem várias possibilidades de datas válidas para os extremos do intervalo.

3) Conhecimento relativo. Quando as datas são definidas em função de outras.

Maiores detalhes desta parte do trabalho podem ser encontrados em [Schi85a].

4.4.2. Um Ambiente para Desenvolvimento de BDDs Temporais

Este trabalho, sendo desenvolvido na Universidade Federal do Rio Grande do Sul pelos Profs. Nina Edelweiss e Antônio Carlos da Rocha Costa [EdCo88], não é exatamente um trabalho dentro da área de raciocínio temporal, mas são definidos alguns conceitos importantes para o desenvolvimento de Bancos de Dados Dedutivos. Este trabalho foi desenvolvido usando a linguagem de programação lógica PROLOG.

O objetivo deste trabalho foi a criação de um ambiente para o desenvolvimento de protótipos de BDD's temporais. O modelo de dados lógico utilizado para representar as informações tem como base a lógica das classes e das relações [Tars54].

No modelo adotado, devem ser definidas classes às quais pertencem as entidades e relações que podem ocorrer entre estas entidades. As classes são divididas em primárias e secundárias. As classes primárias representam todos aqueles símbolos predefinidos reconhecidos pelo PROLOG, por exemplo, os números reais e inteiros. Estas classes servem de base para a definição das classes secundárias ou classes definidas pelo usuário. Entre as classes secundárias existem as classes básicas e as classes derivadas. Estas últimas são definidas a partir das classes básicas, para isso sendo utilizados uma série de operadores como **união**, **interseção**, etc.

As relações são classificadas de maneira similar às classes. As relações primárias operam sobre as classes primárias e são símbolos predicativos pré-definidos pelo PROLOG, por exemplo: <, >, =, etc. As classes secundárias são aquelas definidas pelo usuário.

Da mesma maneira que no cálculo de eventos, nesta abordagem as atualizações são aditivas. Entretanto, neste caso só é suportado o tempo de transação. Portanto, todas as atualizações da base de dados são feitas na ordem temporal de sua ocorrência no mundo real e não é possível fazer atualizações a fatos referentes ao passado.

Cada vez que uma informação é inserida na base de dados, são adicionados os dados do dia e a hora da inserção. Uma remoção é feita através da inserção de termos que definem o tipo de operação e o fato que deixou de ser válido.

Nesta abordagem é permitida a definição de restrições de integridade, as quais podem ser estáticas ou dinâmicas. Para o tratamento de restrições foram adotadas as idéias apresentadas por [Cale87] e [Kowa79]. As restrições devem ser traduzidas em regras de inferência que no momento de serem satisfeitas, indicam a violação da integridade do banco de dados.

5. CONCLUSÕES E DIRETRIZES PARA TRABALHOS FUTUROS

Este relatório é o resultado de uma pesquisa bibliográfica feita na área de bancos de dados temporais e raciocínio temporal. O trabalho faz parte do projeto NICE. O objetivo do projeto NICE é investigar métodos e desenvolver protótipos de software que visem a ambientes que permitam o uso cooperativo de sistemas de informação.

Entretanto, esta revisão bibliográfica, é o início de um trabalho que visa a implementação de bancos de dados dedutivos

temporais. A utilização do cálculo de eventos, como já foi visto, parece promissora. Porém alguns trabalhos devem ainda ser feitos. Entre eles podem ser enumerados os seguintes:

- Tentar acoplar o cálculo de eventos àquelas partes do sistema NICE que já foram implementadas. Sendo assim, faz-se necessário melhorar a interface do cálculo de eventos para facilitar seu uso.

- Estudar a maneira de estender o cálculo de eventos para formalizar alguns conceitos que a lógica de Allen considera, tais como causalidade e intenção [Sadr87].

- Gerar uma formulação em programação lógica para a lógica temporal de Hanks & McDermott de maneira a reconciliar suas características procedimentais e declarativas. Sendo assim, uma comparação do cálculo de eventos com esta abordagem seria mais fácil e permitiria ver a maneira como o cálculo de eventos resolveria os problemas identificados por Hanks & McDermott [Sadr87].

- Definir uma boa massa de exemplos, diferentes daqueles existentes na literatura, que permitam uma maior familiaridade com o cálculo de eventos e que facilitem a descoberta de possíveis falhas da ferramenta ou melhoras a serem feitas nela.

