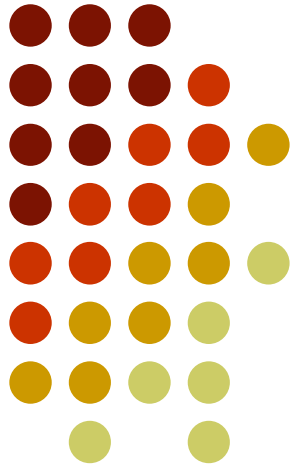
Sistemas de Produção
Reativos e Algoritmo de Rete

Profa. Josiane

Patrick Henry Winston
"*Artificial Intelligence*" – 3ª edição – cap. 7

agosto/2008



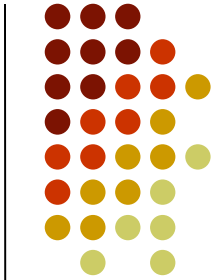
Sistemas de Produção Reativos

- As regras são da forma: **Se** condição **então** ação
  - A ação pode inserir uma nova asserção na MT, ou apagar uma asserção existente, ou executar algum outro procedimento.
- Três fases:
  - Casamento
  - **Resolução de conflitos** e
  - Execução

BAGGER: Um Sistema Reativo

- Objetivo: empacotar gêneros alimentícios em uma mercearia
- BAGGER: decide onde cada item deve ir (em qual sacola)
- BAGGER foi projetado para fazer 4 passos:
  - **Checar**: analisa o que o cliente selecionou, verifica se algum item está faltando e sugere o item para o cliente.
  - **Empacotar itens grandes**: tomando cuidado para colocar as garrafas grandes primeiro.
  - **Empacotar itens médios**: tomando cuidado para colocar os item congelados em sacolas térmicas.
  - **Empacotar itens pequenos**

BAGGER: Um Sistema Reativo
Conteúdo da MT



- Suponha que a MT tenha o seguinte conhecimento sobre os itens que devem ser empacotados:

item	**tipo de embalagem**	**tamanho**	**congelado?**
Pão	saco plástico	médio	não
macarrão inst.	pote	pequeno	não
Granola	caixa	grande	não
Sorvete	pote	médio	sim
Batata Chips	saco plástico	médio	não
Pepsi	garrafa	grande	não

BAGGER: Um Sistema Reativo
Regras

- A memória de trabalho contém a asserção que identifica o passo
- Cada regra de BAGGER identifica o passo
- B1 é disparada somente quando o passo é **checar**

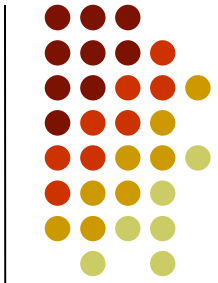
B1 **Se** Passo = checar
empacotar(batata-chips)
não existe empacotar(pepsi)
então pergunte ao cliente se ele gostaria de uma garrafa de Pepsi, em caso de aceite, *inserir*(empacotar(pepsi))

- B2 muda o passo para **empacotar-itens-grandes**

B2 **Se** Passo = checar
então *apagar* (Passo = checar)
inserir (Passo = empacotar-itens-grandes)

- B2 pode evitar que B1 faça o seu trabalho?
  - Não se a estratégia de resolução de conflitos ordenar as regras

BAGGER: Um Sistema Reativo
Regras

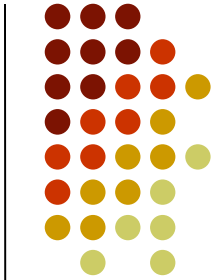


- B3 e B4 são regras para empacotar-itens-grandes

B3 **Se** Passo = empacotar-itens-grandes
empacotar(X)
tamanho(X) = grande
embalagem(X) = garrafa
sacola-corrente(S)
quantidade-itens(S) < 6
então *apagar*(empacotar(X))
inserir(em(X, S))

B4 **Se** Passo = empacotar-itens-grandes
empacotar(X)
tamanho(X) = grande
sacola-corrente(S)
quantidade-itens(S) < 6
então *apagar*(empacotar(X))
inserir(em(X, S))

BAGGER: Um Sistema Reativo
Regras



- B5 muda de sacola

B5 **Se** Passo = empacotar-itens-grandes
empacotar(X)
tamanho(X) = grande
pegar(Sacola-vazia)
então *apagar*(sacola-corrente(S))
inserir(sacola-corrente(Sacola-vazia))

- B6 muda o passo para **empacotar-itens-médios**

B6 **Se** Passo = empacotar-itens-grandes
então *apagar*(Passo = empacotar itens grandes)
inserir(Passo = empacotar-itens-médios)

BAGGER: Um Sistema Reativo
Regras

- B7 e B8 são regras para empacotar-itens-médios

B7 **Se** Passo = empacotar-itens-médios
empacotar(X)
tamanho(X) = médio
congelado(X)
pegar(Sacola-térmica)
então *apagar*(empacotar(X))
inserir(em(X, Sacola-térmica))

B8 **Se** Passo = empacotar-itens-médios
empacotar(X)
tamanho(X) = médio
sacola-corrente(S)
(vazia(S) **ou**
para todo X pertencente a relação em(X, S),
tamanho(X) = médio)
quantidade-itens(S) < 12
então *apagar*(empacotar(X))
inserir(em(X, S))

BAGGER: Um Sistema Reativo
Regras

- B9 muda de sacola

B9 **Se** Passo = empacotar-itens-médios
empacotar(X)
tamanho(X) = médio
pegar(Sacola-vazia)
então *apagar*(sacola-corrente(S))
inserir(sacola-corrente(Sacola-vazia))

- B10 muda o passo para **empacotar-itens-pequenos**

B10 **Se** Passo = empacotar-itens-médios
então *apagar*(Passo = empacotar-itens-médios)
inserir(Passo = empacotar-itens-pequenos)

BAGGER: Um Sistema Reativo
Regras

- B11 empacota itens pequenos

B11 **Se** Passo = empacotar-itens-pequenos
empacotar(X)
tamanho(X) = pequeno
sacola-corrente(S)
(vazia(S) ou
para todo X pertencente a relação em(X, S),
tamanho(X) = pequeno)
quantidade-itens(Sacola-corrente) < 18
então *apagar*(empacotar(X))
inserir(em(X, S))

- B12 muda de sacola

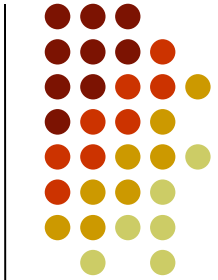
B12 **Se** Passo = empacotar-itens-pequenos
empacotar(X)
tamanho(X) = pequeno
pegar(Sacola-vazia)
então *apagar*(sacola-corrente(S))
inserir(sacola-corrente(Sacola-vazia))

BAGGER: Um Sistema Reativo
Regras

- B13 termina o empacotamento

B13	**Se**	Passo = empacotar-itens-pequenos
	então	*apagar*(Passo = empacotar-itens-pequenos)
		inserir(Passo = terminado)

BAGGER: Um Sistema Reativo
Conteúdo da MT

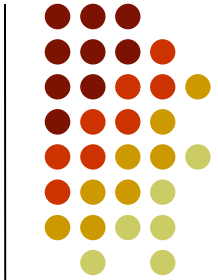


- Suponha que a MT tenha o seguinte conhecimento sobre os o passo, a sacola corrente e sobre os itens que devem ser empacotados:
  - Passo = checar
  - sacola-corrente(s1)
  - empacotar(pão)
  - empacotar(macarrão-instantâneo)
  - empacotar(granola)
  - empacotar(sorvete)
  - empacotar(batata-chips)

BAGGER: Um Sistema Reativo
Funcionamento

- 1º Ciclo:
  - Regras habilitadas: B1 e B2
  - Regras disparadas: B1 (pela ordem) – sugerir ao cliente
  - Nova MT: (supondo que o cliente aceitou a sugestão da Pepsi)
    - Passo = checar
    - sacola-corrente(s1)
    - empacotar(pão)
    - empacotar(macarrão-instantâneo)
    - empacotar(granola)
    - empacotar(sorvete)
    - empacotar(batata-chips)
    - empacotar(pepsi)

BAGGER: Um Sistema Reativo
Funcionamento



- 2º Ciclo:
  - Regras habilitadas: B2
  - Regras disparadas: B2 – mudar o passo
  - Nova MT:
    - sacola-corrente(s1)
    - empacotar(pão)
    - empacotar(macarrão-instantâneo)
    - empacotar(granola)
    - empacotar(sorvete)
    - empacotar(batata-chips)
    - empacotar(pepsi)
    - Passo = empacotar-itens-grandes

BAGGER: Um Sistema Reativo
Funcionamento

- 3º Ciclo:
  - Regras habilitadas: B3, B4, B5 e B6
  - Regras disparadas: B3 (pela ordem) – empacotar garrafas primeiro
    - Unificação = {X/pepsi}
  - Nova MT:
    - sacola-corrente(s1)
    - empacotar(pão)
    - empacotar(macarrão-instantâneo)
    - empacotar(granola)
    - empacotar(sorvete)
    - empacotar(batata-chips)
    - Passo = empacotar-itens-grandes
    - em(pepsi, s1)

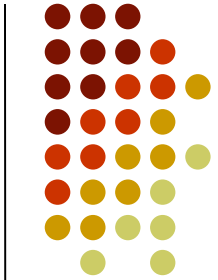
BAGGER: Um Sistema Reativo
Funcionamento

- 4º Ciclo:
  - Regras habilitadas: B4, B5 e B6
  - Regras disparadas: B4 (pela ordem) – empacotar outros itens grandes
    - Unificação = {X/granola}
  - Nova MT:
    - sacola-corrente(s1)
    - empacotar(pão)
    - empacotar(macarrão-instantâneo)
    - empacotar(sorvete)
    - empacotar(batata-chips)
    - Passo = empacotar-itens-grandes
    - em(pepsi, s1)
    - em(granola, s1)

BAGGER: Um Sistema Reativo
Funcionamento

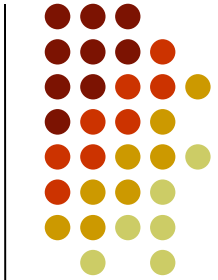
- 5º Ciclo:
  - Regras habilitadas: B6
  - Regras disparadas: B6 – mudar o passo
  - Nova MT:
    - sacola-corrente(s1)
    - empacotar(pão)
    - empacotar(macarrão-instantâneo)
    - empacotar(sorvete)
    - empacotar(batata-chips)
    - em(pepsi, s1)
    - em(granola, s1)
    - Passo = empacotar-itens-médios

BAGGER: Um Sistema Reativo
Funcionamento



- 6º Ciclo:
  - Regras habilitadas: B7, B8, B9, B10
  - Regras disparadas: B7 (pela ordem) – empacotar itens congelados primeiro
    - Unificação {X/sorvete, Sacola-térmica/st1}
  - Nova MT:
    - sacola-corrente(s1)
    - empacotar(pão)
    - empacotar(macarrão-instantâneo)
    - empacotar(batata-chips)
    - em(pepsi, s1)
    - em(granola, s1)
    - Passo = empacotar-itens-médios
    - em(sorvete, st1)

BAGGER: Um Sistema Reativo
Funcionamento



- 7º Ciclo:
  - Regras habilitadas: B9, B10
  - Regras disparadas: B9 (pela ordem) – mudar de sacola
    - Unificação {X/pão, S/s1, Sacola-vazia/s2}
  - Nova MT:
    - empacotar(pão)
    - empacotar(macarrão-instantâneo)
    - empacotar(batata-chips)
    - em(pepsi, s1)
    - em(granola, s1)
    - Passo = empacotar-itens-médios
    - em(sorvete, st1)
    - sacola-corrente(s2)

BAGGER: Um Sistema Reativo
Funcionamento

- 8º Ciclo:
  - Regras habilitadas: B8, B9, B10
  - Regras disparadas: B8 (pela ordem) – empacotar outros itens médios
    - Unificação {X/pão, S/s2}
  - Nova MT:
    - empacotar(macarrão-instantâneo)
    - empacotar(batata-chips)
    - em(pepsi, s1)
    - em(granola, s1)
    - Passo = empacotar-itens-médios
    - em(sorvete, st1)
    - sacola-corrente(s2)
    - em(pão, s2)

BAGGER: Um Sistema Reativo
Funcionamento

- 9º Ciclo:
  - Regras habilitadas: B8, B9, B10
  - Regras disparadas: B8 (pela ordem) – empacotar outros itens médios
    - Unificação {X/batata-chips, S/s2}
  - Nova MT:
    - empacotar(macarrão-instantâneo)
    - em(pepsi, s1)
    - em(granola, s1)
    - Passo = empacotar-itens-médios
    - em(sorvete, st1)
    - sacola-corrente(s2)
    - em(pão, s2)
    - em(batata-chips, s2)