As seguintes diretrizes estão mais relacionadas com o uso do PROLOG em bancos de dados. Já foi visto que o uso de programação lógica em bancos de dados é de grande importância. Porém, sistemas tais como PROLOG não tem todas as qualidades que um sistema de bancos de dados deveria ter. [BrJa86, PCGJ86, Melo88] listam alguns dos problemas que serão descritos a seguir:

- O uso de PROLOG para expressar regras envolve algumas desvantagens e o aspecto procedimental é uma delas. As regras não lineares, onde o predicado recursivo ocorre mais de uma vez no corpo, não são admitidas, por exemplo [Melo88, PCGJ86],

```
voa(X, Z) :- voa(X,Y), voa(Y,Z)
```

Esquemas, dependências funcionais, restrições de integridade, e explicações podem ser facilmente implementadas em PROLOG. Entretanto, há conceitos de bancos de dados que não são disponíveis em sistemas de programação lógica e cuja introdução não é trivial. Por exemplo:

Transações que permitam ao sistema PROLOG se recuperar das falhas;

O uso de tipos de dados em bancos de dados para facilitar a integridade semântica e a validação dos dados [BrJa86]. Uma maneira de implementar tipos é através de predicados, este é o caso do sistema analisado na seção 4.4.2 [EdCo88].

Entretanto, este suporte não é o suficientemente eficiente,

Indexação; e

Disparadores ("triggers").

Para solucionar os problemas acima é necessário trabalhar com meta-regras que permitam a manipulação de tipos, disparadores e definição do esquema. O projeto NICE investigará estes problemas.

No projeto NICE utiliza-se um sistema de banco de dados existente (no caso SQL/ARITY) acoplado a um sistema de programação lógica (ARITY/PROLOG). Um trabalho a ser feito é o estudo de interfaces mais eficientes entre estes dois sistemas principalmente em ambientes concorrentes ou dinâmicos [PCGJ86].

REFERÊNCIAS BIBLIOGRÁFICAS

[Alle83]
ALLEN, J.F. "Maintaining Knowledge About Temporal Intervals". **Communications of ACM**. 26(11):832-843, 1983.

[Alle84]
ALLEN, J.F. "Towards a General Theory of Action and Time". **Artificial Intelligence**, 23:123-154, 1984.

[Aria86]
ARIAV, G. "A Temporally Oriented Data Model". **ACM Transactions on Databases Systems**, 11(4):499-527, 1986

[Arit86]
ARITY, Corp. **"The Arity Prolog Programming Language"**. Arity Corporation, Ma., 1986. 190p.

[BADW82]
BOLOUR, A. & ANDERSON, T.L. & DEKEYSER, L.J. & WONG, H.K.T. "The Role of Time in Information Processing: A Survey". **ACM SIGMOD Review**, 12(3):27-50, 1982.

[Benz82]
BEN-ZVI, J. **"The Time Relational Model"**, Los Angeles, UCLA, 1982. (PhD. Dissertation).

[BoKo82]
BOWEN, K. & KOWALSKI, R.A. "Amalgamating Language and Metalanguage in Logic Programming". In: CLARK, K.L. & TARNLUND, S.A. (eds.). **"Logic Programming"**. Academic Press, New York, 1982.

[BrJa86]
BRODIE, M. & JARKE, M. "On Integrating Logic Programming and Databases". In: KERSCHBERG, L. (ed.) **"Expert Database Systems"**. Benjamin Cummings Publ. Co., 1986, Pags 191-207.

[CaLe87]
CASTILHO, J.M.V. & LEMOS, A.S. "A Construção de Protótipos Básicos de Sisttemas de Bancos de Dados: Uma Proposta". CPGC/UFRGS, Porto Alegre, 1987. (Relatório Técnico No. 78).

[ClWa83]
CLIFFORD, J. & WARREN, D.S. "Formal Semantics for Time in Databases". **ACM Transactions on Database Systems**, 8(2):214-254, 1983.

[Codd79]
CODD, E.F. "Extending the Database Relational Model to Capture More Meaning". **ACM Transactions on Database Systems**, 4(4):397-434, 1979.

[CoMa84]
COPELAND, G. & MAIER, D. "Making Smalltalk a Database System". In: **PROCEEDINGS OF ACM SIGMOD CONFERENCE ON MANAGEMENT OF DATA**, AUSTIN, TX., 1984. Pags. 316-325.

[DBGH85]
DAYAL, U. & et alii **"A Research Project in Knowledge Oriented Database Systems: Preliminary Analysis"**. Computer Corporation of America, July 1985. (Technical Report CCA-85-03). Pags 43-54.