BAGGER: Um Sistema Reativo
Funcionamento

- 10º Ciclo:
  - Regras habilitadas: B10
  - Regras disparadas: B10 – mudar o passo
  - Nova MT:
    - empacotar(macarrão-instantâneo)
    - em(pepsi, s1)
    - em(granola, s1)
    - em(sorvete, st1)
    - sacola-corrente(s2)
    - em(pão, s2)
    - em(batata-chips, s2)
    - Passo = empacotar-itens-pequenos

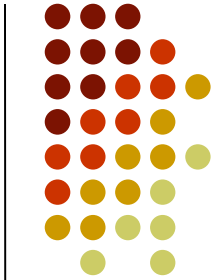
BAGGER: Um Sistema Reativo
Funcionamento

- 11º Ciclo:
  - Regras habilitadas: B12, B13
  - Regras disparadas: B12 (pela ordem) – mudar a sacola
    - Unificação = {X/macarrão-instantâneo, Sacola-vazia/s3, S/s2}
  - Nova MT:
    - empacotar(macarrão-instantâneo)
    - em(pepsi, s1)
    - em(granola, s1)
    - em(sorvete, st1)
    - em(pão, s2)
    - em(batata-chips, s2)
    - Passo = empacotar-itens-pequenos
    - sacola-corrente(s3)

BAGGER: Um Sistema Reativo
Funcionamento

- 12º Ciclo:
  - Regras habilitadas: B11, B12, B13
  - Regras disparadas: B11 (pela ordem) – empacotar itens pequenos
    - Unificação = {X/macarrão-instantâneo, S/s3}
  - Nova MT:
    - em(pepsi, s1)
    - em(granola, s1)
    - em(sorvete, st1)
    - em(pão, s2)
    - em(batata-chips, s2)
    - Passo = empacotar-itens-pequenos
    - sacola-corrente(s3)
    - em(macarrão-instantâneo, s3)

BAGGER: Um Sistema Reativo
Funcionamento

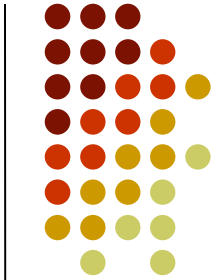


- 13º Ciclo:
  - Regras habilitadas: B13
  - Regras disparadas: B13 - terminar
  - Nova MT:
    - em(pepsi, s1)
    - em(granola, s1)
    - em(sorvete, st1)
    - em(pão, s2)
    - em(batata-chips, s2)
    - sacola-corrente(s3)
    - em(macarrão-instantâneo, s3)
    - Passo = terminado

BAGGER: Um Sistema Reativo
Funcionamento

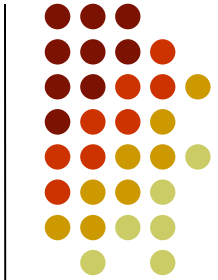
- 14º Ciclo:
  - Regras habilitadas: nenhuma
  - Regras disparadas: nenhuma
  - MT Final:
    - em(pepsi, s1)
    - em(granola, s1)
    - em(sorvete, st1)
    - em(pão, s2)
    - em(batata-chips, s2)
    - sacola-corrente(s3)
    - em(macarrão-instantâneo, s3)
    - Passo = terminado

Fase de Casamento (*match*)



- Casamento
  - O sistema, em cada ciclo, computa o subconjunto de regras no qual os antecedentes são satisfeitos pelo conteúdo atual da memória de trabalho

Exemplo da fase de casamento
Cavalos de corrida

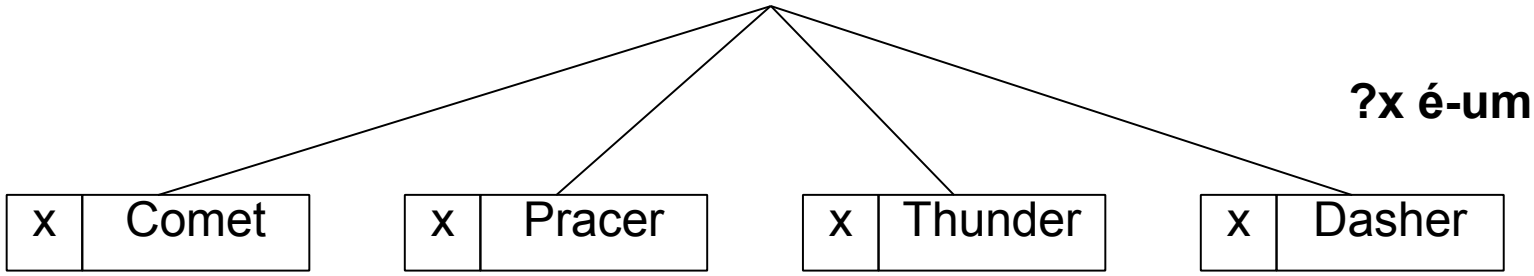


- Conteúdo da MT:
  - Comet é-um cavalo
  - Pracer é-um cavalo
  - Comet é-pai-de Dasher
  - Comet é-pai-de Pracer
  - Pracer é rápido
  - Dasher é-pai-de Thunder
  - Thunder é rápido
  - Thunder é-um cavalo
  - Dasher é-um cavalo
- Regra para estabelecer se um cavalo é valioso:

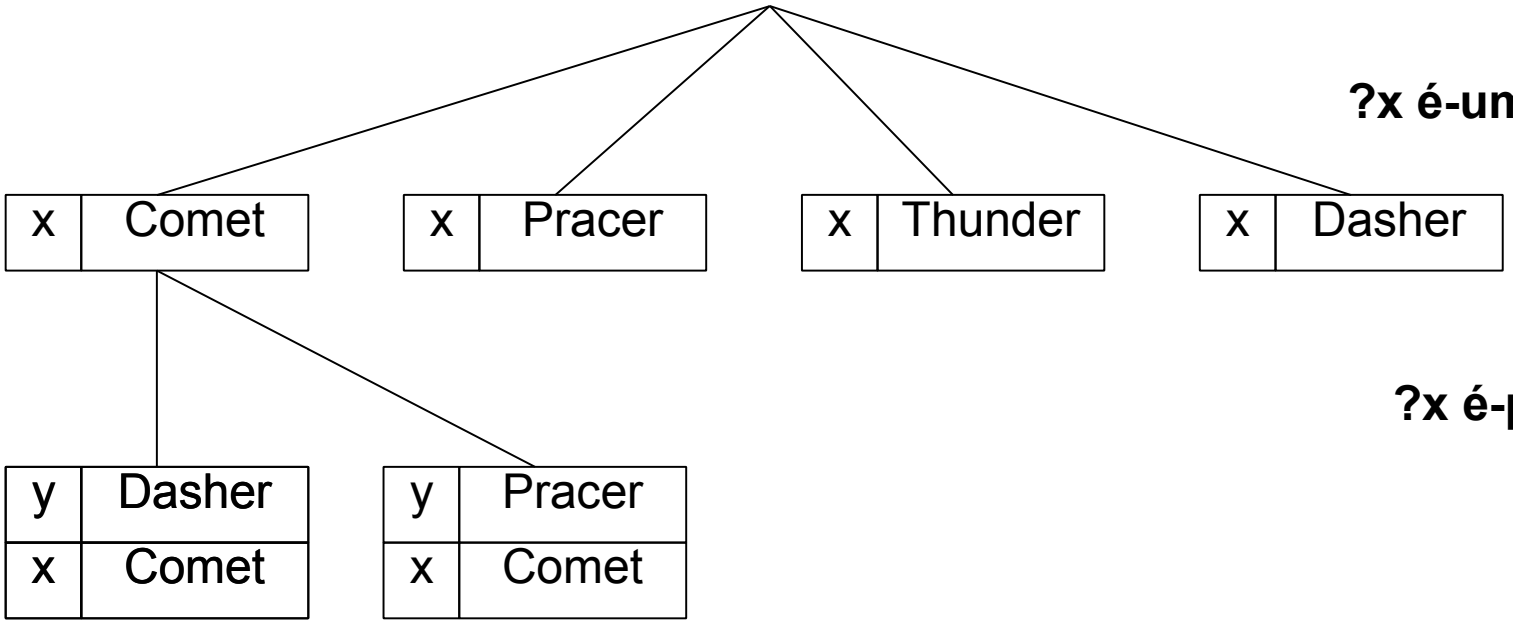
se ?x é-um cavalo
?x é-pai-de ?y
?y é rápido

então ?x é valioso

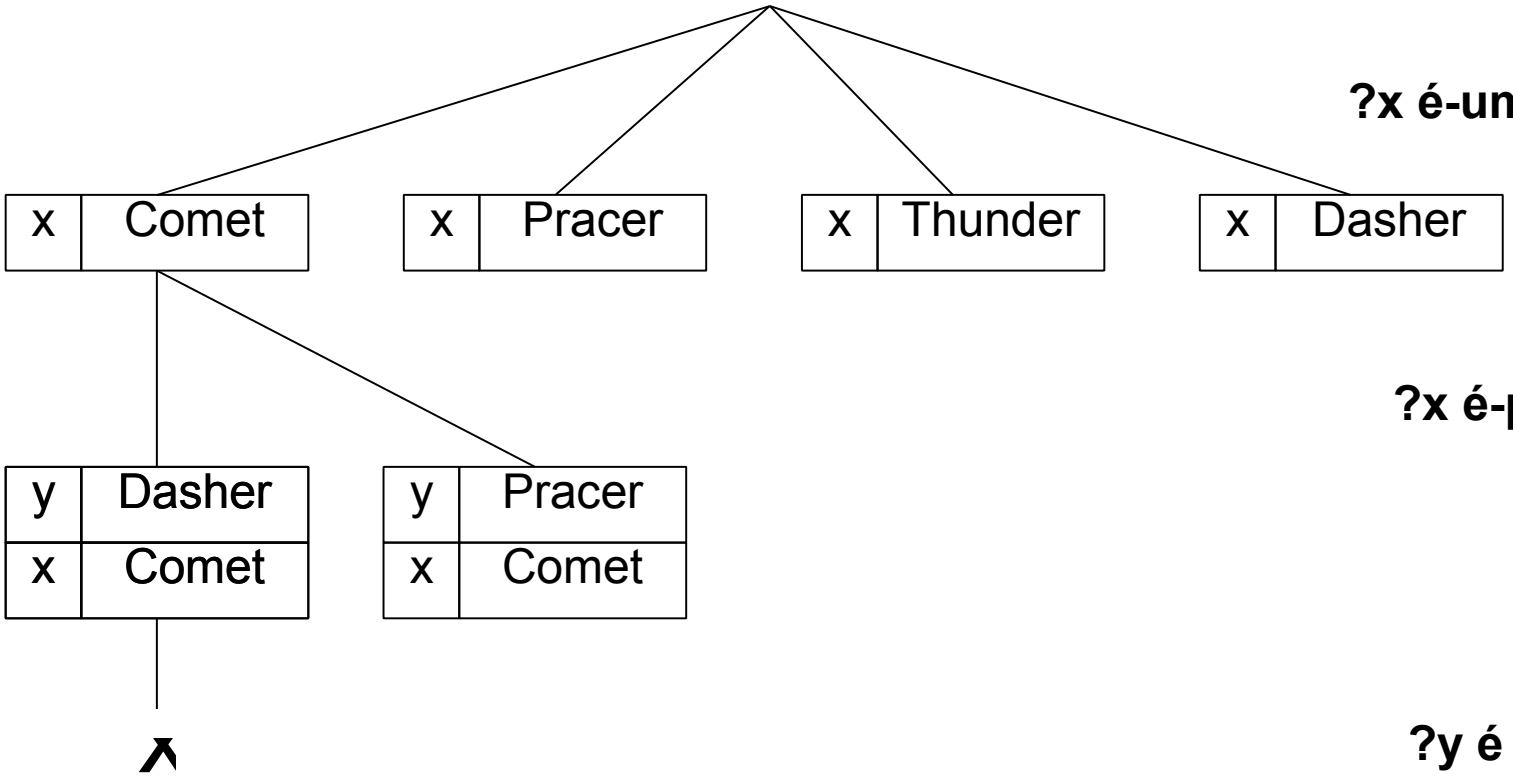
Exemplo da fase de casamento com encadeamento para frente