[EdCo88]
EDELWEISS, N. & COSTA, A.C.R. "Um Ambiente para Desenvolvimento de Protótipos de Bancos de Dados Dedutivos Temporais". In: **ANAIS DO SIMPóSIO BRASILEIRO DE BANCOS DE DADOS**, 4, Campinas, SP., 1988, Pags. 163-173.

[Fros86]
FROST, R. **"Introduction to knowledge Base Systems"**, Collins, London, 1986. Caps. 5-6.

[FuNe86]
FURTADO, A.L. & NEUHOLD, E.J. **"Formal Techniques For Database Design"**. Springer Verlag, 1986.

[GaMN84]
GALLAIRE, H. & MINKER, J. & NICOLAS, J.M. "Logic and Databases: A Deductive Approach". **ACM Computing Surveys**, 16(2):153-185, 1984.

[GaVa88]
GARDARIN, G. & VALDURIEZ, P. **"Principles and Algorithms of Relational Database Systems"**, Addison Wesley, MA., 1988.

[Gols86]
GOLSHANI, F. "Specification and Design of Expert Database Systems". In: KERSCHBERG, L. (ed.) **"Expert Database Systems"**. Benjamin Cummings Publ. Co., 1986, Pags 369-381.

[GrMi83]
GRANT, J. & MINKER, J. "Answering Queries in Indefinite Database and the Null Value Problem". Computer Science Dept., University of Maryland, College Park, 1983. (Tech. Rep. 1374).

[HaMc81]
HAMMER, M. & McLEOD, D. "Database Description with SDM: A Semantic Database Model". **ACM Transactions on Database Systems**, 6(3):351-386,1981.

[HaMc85]
HANKS, S. & McDERMOTT, D. **"Temporal Reasoning and Default Logics"**. Computer Science, Yale University, 1985. (Research Report No. 430).

[HaMc87]
HANKS, S. & McDERMOTT, D. "Non monotonic Logic and Temporal Projection". **Artificial Intelligence**, 33(3):379-412, 1987

[KoSe86]
KOWALSKI, R.A. & SERGOT, M. "A Logic-based Calculus of Events", **New Generation Computing**. Ohmsa ltd. and Springer Verlag, 4(1):67-95, 1986.

[Kowa86]
KOWALSKI, R.A. **"Logic for Problem Solving**. North-Holland, New York, 1979.

[Kowa86]
KOWALSKI, R.A. **"Database Updates in the Event Calculus"**. Imperial College, Department of Computing, London, July 1986, revised 1989. (Research Report DoC 86/12).

[LeCC85]
LEE, R. M. & COELHO, H. & COTTA, J.C. "Temporal Inferencing on Administrative Databases". **Information Systems**, 10(2):197-206, 1985.

[Marc86]
MARCUS, C. "Prolog Programming". Addison Wesley, Reading, MA., 1986. 325p.

[McHa69]
McCARTHY, J. & HAYES, P.J. "Some Philosophical Problems from the Standpoint of Artificial Intelligence". In: MELTZER, D. & MICHIE, D. (eds.) **"Machine Intelligence"**, 4, Edinburgh University Press, Edinburgh, 1969, Pags 463-502

[Mcke86]
McKENZIE, E. "Bibliographic: Temporal Databases", **ACM SIGMOD Record**, New York, 15(4):40-52, 1986.

[Melo88]
MELO, R. N. **"Bancos de Dados Não Convencionais a Tecnologia de BD e suas Novas Áreas de Aplicação"**. VI Escola de Computação, Campinas, SP., 1988.

[PCGJ86]
PARKER Jr., D.S. & CAREY, M. & GOLSHANI, F. & JARKE, M. & SCIORE, E. & WALKER, A. "Logic Programming and databases". In: KERSCHBERG, L. (ed.) **"Expert Database Systems"**. Benjamin Cummings Publ. Co., 1986, Pags 35-47.

[Qui168]
QUILLIAN, M.R. "Semantic Memory". In: MINSKY, M. (ed.) **"Semantic Information Processing"**. Cambridge, MA., MIT Press, 1968, Pags 227-270.

[Reit84]
REITER, R. "Towards a Logical Reconstruction of Relational Databases Theory". In: BRODIE, M. & MYLOPOULOS, J. & SCHMIDT, J. (eds.), **"On Knowledge Base Management Systems"**, Springer Verlag, 1984, Pags 191-233.