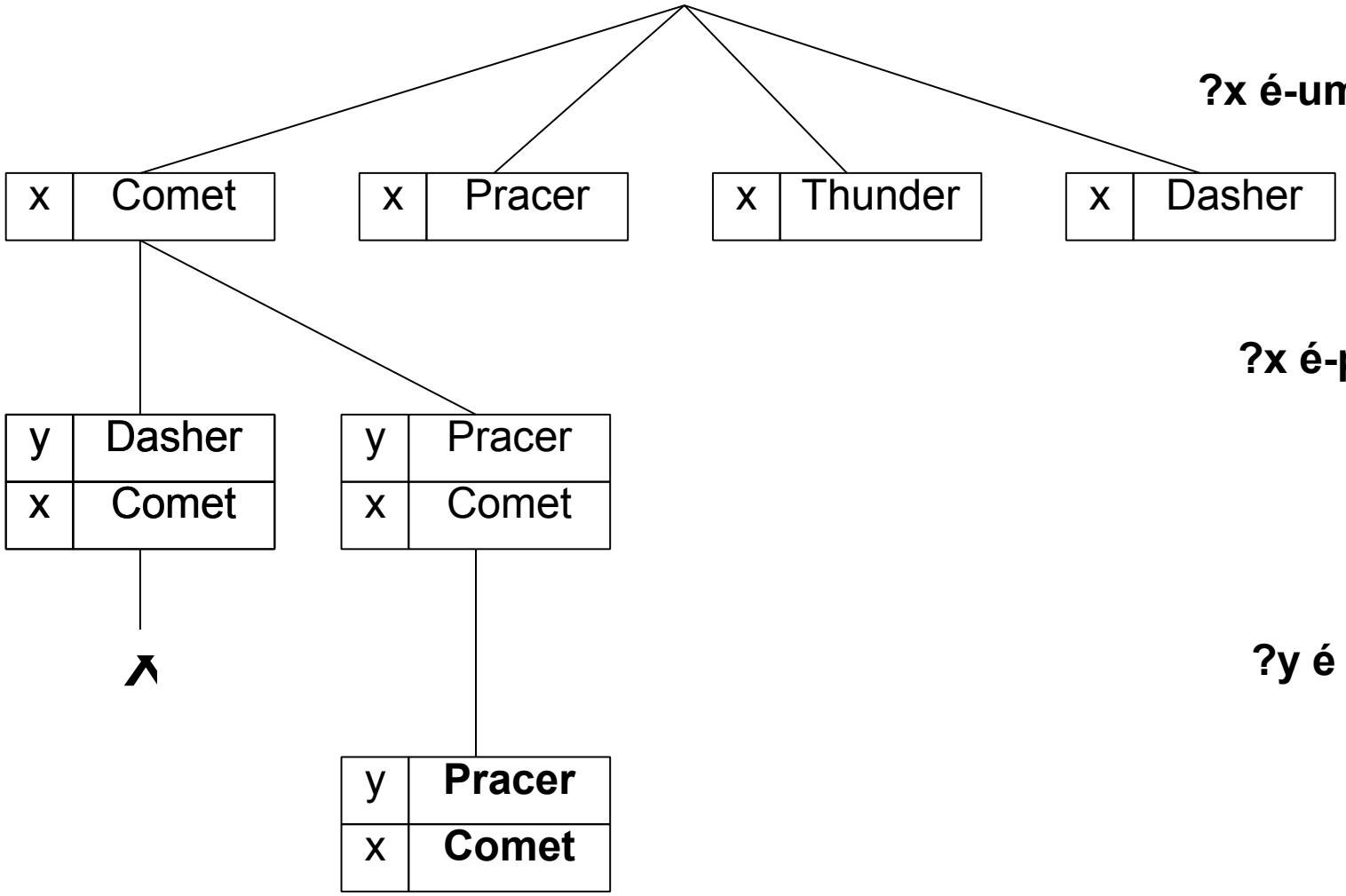
Exemplo da fase de casamento com encadeamento para frente



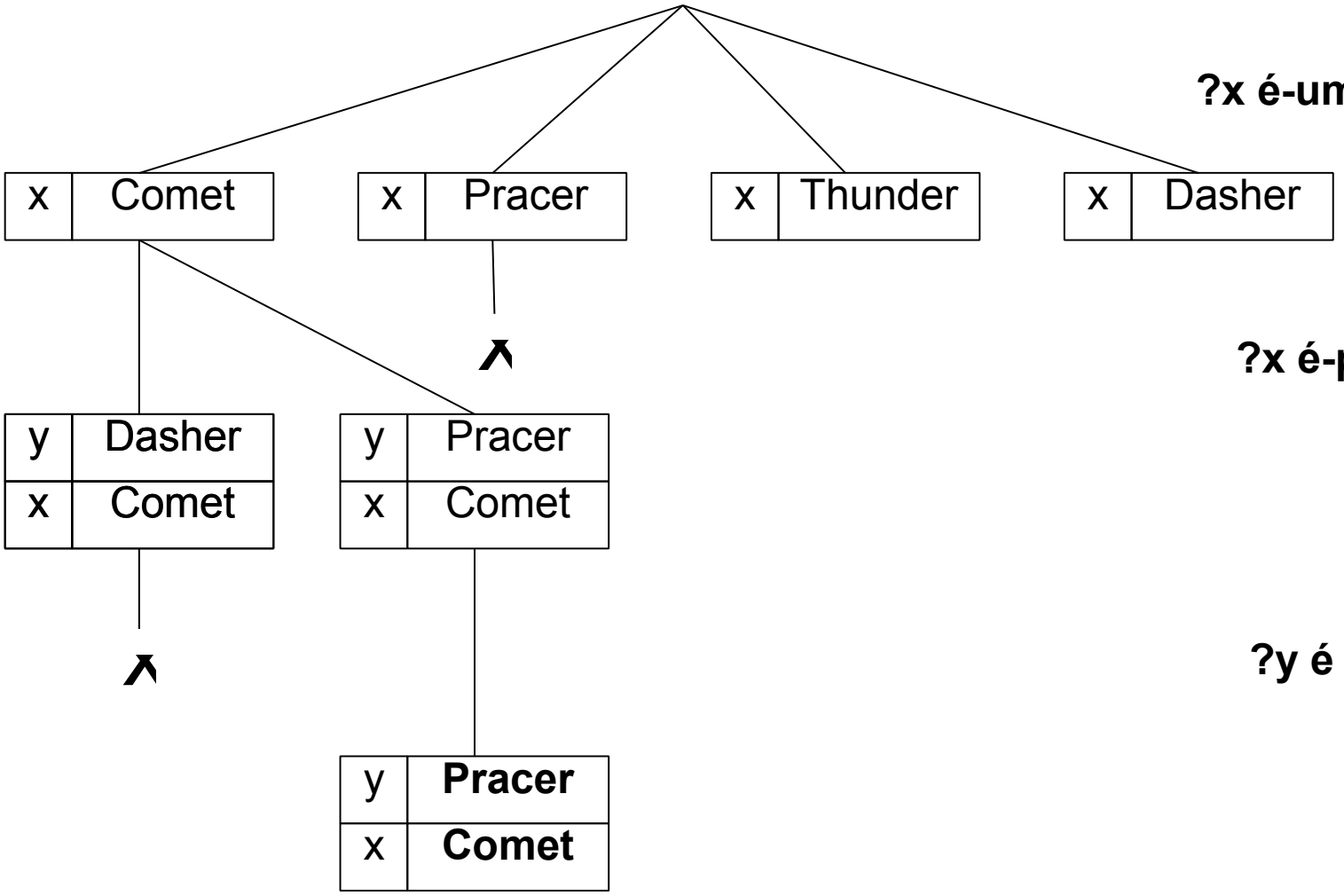
Exemplo da fase de casamento com encadeamento para frente



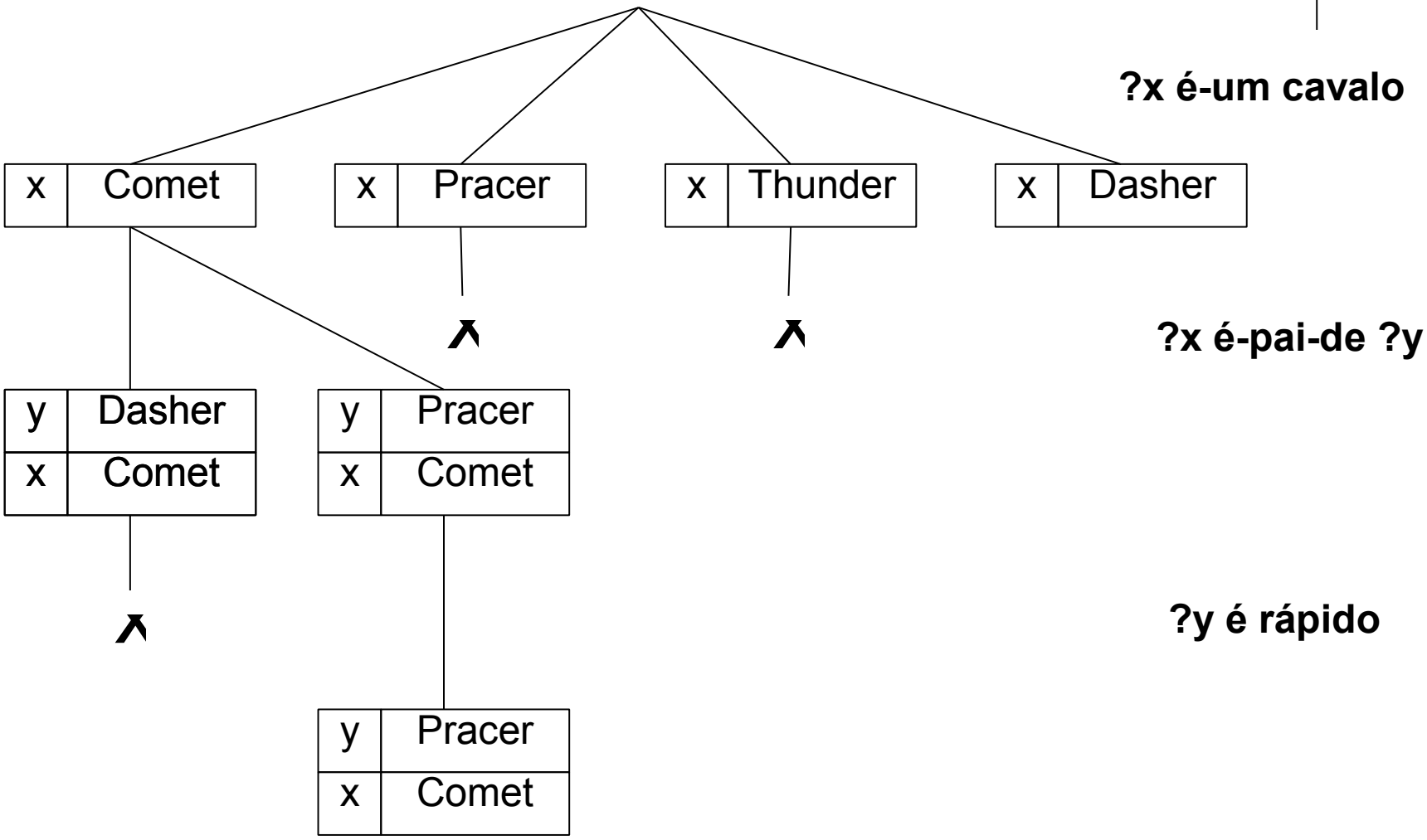
Exemplo da fase de casamento com encadeamento para frente



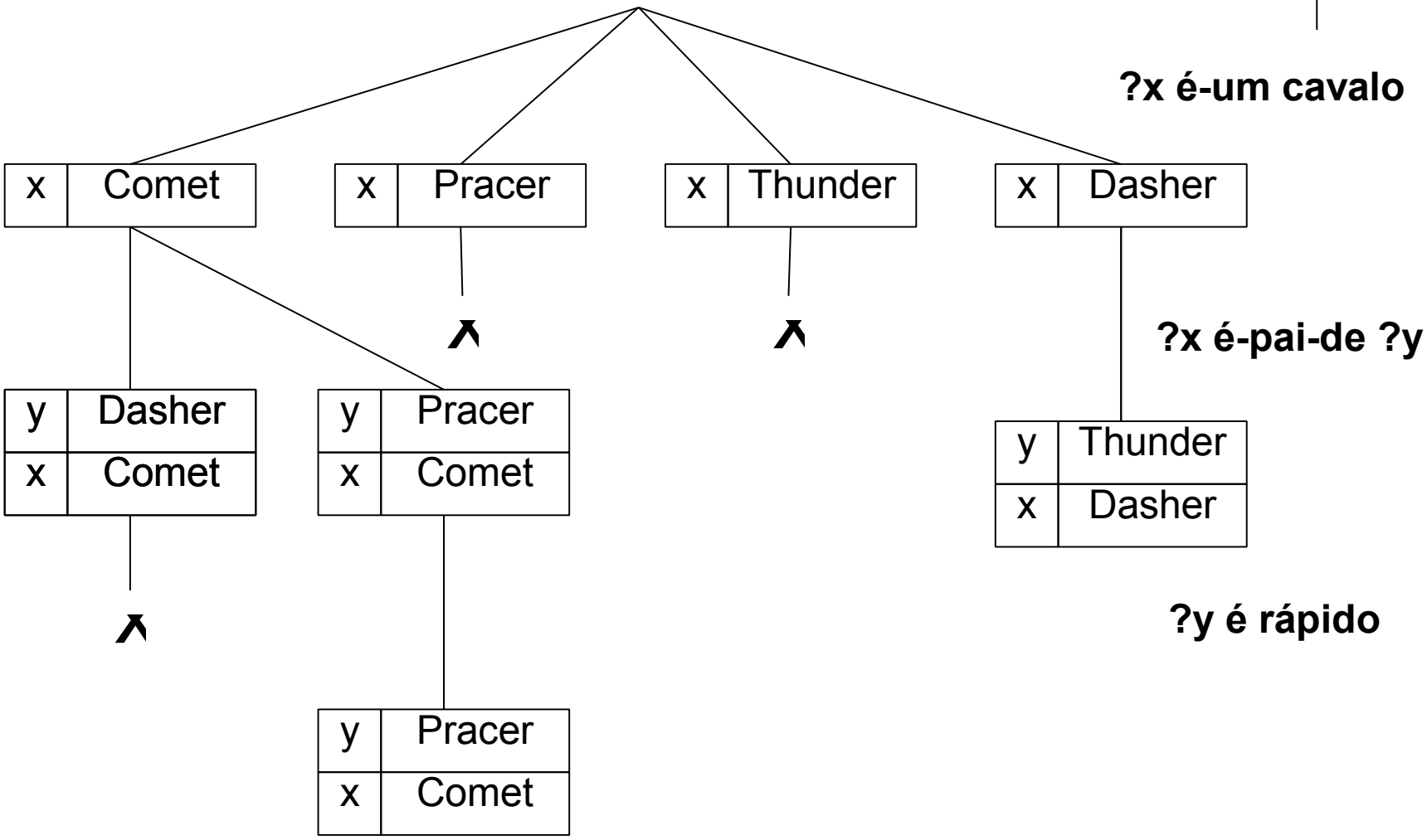
Exemplo da fase de casamento com encadeamento para frente



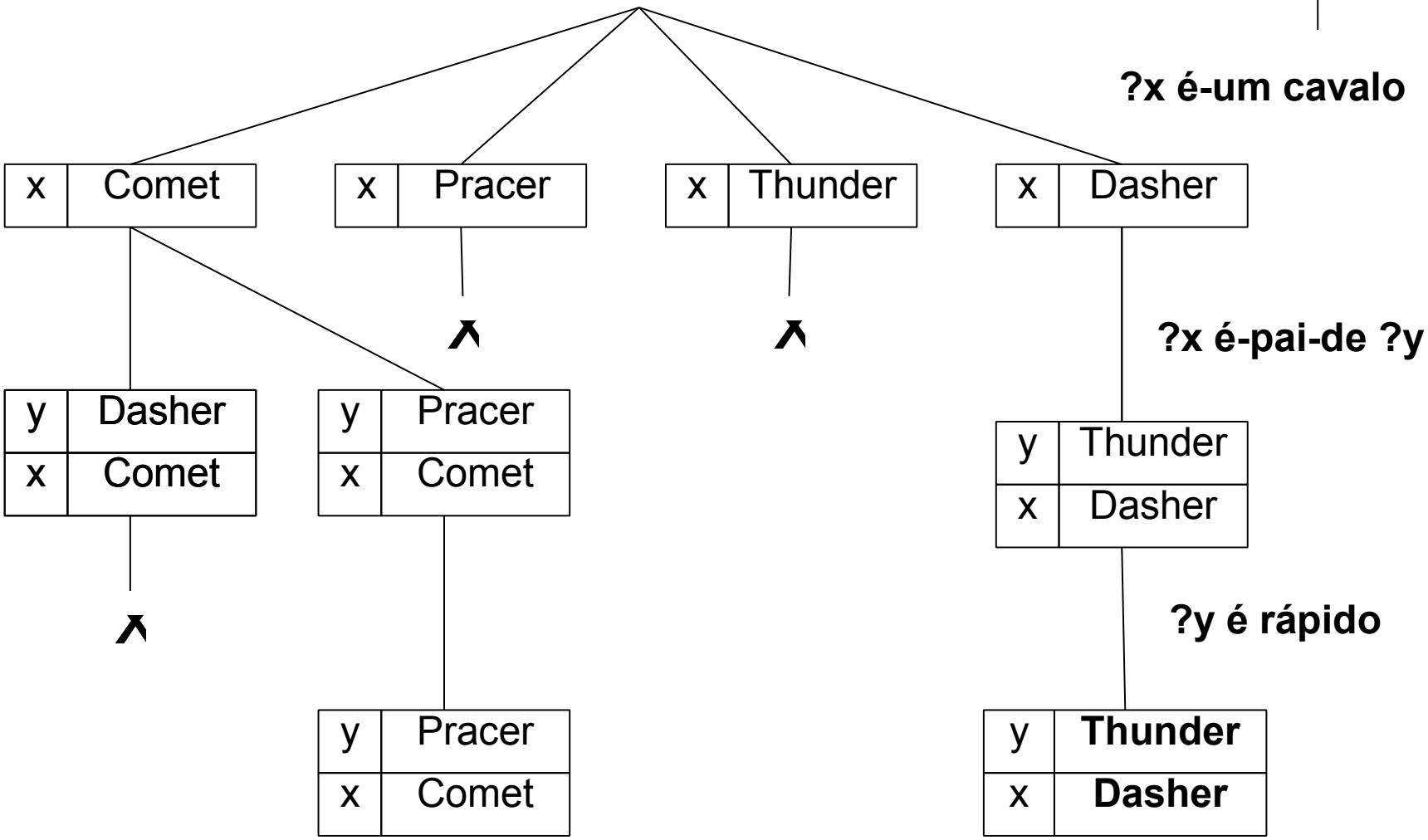
Exemplo da fase de casamento com encadeamento para frente



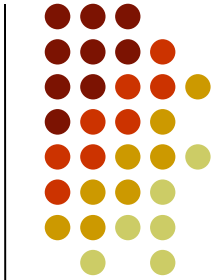
Exemplo da fase de casamento com encadeamento para frente



Exemplo da fase de casamento com encadeamento para frente



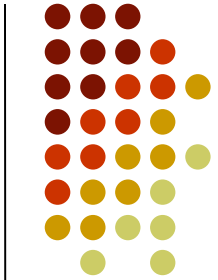
Exemplo da fase de casamento como operações relacionais



- Vamos pensar na MT como uma relação:

Primeira	Segunda	Terceira
Comet	É-um	Cavalo
Pracer	É-um	Cavalo
Comet	É-pai-de	Dasher
Comet	É-pai-de	Pracer
Pracer	É	Rápido
Dasher	É-pai-de	Thunder
Thunder	É	Rápido
Thunder	É-um	Cavalo
Dasher	É-um	Cavalo

Exemplo da fase de casamento como operações relacionais



- Para determinar os valores de ?x e?y para a regra dos cavalos valiosos
  - Vamos determinar quais os registros da relação que casam com o primeiro antecedente da regra (*?x é-um cavalo*)
  - Isso implica em selecionar os registros nos quais a segunda coluna é igual a *é-um* e a terceira coluna é igual a *cavalo* ou
  - SELECT * FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
  - Temos como resultado:

Primeira	**Segunda**	**Terceira**
Comet	É-um	Cavalo
Pracer	É-um	Cavalo
Thunder	É-um	Cavalo
Dasher	É-um	Cavalo

Exemplo da fase de casamento como operações relacionais

- Queremos saber quais valores casam com ?x, desta forma estamos interessados nos valores da primeira coluna
- Isto implica em fazer uma projeção sobre a primeira coluna da relação resultante da seleção
- SELECT primeira FROM MT

Primeira
Comet
Pracer
Thunder
Dasher

Renomeando ==>

A1	**X**
	Comet
	Pracer
	Thunder
	Dasher

- Poderíamos ter feito diretamente: SELECT primeira FROM MT WHERE segunda=”é-um” AND terceira=”cavalo”

Exemplo da fase de casamento como operações relacionais

- Vamos determinar quais os registros da relação que casam com o segundo antecedente da regra (?x é-pai-de ?y)
  - SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
  - Temos como resultado:

A2

X	Y
Comet	Dasher
Comet	Pracer
Dasher	Thunder

Exemplo da fase de casamento como operações relacionais

- Vamos determinar quais os registros da relação que casam com o terceiro antecedente da regra (?y é rápido)
  - SELECT primeira FROM MT WHERE segunda = “é” AND terceira=”rápido”
  - Temos como resultado:

A3	**Y**
	Pracer
	Thunder

Exemplo da fase de casamento como operações relacionais

- Agora temos as seguinte relações:

A1

X
Comet
Pracer
Thunder
Dasher

A2

X	Y
Comet	Dasher
Comet	Pracer
Dasher	Thunder

A3

Y
Pracer
Thunder

- E queremos saber quais os valores de ?x que satisfazem o primeiro e o segundo antecedentes
  - Isto é equivalente a um JOIN entre a relação A1 e A2, resultando em:

B1

X	Y
Comet	Dasher
Comet	Pracer
Dasher	Thunder

Exemplo da fase de casamento como operações relacionais

- Agora temos as seguinte relações:

B1

X	Y
Comet	Dasher
Comet	Pracer
Dasher	Thunder

A3

Y
Pracer
Thunder

- E queremos saber quais os valores de ?Y que satisfazem o segundo e terceiro antecedentes
  - Isto é equivalente a um JOIN entre a relação A3 e B1, resultando em:

B2

X	Y
Comet	Pracer
Dasher	Thunder

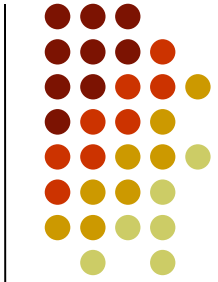
Exemplo da fase de casamento como operações relacionais

- Como o consequente da regra diz respeito somente a ?x:

B2	**X**
	Comet
	Dasher

- Isto significa que tanto Comet quanto Dasher são valiosos.
  - Em sistemas dedutivos, as duas instanciações de ?x podem ser usadas
  - Em sistemas reativos, o procedimento de resolução de conflitos selecionará uma ação

Fase de casamento como operações relacionais



- Problema: este procedimento ocupa muita computação
- Se a regra tem *n* antecedentes, precisamos de:
  - *n* selects, *n* projects para produzir as relações A +
  - *n-1* joins e *n-1* projects para produzir as relações B
- Se existir *m* regras, a cada nova asserção, teremos:
  - *mn* selects, *m(2n-1)* projects, *m(n-1)* joins – que são caros
- Solução: Algoritmo de Rete
  - Os JOINs são executados entre uma linha de uma relação e a outra relação, o que diminui o custo de JOINs grandes