[ReUr71]
RESCHER, N. & URQUHART, A. **"Temporal Logic"**. Springer Verlag, New York, 1971.

[Sadr87]
SADRI, F. "Three Recent Approaches to Temporal Reasoning". In: GALTON, A. (ed.). **"Temporal Logics and Their Applications"**. New York, Academic Press, 1987, Pags. 121-168.

[Schi83]
SCHIEL, U. "An Abstract Introduction to the Temporal Hierachic Data Model". In: "PROCEEDINGS OF INTERNATIONAL CONFERENCE ON VERY LARGE DATABASE", 9, Florence, Italy, 1983.

[Schi85a]
SCHIEL, U. **"Representation and Retrieval of Incomplete Data and Temporal Information"**. Universidade Federal da Paraiba, Oct. 1985. (Technical Report DSC-01 85).

[Schi85b]
SCHIEL, U. "The Time Dimension in Information Systems". In: **PROCEEDINGS OF THE IFIP WG 8.1 WORKING CONFERENCE ON THEORETICAL AND FORMAL ASPECTS OF INFORMATION SYSTEMS**, Barcelona, Spain, 1985. Pags. 67-76.

[Sern80]
SERNADAS, A. "Temporal Aspects of Logical Procedure Definition". **Information Systems**, 5:167-187, 1980.

[SnAh85]
SNODGRASS, R. & AHN, I. "A Taxonomy of Time in Databases". In: **PROCEEDINGS OF ACM SIGMOD CONFERENCE ON MANAGEMENT OF DATA**, AUSTIN, TX., 1985. Pags. 236-246.

[SnAh86]
SNODGRASS, R. & AHN, I. "Temporal Databases". **IEEE Computer**, New York, 19(9):35-42, 1986.

[Snod86]
SNODGRASS, R. "Research Concerning TTime in Databases: Project Summaries". **ACM SIGMOD Record**, 15(4):14-39,1986.

[Snod87]
SNODGRASS, R. "The Temporal Query Language TQuel". **ACM Transactions on Databases Systems**, 12(2):247-298.

[Srip88]
SRIPADA, S.M. "A Logical Framework for Temporal Deductive Databases". In: **PROCEEDINGS OF INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES**, 14, Los Angeles, 1988, Pags. 171-182.

[StRo86]
STONEBRAKER, M. & ROWE, L. "The Design of Postgres". In: **PROCEEDINGS OF ACM SIGMOD CONFERENCE ON MANAGEMENT OF DATA**, Washington, DC., 1986. Pags. 340-355.

[Tars54]
TARSKI, A. **"Introduction to Logic and to the Metthodology of Sciences"**. Oxford University Press, Oxford, 1954.

[TsLo82]
TSICHRITZIS, D.C. & LOCHOVSKY, F.H. **"Data Models"**. Prentice-Hall, Englewood Cliffs, N.J. 1982, 381p.

[vonW65]
VON WRIGHT, G.H. "And Next". **Acta Philosophica Fennica**, Fasc. XVIII, Helsinski, Pags. 293-301, 1965.

APÊNDICE A

UMA IMPLEMENTAÇÃO DO CÁLCULO DE EVENTOS

```
%-------------------------------------------------------------%
%              PRIMEIRA VERSAO CALCULO DE EVENTOS
%              DATA 1/6/89
%-------------------------------------------------------------%

%-------------------------------------------------------------%
%                  DEFINE COMAPARADORES DE DATAS
%-------------------------------------------------------------%

:- op(400,yfx,.<.).
:- op(400,yfx,.<=.).
:- op(400,yfx,.>=.).
:- op(400,yfx,.>.).

 .<.(T1,T2) :- date1(T1,D1),
               date1(T2,D2),
               D1 < D2.

 .<=.(T1,T2) :- date1(T1,D1),
                date1(T2,D2),
                D1 =< D2.

 .>=.(T1,T2) :- date1(T1,D1),
                date1(T2,D2),
                D1 >= D2.

 .>.(T1,T2) :- date1(T1,D1),
               date1(T2,D2),
               D1 > D2.

date1(T,D) :- arg(1,T,I),
              arg(2,T,J),
              arg(3,T,K),
              date(I,J,K,D).

%-------------------------------------------------------------%
%                          FATOS
%-------------------------------------------------------------%

act(e1, contrato).
actor(e1, maria).
destination(e1,auxiliar).
time(e1, d1(1970,5,10)).

act(e2, renuncia).
actor(e2, joao).
renunciation(e2,auxiliar).
time(e2, d2(1975,6,1)).

act(e3, renuncia).
actor(e3, maria).
```

```
renunciation(e3, adjunto).
time(e3, d3(1980,10,1)).

act(e4, promocao).
actor(e4, maria).
renunciation(e4, auxiliar).
destination(e4, adjunto).
time(e4, d4(1975,6,1)).

%-----------------------------------------------------------------%
%                    REGRAS ESPECIFICAS
%-----------------------------------------------------------------%

initiates(E, possesses(X, Y)) :- act(E, give),
                                 recipient(E, X),
                                 object(E, Y).

terminates(E, possesses(X, Y)) :- act(E, give),
                                  donor(E, X),
                                  object(E, Y).

initiates(E, posicao(X, Y)) :- act(E, contrato),
                               actor(E, X),
                               destination(E, Y).

initiates(E, posicao(X, Y)) :- act(E, promocao),
                               actor(E, X),
                               destination(E, Y).

terminates(E, posicao(X, Y)) :- act(E, renuncia),
                                actor(E, X),
                                renunciation(E, Y).

terminates(E, posicao(X, Y)) :- act(E, promocao),
                                actor(E, X),
                                renunciation(E, Y).

incompatible(posicao(X, Y), posicao(X, Y1)) :- not(Y == Y1).
incompatible(possesses(X, Y), possesses(X, Y1)) :- not(X == X1).

Y == Y.

%-----------------------------------------------------------------%
%                    REGRAS GERAIS
%-----------------------------------------------------------------%

holds(before(E, R)) :- terminates(E, R).
holds(after(E, R)) :- initiates(E, R).

start(after(E, R), E).
end(before(E, R), E).

intigual(after(E, U), before(E1, U)) :- holds(after(E, U)),
                                        holds(before(E1, U)),
                                        less(E, E1),
```

```
                                        not(broken(E, U, E1)).

start(before(E1, U), E) :- intigual(after(E, U), before(E1, U)).
end(after(E, U), E1) :- intigual(after(E, U), before(E1, U)).

broken(E, U, E1) :- holds(after(E2, U2)),
                    exclusive(U, U2),
                    less(E, E2),
                    less(E2, E1),!.

broken(E, U, E1) :- holds(before(E2, U2)),
                    exclusive(U, U2),
                    less(E, E2),
                    less(E2, E1).

exclusive(U, U)
exclusive(U, U1) :- incompatible(U, U1).

holdsat(U, T) :- holds(after(E, U)),
                 in(T, after(E, U)).

holdsat(U, T) :- holds(before(E, U)),
                 in(T, before(E, U)).

in(T, P) :- start(P, E1), end(P, E2),
            time(E1, T1), time(E2, T2),
            T1.<.T, T .<.T2.

start(before(E1, U1), init(before(E1, U1))) :- holds(before(E, U)),