Fase de casamento: Algoritmo de Rete

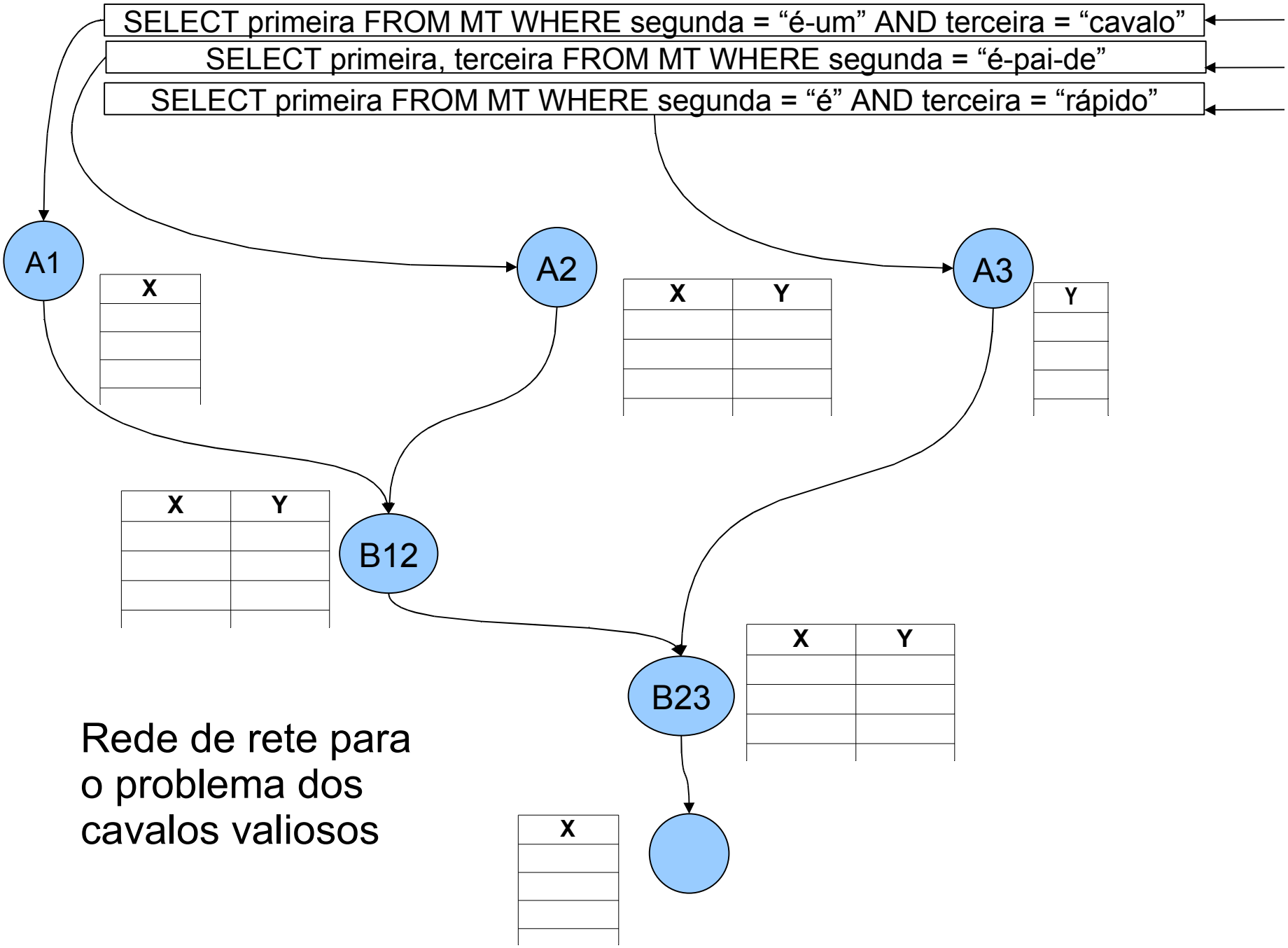
- Inventado por Dr. Charles Forgy em 1979
- Duas partes:
  - Tempo de Compilação
    - Descreve como gerar uma rede de discriminação para as regras da base que possa auxiliar a fase de casamento
    - A rede de discriminação é utilizada com um “filtro de dados”
  - Tempo de Execução
    - A rede é utilizada para unificar a memória de trabalho com as regras da base de forma mais eficiente

Fase de casamento: Algoritmo de Rete
Construção da Rede de Rete

Para cada antecedente que aparece no conjunto de regras, crie uma operação de SELECT que examina novas asserções

Para cada regra

- Para cada antecedente
  - Crie um nó alfa e anexe a ele a operação de SELECT correspondente (já criada)
- Para cada nó alfa exceto o primeiro
  - Crie um nó beta e anexe a operação de JOIN correspondente
  - Se o nó beta é o primeiro anexe a ele o primeiro e segundo nós alfa
  - Caso contrário, anexe ao nó beta o nó alfa correspondente e o nó beta anterior
- Anexe uma a operação de PROJECT correspondente ao nó beta final


SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
A1
A2
A3
B12
B23
X
X
Y
Y
X
Y
X
Y
X
Rede de rete para
o problema dos
cavalos valiosos

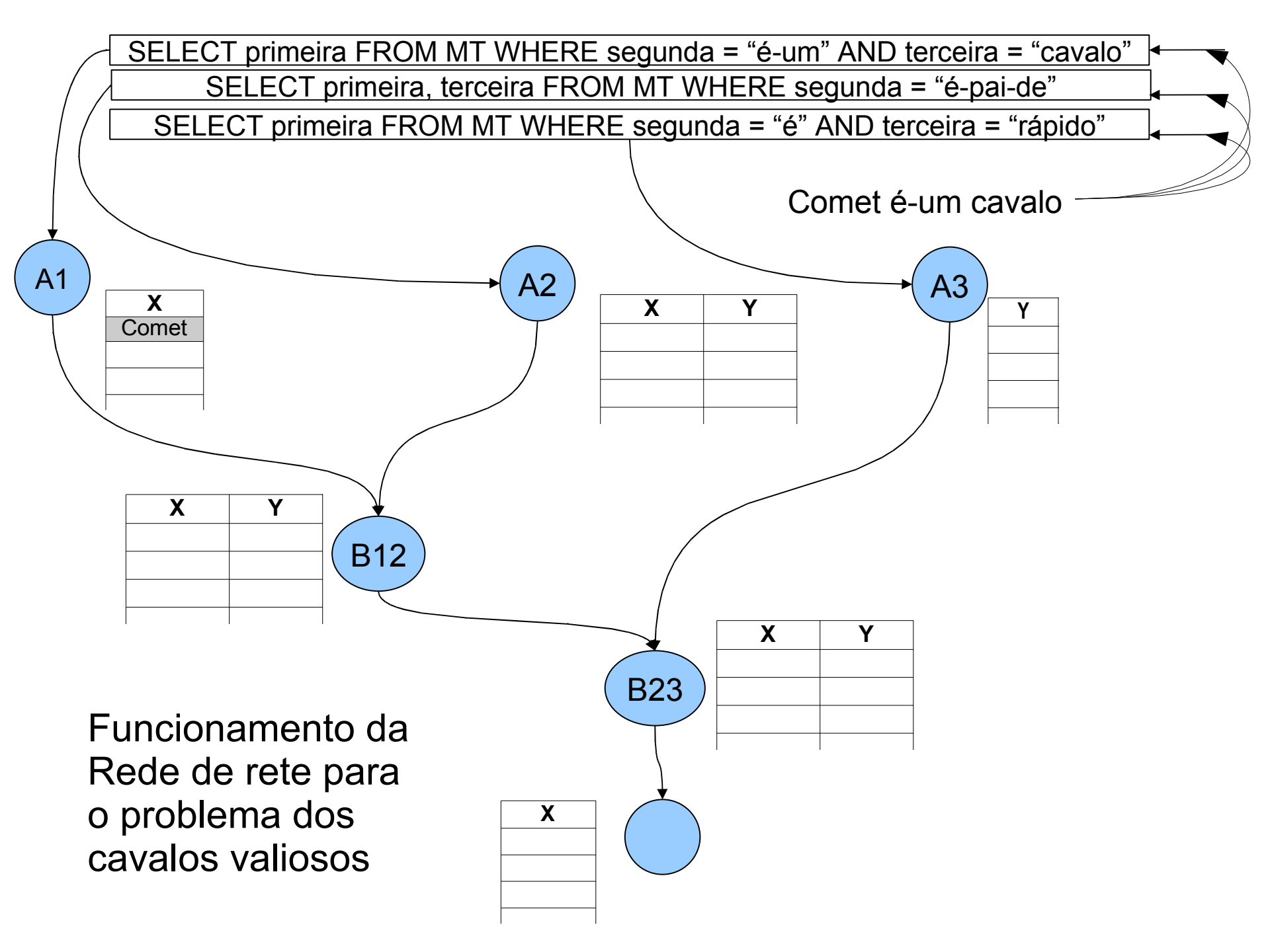
Fase de casamento: Algoritmo de Rete
Uso da Rede de Rete como filtro

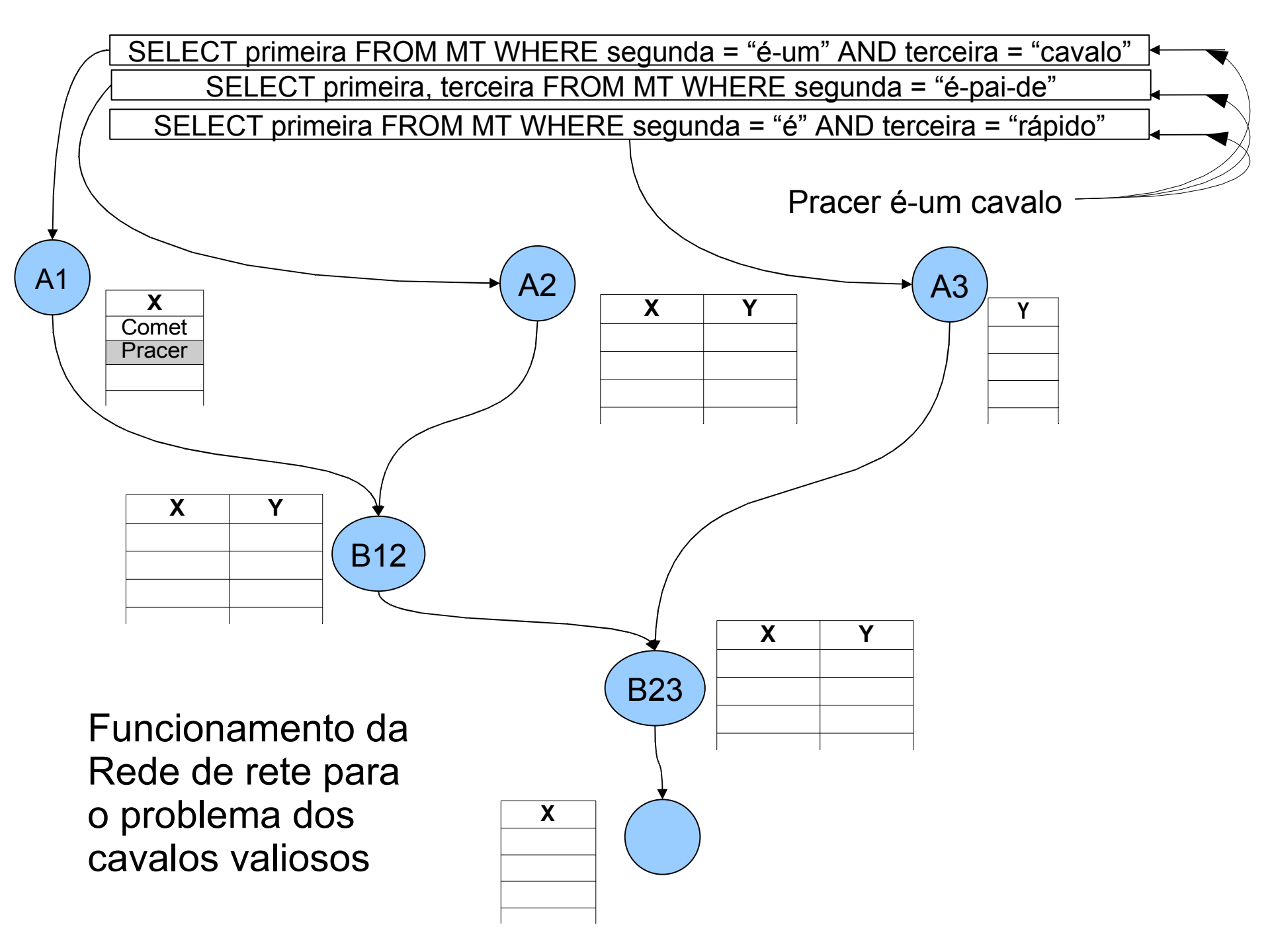
Para cada asserção, filtre a asserção pelas operações de SELECT, passando a asserção através da rede para os nós alfa associados

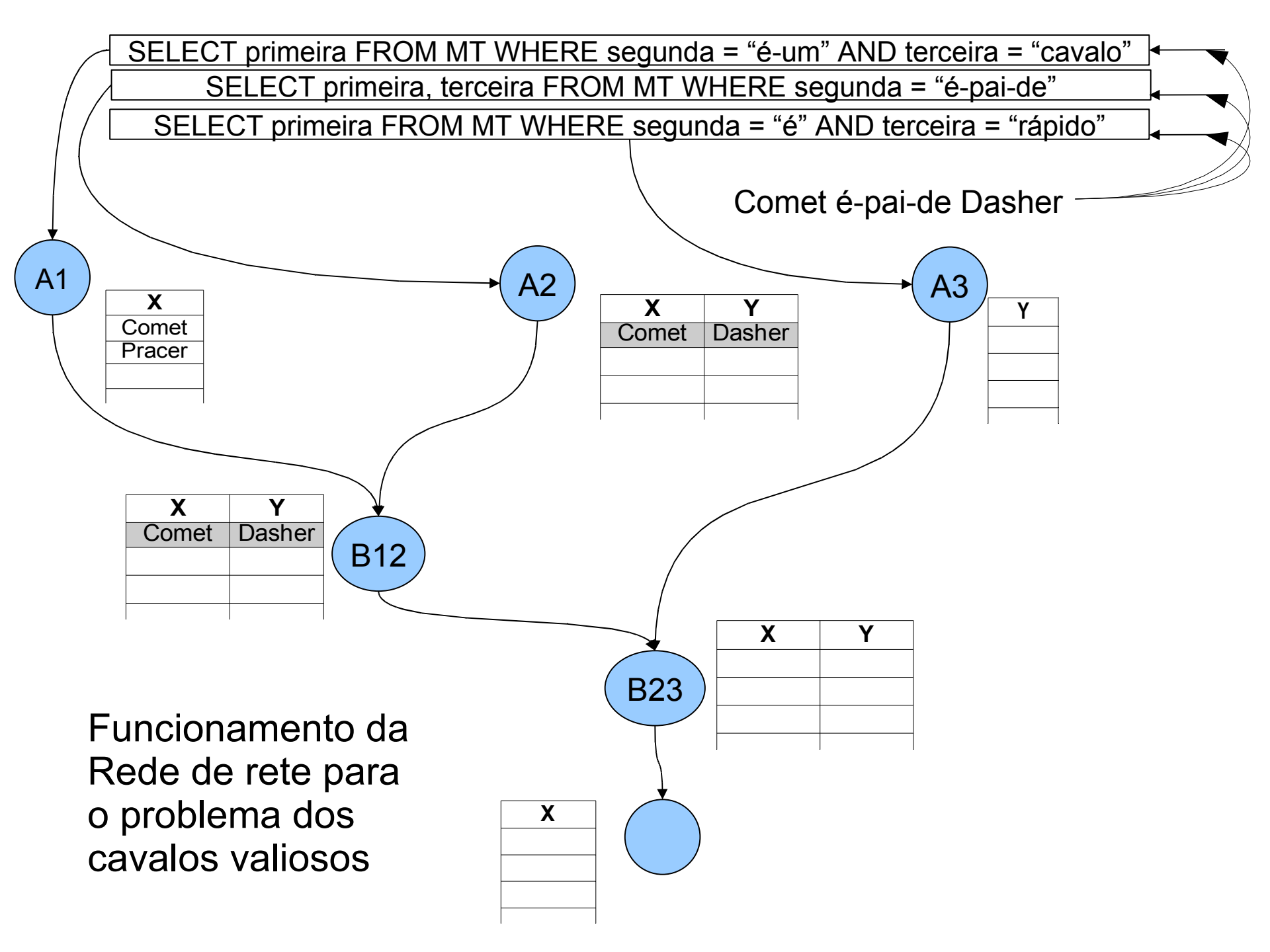
Para cada nó alfa que recebe uma asserção, use a operação de JOIN passando a asserção através da rede para o nó beta associado

Para cada nó beta que recebe uma nova asserção, use a operação de JOIN para propagar as mudanças através da rede para os outros nós beta associados a ele

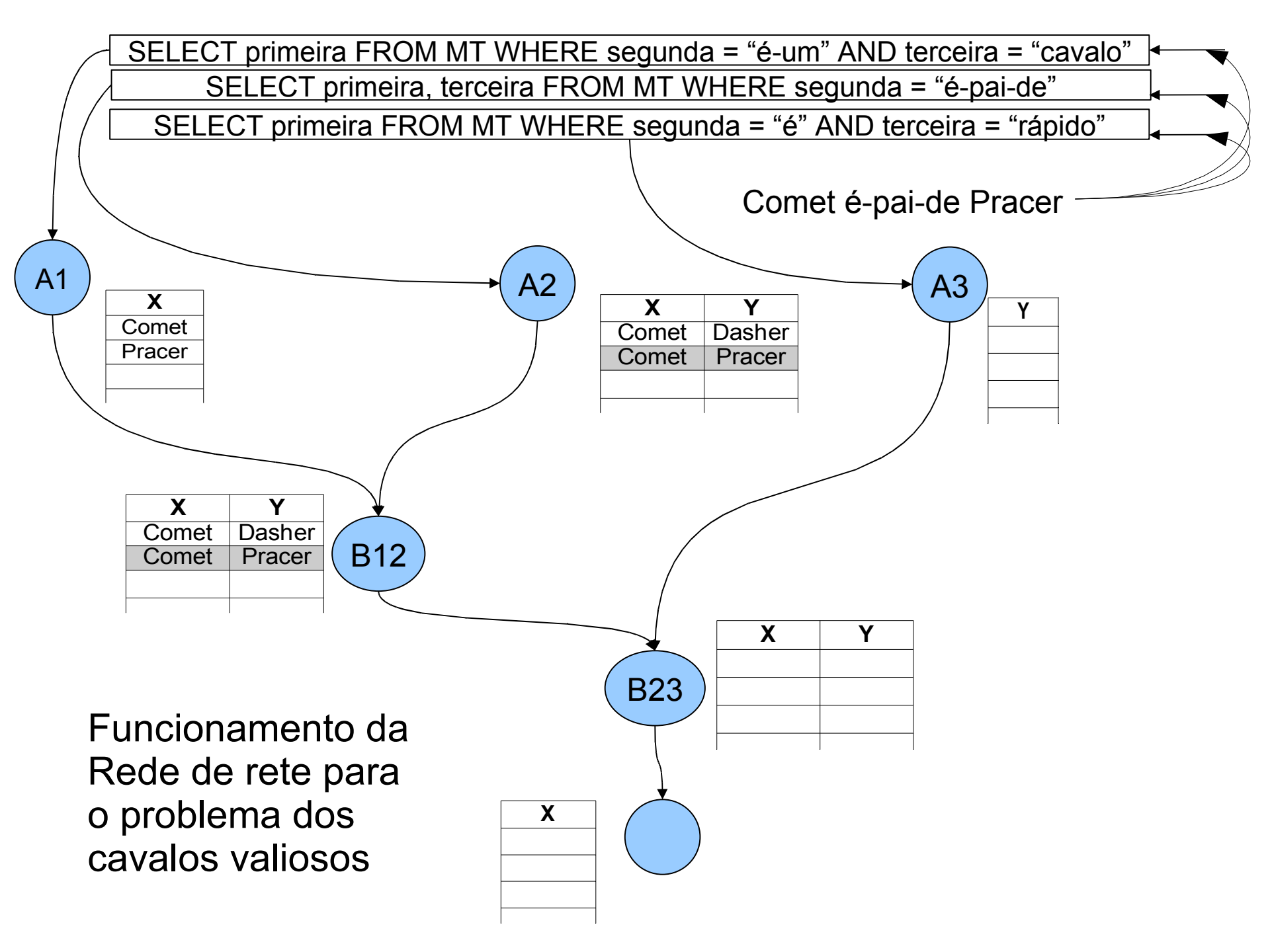
Para cada regra, use a operação de PROJECT para isolar o valor associado à variável correspondente ao consequente da regra

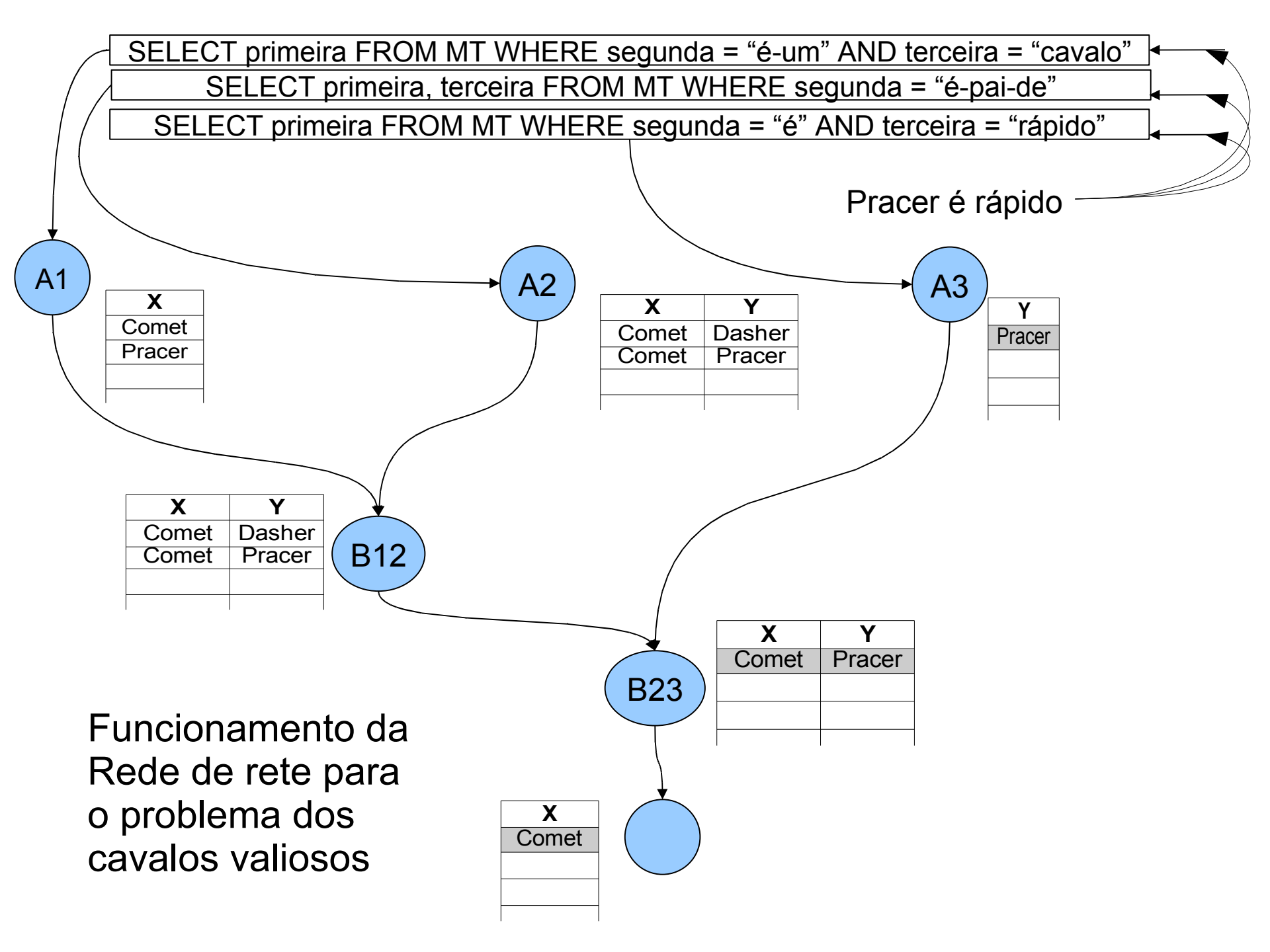

SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Comet é-um cavalo
A1
A2
A3
X
Comet
X
Y
Y
X
Y
B12
B23
X
Y
X
Funcionamento da Rede de rete para o problema dos cavalos valiosos

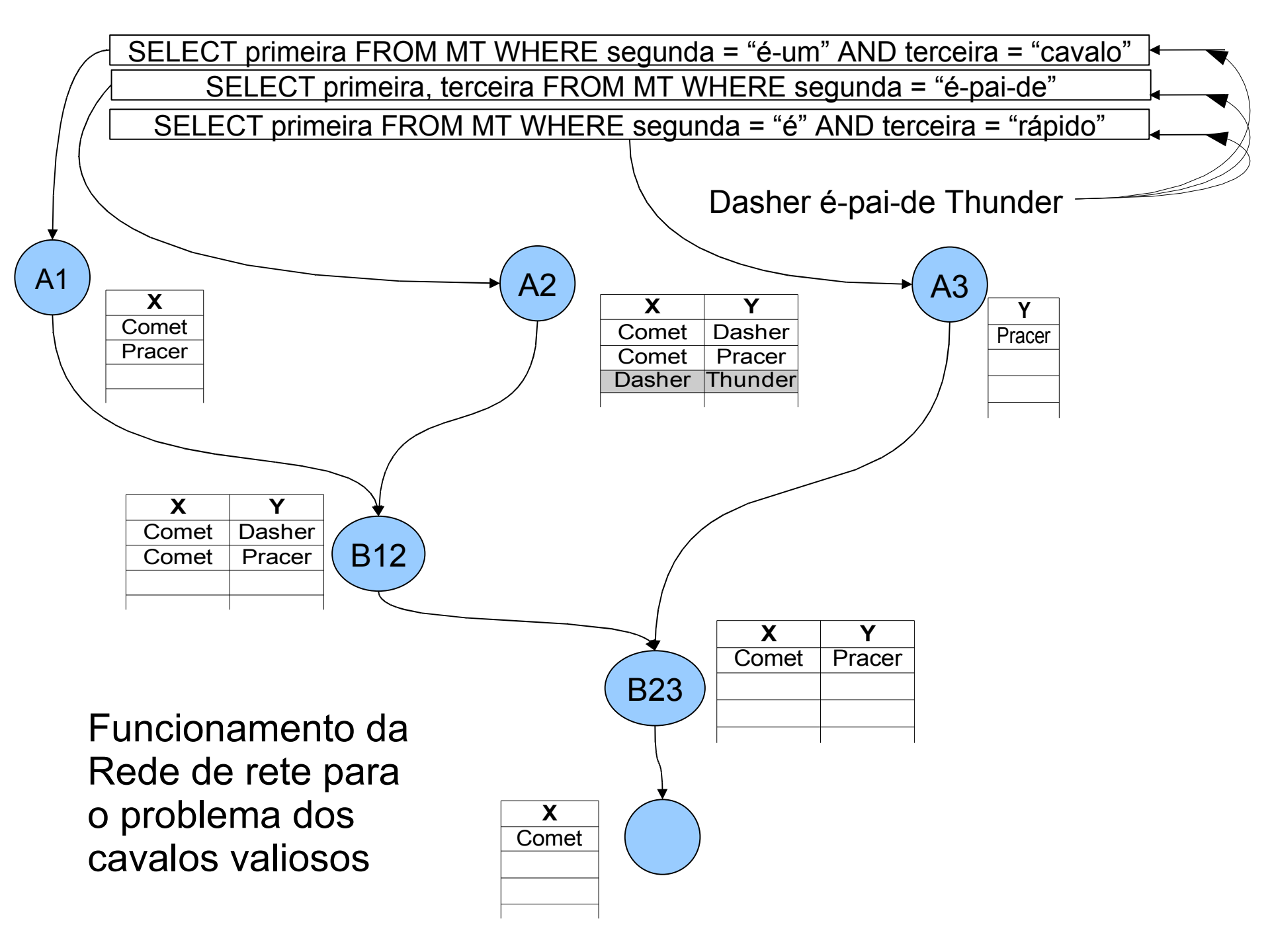

SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Pracer é-um cavalo
A1
A2
A3
X
Comet
Pracer
X
Y
Y
X
Y
B12
B23
X
Y
X
Funcionamento da Rede de rete para o problema dos cavalos valiosos