                                               holds(before(E1, U1)),
                                               exclusive(U, U1),
                                               menor(E, E1),
                                               not(broken(E, U1, E1)).

lessequal(E, init(before(E1, U1))) :- holds(before(E, U)),
                                      holds(before(E1, U1)),
                                      exclusive(U, U1),
                                      less(E, E1),
                                      not(broken(E, U1, E1)).

end(after(E, U), fin(after(E, U))) :- holds(after(E, U)),
                                      holds(after(E1, U1)),
                                      exclusive(U, U1),
                                      less(E, E1),
                                      not(broken(E, U, E1)).

lessequal(fin(after(E, U)),  E1) :- holds(after(E, U)),
                                    holds(after(E1, U1)),
                                    exclusive(U, U1),
                                    less(E, E1),
                                    not(broken(E, U, E1)).

lessequal(fin(after(E, U)), init(before(E1, U1))) :- holds(after(E, U)),
                                                     holds(before(E1, U1)),
                                                     incompatible(U, U1),
```

```
                                                    less(E, E1),
                                                    not(broken(E, U, E1)).

start(before(E1, U1), init(before(E1, U1))) :- holds(after(E, U)),
                                               holds(before(E1, U1)),
                                               incompatible(U, U1),
                                               less(E, E1),
                                               not(broken(E, U, E1)).

end(after(E, U), fin(after(E, U))) :- holds(after(E, U)),
                                      holds(before(E1, U1)),
                                      incompatible(U, U1),
                                      less(E, E1),
                                      not(broken(E, U, E1)).

less(E, E1) :- time(E, X),
               time(E1, Y),
               X.<.Y.

lessequal(E, E1) :- time(E, X),
                    time(E1, Y),
                    X.<=.Y.

great(E, E1) :- time(E, X),
                time(E1, Y),
                X.>.Y.

greatequal(E, E1) :- time(E, X),
                     time(E1, Y),
                     X.>=.Y.

%-------------------------------------------------------------------%
%                  CONVERTE DATAS A DIAS
%-------------------------------------------------------------------%

date(I,J,K,D) :-
  (var(I), var(D),!,
  write('Erro, anos e dias variaveis');
  (var(D), !,
   anosbis(I,A,N),
   Days is (A * 366) + (N * 365),
   qualTab(I,T),
   X is J - 1,
   d_arg(X,T,V),
   D is V + K + Days;
   X is D // 365,
   (0 = D mod 365,!,
    I1 is 1900 + X - 1,
    anosbis(I1,A,N),
    (leap(I1), !,
     P is 366 - A;
     P is 365 - A
    ),
    I is I1,
    W is P;
```

```
      I1 is 1900 + X,
      anosbis(I1,A,N),
      P is D mod 365 - A,
      (P =< 0, !,
       I is I1 - 1,
       (leap(I1), !,
        W is 366 + P;
        W is 365 + P
       );
       I is I1,
       W is P
      )
     ),
     qualTab(I,T),
     Z is W // 30,
     d_arg(Z,T,V),
     (W > V, !, J is Z + 1, U = V;
      Y is Z - 1, d_arg(Y,T,G),
      (W > Q, !, J = Z, U = Q)
     ),
     K is W - U
    )
   )

d_arg(0,_,0) :- !.
d_arg(I,T,V) :- arg(I,T,V).

qualTab(I,T) :-
   (leap(I), !,
    T = t(31,60,91,121,152,182,213,244,274,305,335,366);
    T = t(31,59,90,120,151,181,212,243,273,304,334,365))

leap(I) :-
   0 is I mod 4,
   (0 is I mod 100, !, 0 is I mod 400;
    true).

anosbis(X,Y,Z) :-
   Tab = bis(0,1,2,2,3,4,4,5,6,6,7,8,8,9,10,10,11,12,12,13,14,14,15,16),
   A is X - 1900,
   Y2 is A // 4,
   (A > 100, !,
    W is A // 100,
    d_arg(W,Tab,D),
    Y1 is Y2 - D;
    Y1 is Y2
   ),
   (leap(X), !,
    Y is Y1 - 1;
    Y is y1
   ),
   Z is A - Y.
```

APÊNDICE B
UMA IMPLEMENTAÇÃO DO CÁLCULO DE EVENTOS ESTENDIDO

```
%-------------------------------------------------------------%
%              SEGUNDA VERSAO CALCULO DE EVENTOS
%              INCORPORA O TEMPO DE TRANSACAO
%              DATA 26/6/89
%-------------------------------------------------------------%

%-------------------------------------------------------------%
%                 DEFINE COMAPARADORES DE DATAS
%-------------------------------------------------------------%

:- op(400,yfx,.<.).
:- op(400,yfx,.<=.).
:- op(400,yfx,.>=.).
:- op(400,yfx,.>.).

 .<.(T1,T2) :- date1(T1,D1),
               date1(T2,D2),
               D1 < D2.

 .<=.(T1,T2) :- date1(T1,D1),
                date1(T2,D2),
                D1 =< D2.

 .>=.(T1,T2) :- date1(T1,D1),
                date1(T2,D2),
                D1 >= D2.

 .>.(T1,T2) :- date1(T1,D1),
               date1(T2,D2),
               D1 > D2.

date1(T,D) :- arg(1,T,I),
              arg(2,T,J),
              arg(3,T,K),
              date(I,J,K,D).

%-------------------------------------------------------------%
%                          FATOS
%-------------------------------------------------------------%

base(tr1,e1).
tempo(tr1, d1(1985,4,1)).
ato(e1, contrato).
ator(e1, maria).
cargonovo(e1,auxiliar).
tempo(e1, d1(1985,4,1)).

base(tr2,e2).
base(tr2,e3).
tempo(tr2, d2(1985,6,1)).
ato(e2,promocao).
ator(e2,maria).
```

```
cargovelho(e2,auxiliar).
cargonovo(e2,adjunto).
tempo(e2, d3(1985,8,1)).
ato(e3,promocao).
actor(e3,maria).
cargovelho(e3,adjunto).
cargonovo(e3, titular).
tempo(e3, d4(1987,8,1)).

base(tr3,e4).
tempo(tr3, d5(1985,10,1)).
ato(e4, promocao).
actor(e4, maria).
cargovelho(e3, auxiliar).
cargonovo(e4, titular).
tempo(e4, d6(1985,5,1)).
corrige(e4, e2).
corrige(e4, e3).

%-----------------------------------------------------------------%
%                    REGRAS ESPECIFICAS
%-----------------------------------------------------------------%

inicia(E, posicao(X, Y)) :- ato(E, contrato),
                            ator(E, X),
                            cargonovo(E, Y).

inicia(E, posicao(X, Y)) :- ato(E, promocao),
                            ator(E, X),
                            cargonovo(E, Y).

termina(E, posicao(X, Y)) :- ato(E, renuncia),
                             ator(E, X),
                             cargovelho(E, Y).

termina(E, posicao(X, Y)) :- ato(E, promocao),
                             ator(E, X),
                             cargovelho(E, Y).

incompativel(posicao(X, Y), posicao(X, Y1)) :- not(Y == Y1).

Y == Y.

%-----------------------------------------------------------------%
%                    REGRAS GERAIS
%-----------------------------------------------------------------%

be_valido(E, depois(Tr, E)) :- base(Tr, E),
                               not(corrige(E1, E),
                                   base(Tr1, E1),
                                   menorigual(Tr1, Tr)
                                  ).

be_validoem(E, T) :- be_valido(E, depois(Tr, E)),
                     dentrode(T, depois(Tr, E)).
```

```
dentrode(T, P) :- bcomeco(P, Tr1),
                  bfim(P, Tr2),
                  tempo(Tr1, T1),
                  tempo(Tr2, T2),
                  T1.<.T,
                  T.<.T2.

dentrode(T, P) :- bcomeco(P, Tr1),
                  tempo(Tr1, T1),
                  T1.<.T,
                  not(bfim(P, Tr2)).

bcomeco(depois(Tr, E), Tr)

bfim(depois(Tr, E), Tr1) :- base(Tr1, E1),
                            corrige(E1, E).

pe_valido(antes(E, R),P) :- termina(E, R),
                            be_valido(E, P).

pe_validoem(antes(E, R), T) :- pe_valido(antes(E, R), P),
                               dentrode(T, P).

pe_valido(depois(E, R), P) :- inicia(E, R),
                              be_valido(E, P).

pe_validoem(depois(E, R), T)) :- pe_valido(depois(E, R), P),
                                 dentrode(T, P).

comeco(depois(E, R), E, Tr).
fim(antes(E, R), E, Tr).

intigual_emTr(depois(E, U), antes(E1, U), Tr) :- pe_valido(depois(E,U),Tr),
                                                 pe_valido(antes(E,U),Tr),
                                                 menor(E, E1),
                                                 not(quebra(E, U, E1, Tr)).

comeco(antes(E1, U), E, Tr) :- intigual_emTr(depois(E, U), antes(E1, U),Tr).
fim(depois(E, U), E1, Tr) :- intigual_emTr(depois(E, U), antes(E1, U),Tr).

quebra(E, U, E1, Tr) :- pe_validoem(depois(E2, U2), Tr),
                        exclusivo(U, U2),
                        menor(E, E2),
                        menor(E2, E1), !.

quebra(E, U, E1, Tr) :- pe_validoem(antes(E2, U2), Tr),
                        exclusivo(U, U2),
                        menor(E, E2),
                        menor(E2, E1).

exclusivo(U, U).
exclusivo(U, U1) :- incompativel(U, U1).