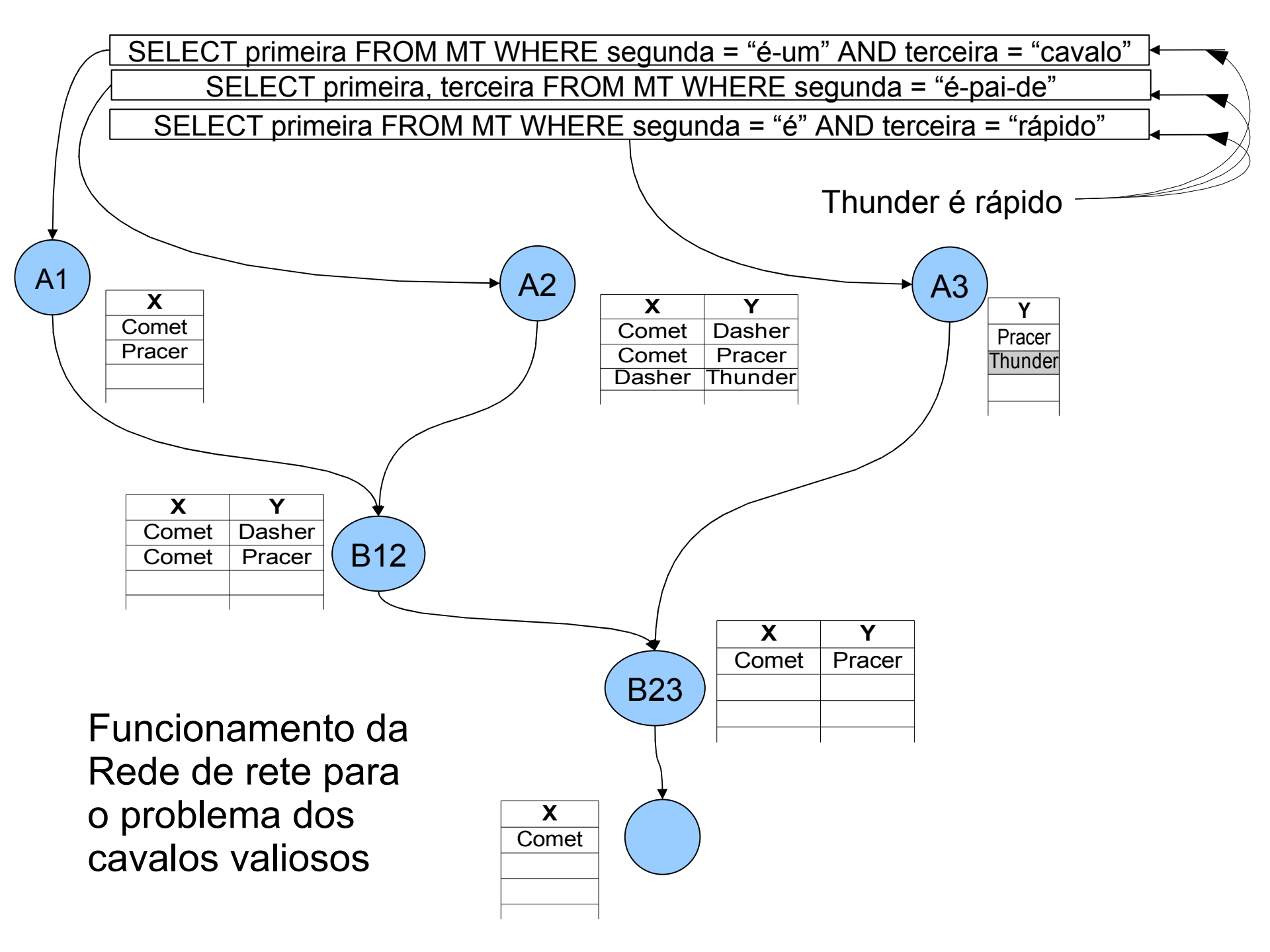


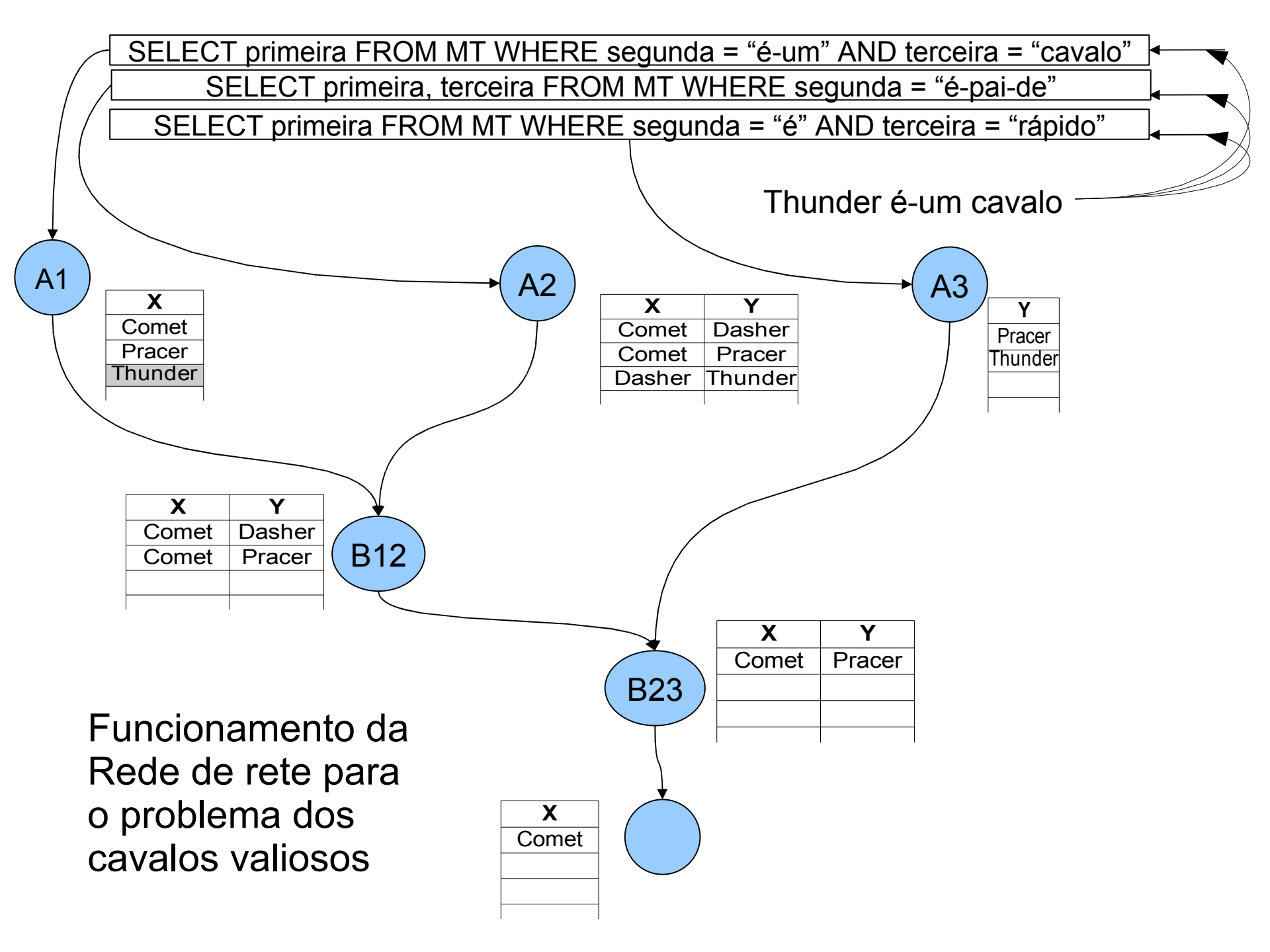
Funcionamento da Rede de rete para o problema dos cavalos valiosos

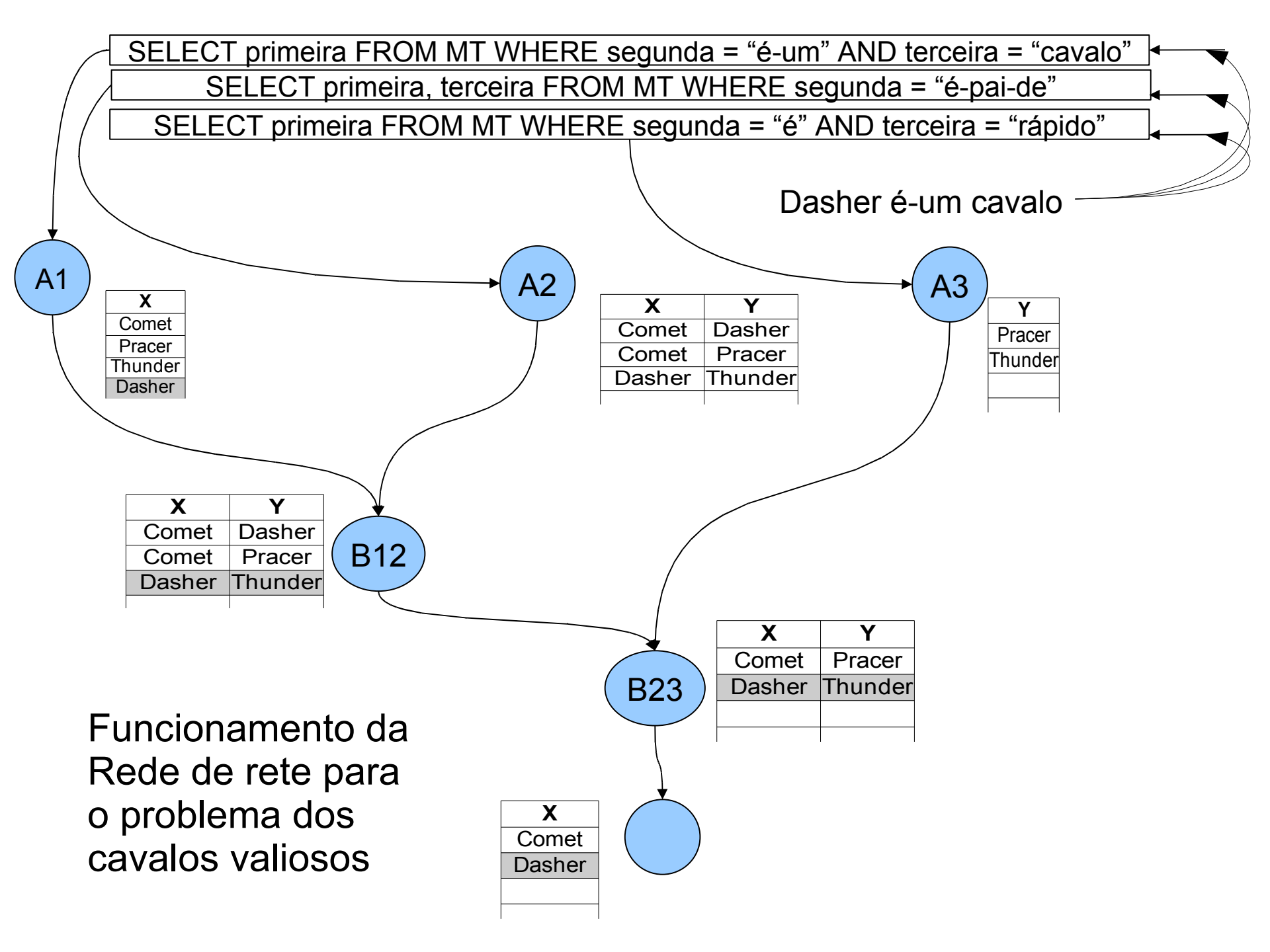

SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Comet é-pai-de Pracer
A1
A2
A3
B12
B23
X
Comet
Pracer
X
Y
Comet
Dasher
Comet
Pracer
Y
X
Y
Comet
Dasher
Comet
Pracer
X
Y
X
Funcionamento da Rede de rete para o problema dos cavalos valiosos

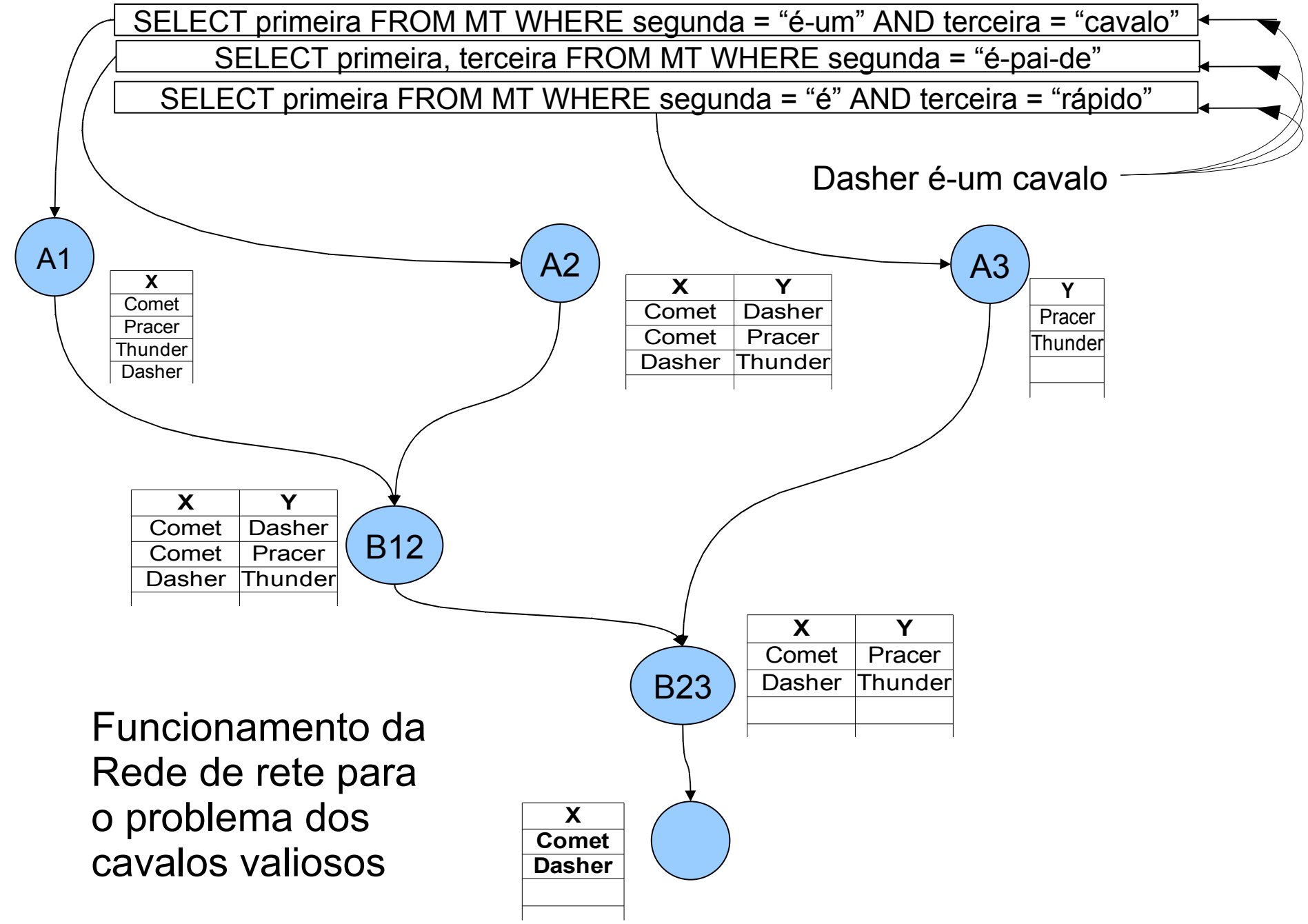

SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Pracer é rápido
A1
A2
A3
B12
B23
X
Comet
Pracer
X
Y
Comet
Dasher
Comet
Pracer
Y
Pracer
X
Y
Comet
Dasher
Comet
Pracer
X
Y
Comet
Pracer
X
Comet
Funcionamento da Rede de rete para o problema dos cavalos valiosos


SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Dasher é-pai-de Thunder
A1
A2
A3
X
Comet
Pracer
X Y
Comet Dasher
Comet Pracer
Dasher Thunder
Y
Pracer
X Y
Comet Dasher
Comet Pracer
B12
X Y
Comet Pracer
B23
X
Comet
Funcionamento da Rede de rete para o problema dos cavalos valiosos


SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Thunder é rápido
A1
A2
A3
B12
B23
X
Comet
Pracer
X
Y
Comet
Dasher
Comet
Pracer
Dasher
Thunder
Y
Pracer
Thunder
X
Y
Comet
Dasher
Comet
Pracer
X
Y
Comet
Pracer
X
Comet
Funcionamento da Rede de rete para o problema dos cavalos valiosos


SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Thunder é-um cavalo
A1
A2
A3
B12
B23
X
Comet
Pracer
Thunder
X
Y
Comet
Dasher
Comet
Pracer
Dasher
Thunder
Y
Pracer
Thunder
X
Y
Comet
Dasher
Comet
Pracer
X
Y
Comet
Pracer
X
Comet
Funcionamento da Rede de rete para o problema dos cavalos valiosos


SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Dasher é-um cavalo
A1
A2
A3
B12
B23
X
Comet
Pracer
Thunder
Dasher
X
Y
Comet
Dasher
Comet
Pracer
Dasher
Thunder
Y
Pracer
Thunder
X
Y
Comet
Dasher
Comet
Pracer
Dasher
Thunder
X
Y
Comet
Pracer
Dasher
Thunder
X
Comet
Dasher
Funcionamento da Rede de rete para o problema dos cavalos valiosos


SELECT primeira FROM MT WHERE segunda = “é-um” AND terceira = “cavalo”
SELECT primeira, terceira FROM MT WHERE segunda = “é-pai-de”
SELECT primeira FROM MT WHERE segunda = “é” AND terceira = “rápido”
Dasher é-um cavalo
A1
A2
A3
B12
B23
X
Comet
Pracer
Thunder
Dasher
X Y
Comet Dasher
Comet Pracer
Dasher Thunder
Y
Pracer
Thunder
X Y
Comet Dasher
Comet Pracer
Dasher Thunder
X Y
Comet Pracer
Dasher Thunder
X
Comet
Dasher
Funcionamento da Rede de rete para o problema dos cavalos valiosos

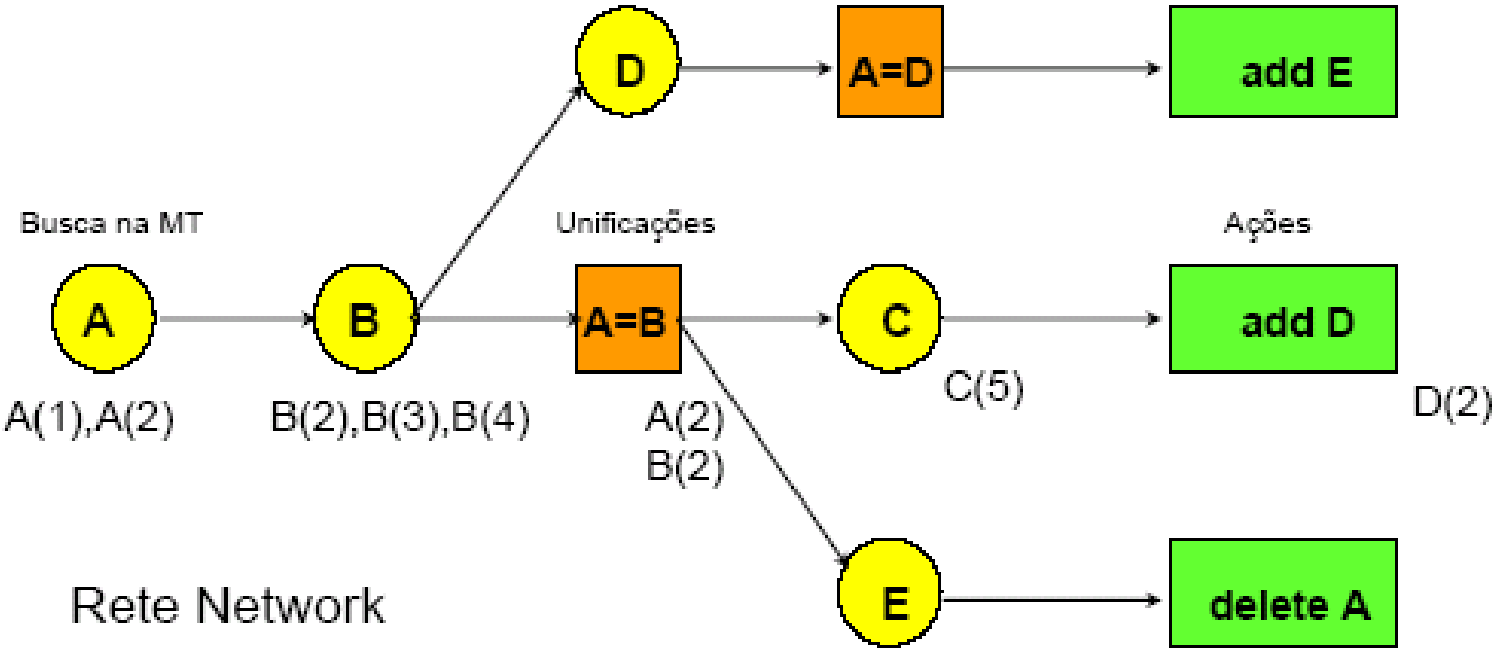
Algoritmo de Rete

Base de Regras:

A(x) ^ B(x) ^ C(y) => add D(x)
A(x) ^ B(y) ^ D(x) => add E(x)
A(x) ^ B(x) ^ E(x) => delete A(x)

Memória de Trabalho:

{A(1),A(2),B(2),B(3),B(4),C(5)}



Fase de Resolução de conflitos

- Estratégias para **resolução de conflitos** entre as regras disparadas
  - **Ordenação das regras**: Ordenar as regras em uma lista de prioridade, executar aquela com maior prioridade
  - **Ordenação de tamanho**: executar a regra tem o maior número de antecedentes
  - **Ordenação de dados**: Ordenar as asserções por prioridade, executar a regra que casa com a asserção de maior prioridade
  - **Refração**: não executar a mesma regra com os mesmos argumentos duas vezes
  - **Recência**: dar preferência as regras que se referem a elementos da MT criados recentemente (simula o foco de atenção do discurso)
  - **Especificidade**: dar preferência as regras que são mais específicas
  - Outras

Exercício

- Mostre os quatro (4) primeiros ciclos de operação do sistema de produção a seguir (se possível) listando:
  - i) o Ciclo; ii) as Regras satisfeitas; ii) a Regra escolhida em cada ciclo e iv) a nova memória de trabalho.
- ?x, ?y e ?z são variáveis e os outros termos são constantes.
- A estratégia de resolução de conflitos:
  - Escolher a regra com o maior número de precondições
  - Nenhuma regra pode ser usada mais de uma vez **com o mesmo conjunto de variáveis atribuídas**
  - Se ainda houver conflito entre as regras, a última regra satisfeita deve ser aplicada.
- ¬P(x) significa que P(x) não pode estar na MT

Exercício

- Regras:

  (R1) IF P(?x, ?y) Q(?x, a, ∧ ?z) ∧ ¬Q(?z, a, ?z)
  THEN assert R(?x)
  retract Q(?x, a, ?z)

  (R2) IF R(?x) ∧ Q(?x, ?y, ?z)
  THEN retract R(?x)

  (R3) IF P(f(?x), ?y) ∧ ?x ≤ ?y ∧ Q(f(?w), ?w, f(?z)) ∧ ?y ≤ ?z
  THEN print SUCCESS!

  (R4) IF R(?x)
  THEN assert Q(f(?x), ?x, f(?x))

- Memoria de Trabalho inicial:

  P(f(1), 2) Q(f(1), a, f(1)) R(f(1))
  P(3, f(4)) Q(4, a, 3)
  P(4, 4)