comeco(antes(E1, U1), inic(antes(E1, U1)), Tr) :-
```

```
                                        pe_validoem(antes(E, U), Tr),
                                        pe_validoem(antes(E1, U1), Tr),
                                        exclusivo(U, U1),
                                        menor(E, E1),
                                        not(quebra(E, U1, E1), Tr).

e_validoem(U, T) :- pe_validoem(depois(E, U), Tr),
                    em_Tr(T, depois(E, U), Tr).

e_validoem(U, T) :- pe_validoem(antes(E, U)),
                    em_Tr(T, antes(E, U), Tr).

em_Tr(T, P) :- comeco(P, E1),
               fim(P, E2),
               tempo(E1, T1), tempo(E2, T2),
               T1.<.T, T .<.T2.

menorigual(E, inic(antes(E1, U1))) :- pe_validoem(antes(E, U), Tr),
                                      pe_validoem(antes(E1, U1), Tr),
                                      exclusivo(U, U1),
                                      menor(E, E1),
                                      not(quebra(E, U1, E1, Tr)).

fim(depois(E, U), final(depois(E, U))) :- pe_validoem(depois(E, U), Tr),
                                          pe_validoem(depois(E1, U1), Tr),
                                          exclusivo(U, U1),
                                          menor(E, E1),
                                          not(quebra(E, U, E1, Tr)).

menorigual(final(depois(E, U)),  E1) :- pe_validoem(depois(E, U), Tr),
                                        pe_validoem(depois(E1, U1), Tr),
                                        exclusivo(U, U1),
                                        menor(E, E1),
                                        not(quebra(E, U, E1, Tr)).

menorigual(final(depois(E, U)), inic(antes(E1, U1))) :-
                                              pe_validoem(depois(E, U), Tr),
                                              pe_validoem(antes(E1, U1), Tr),
                                              exclusivo(U, U1),
                                              not(U = U1),
                                              menor(E, E1),
                                              not(quebra(E, U, E1, Tr)).

comeco(antes(E1, U1), inic(antes(E1, U1))) :- pe_validoem(depois(E, U), Tr),
                                              pe_validoem(antes(E1, U1), Tr),
                                              exclusivo(U, U1),
                                              not(U = U1),
                                              menor(E, E1),
                                              not(quebra(E, U, E1, Tr)).

fim(depois(E, U), final(depois(E, U))) :- pe_validoem(depois(E, U), Tr),
                                          pe_validoem(antes(E1, U1), Tr),
                                          exclusivo(U, U1),
                                          not(U = U1),
                                          menor(E, E1),
```

```
                                                    not(quebra(E, U, E1, Tr)).

menor(E, E1) :- tempo(E, X),
                tempo(E1, Y),
                X.<.Y.

menorigual(E, E1) :- tempo(E, X),
                     tempo(E1, Y),
                     X.<=.Y.

great(E, E1) :- tempo(E, X),
                tempo(E1, Y),
                X.>.Y.

greatequal(E, E1) :- tempo(E, X),
                     tempo(E1, Y),
                     X.>=.Y.

%-------------------------------------------------------------------------%
%                  CONVERTE DATAS A DIAS
%-------------------------------------------------------------------------%

date(I,J,K,D) :-
  (var(I), var(D),!,
  write('Erro, anos e dias variaveis');
  (var(D), !,
   anosbis(I,A,N),
   Days is (A * 366) + (N * 365),
   qualTab(I,T),
   X is J - 1,
   d_arg(X,T,V),
   D is V + K + Days;
   X is D // 365,
   (0 = D mod 365,!,
    I1 is 1900 + X - 1,
    anosbis(I1,A,N),
    (leap(I1), !,
     P is 366 - A;
     P is 365 - A
    ),
    I is I1,
    W is P;
    I1 is 1900 + X,
    anosbis(I1,A,N),
    P is D mod 365 - A,
    (P =< 0, !,
     I is I1 - 1,
     (leap(I1), !,
      W is 366 + P;
      W is 365 + P
     );
     I is I1,
     W is P
    )
   ),
```

```
    qualTab(I,T),
    Z is W // 30,
    d_arg(Z,T,V),
    (W > V, !, J is Z + 1, U = V;
     Y is Z - 1, d_arg(Y,T,G),
     (W > Q, !, J = Z, U = Q)
    ),
    K is W - U
   )
  ).

d_arg(0,_,0) :- !.
d_arg(I,T,V) :- arg(I,T,V).

qualTab(I,T) :-
   (leap(I), !,
    T = t(31,60,91,121,152,182,213,244,274,305,335,366);
    T = t(31,59,90,120,151,181,212,243,273,304,334,365)).

leap(I) :-
   0 is I mod 4,
   (0 is I mod 100, !, 0 is I mod 400;
    true).

anosbis(X,Y,Z) :-
   Tab = bis(0,1,2,2,3,4,4,5,6,6,7,8,8,9,10,10,11,12,12,13,14,14,15,16),
   A is X - 1900,
   Y2 is A // 4,
   (A > 100, !,
    W is A // 100,
    d_arg(W,Tab,D),
    Y1 is Y2 - D;
    Y1 is Y2
   ),
   (leap(X), !,
    Y is Y1 - 1;
    Y is y1
   ),
   Z is A - Y.
```