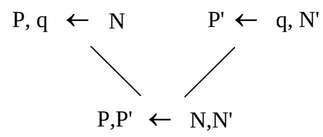
2. RESOLUÇÃO

2.1 RESOLUÇÃO PROPOSICIONAL

Regra de Inferência

$$\frac{\begin{array}{l} P, q \leftarrow N \\ P' \leftarrow q, N' \end{array}}{P, P' \leftarrow N, N'}$$



Definição.

Cada aplicação da regra determina um *passo de resolução.* A cláusula derivada num passo de resolução designa-se por *resolvente* das premissas.

Exemplos.

$$\frac{\begin{array}{l} q \leftarrow p \\ p \leftarrow \end{array}}{q \leftarrow} \qquad \frac{\begin{array}{l} q \leftarrow p \\ \leftarrow q \end{array}}{\leftarrow p} \qquad \frac{\begin{array}{l} q \leftarrow \\ \leftarrow q \end{array}}{\leftarrow \ \square}$$

modus ponens *modus tolens*

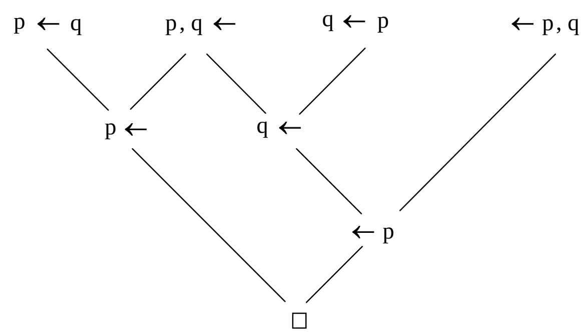
Definição.

Dado um conjunto de cláusulas C, uma *derivação* a partir de C é uma sequência de cláusulas $c_1,\ldots,c_n$, tal que, para todo i, $c_i \in C$ ou c_i é um resolvente de c_j e c_k com $j,k<i$. A cláusula c_n é a conclusão da derivação.

Uma *refutação* de C é uma derivação da cláusula vazia □ a partir de C.

Observação.

A representação arborescente é mais usual



2.2 UNIFICAÇÃO

Definição.

Uma substituição θ é uma função que associa um termo a cada variável.

Observação.

Quando uma substituição difere da identidade num conjunto finito $X_1,\dots,X_n$ de variáveis, denotamo-la através do conjunto $\{X_1/t_1,\dots,X_n/t_n\}$, onde $t_i=\theta(X_i)=\theta X_i$.

Note-se que nem todo o conjunto de pares X_i/t_i denota uma substituição: os X_i têm de ser todos diferentes!

Definição.

A *instanciação de uma expressão* e (termo ou fórmula atómica) *por uma substituição* θ é a expressão que se obtém de *e* por substituição de cada variável X pelo termo θX:

- para todo o símbolo de função f de aridade n, $\theta f(t_1,\dots,t_n)$ é $f(\theta t_1,\dots,\theta t_n)$,
- para todo o símbolo de predicado p de aridade n, $\theta p(t_1,\dots,t_n)$ é $p(\theta t_1,\dots,\theta t_n)$.

A expressão θe é a *instância* de e obtida por θ.

Uma instância θe de e diz-se *chã* se θe é uma expressão chã.

Definição.

O conceito de instanciação pode estender-se a conjuntos de expressões $\theta E = \{\theta e: e \in E\}$ e a cláusulas $\theta(P \leftarrow N) = (\theta P \leftarrow \theta N)$.

Definição.

A *composição de substituições* é a correspondente composição de funções: $(\theta\circ\sigma)X = \theta\sigma X = \theta(\sigma X)$ (o que implica $\theta\sigma e = \theta(\sigma e)$ para qualquer expressão *e*).

Em consequência, a composição de substituições é uma operação associativa e tem como elemento neutro a *atribuição identidade (vazia)*.

Observação.

Têm especial interesse as substituições *idempotentes*, i.e. as que satisfazem a propriedade $\theta\circ\theta=\theta$.

Exemplo.

{X/f(Y), Y/b} não é idempotente.

{X/f(b), Y/b} é idempotente

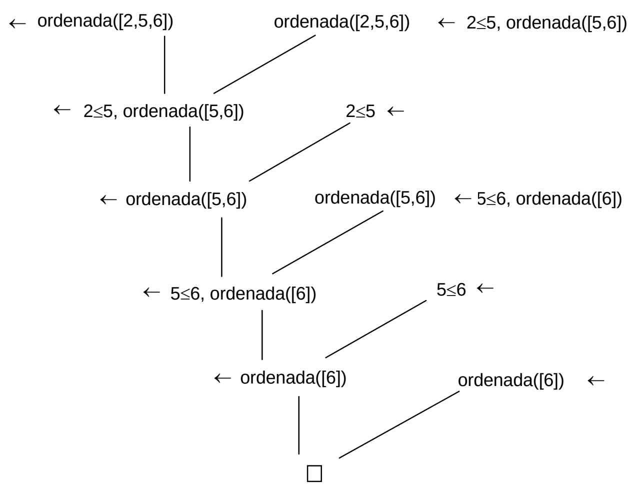
Definição.

Designa-se por *resolução chã* de um conjunto de cláusulas C toda a aplicação da resolução proposicional a instâncias chãs das cláusulas de C.

Exemplo.

ordenada([X]) ←
ordenada([X,Y|Z]) ← X≤Y, ordenada([Y|Z])
← ordenada([2,5,6])
/* mais a definição de ≤ */

instâncias chãs:
ordenada([6]) ←
ordenada([5,6]) ← 5≤6, ordenada([6])
ordenada([2,5,6]) ← 2≤5, ordenada([5,6])
← ordenada([2,5,6])



***Questão*.**

Como saber que instâncias gerar?

Como responder a interrogações que contêm variáveis, por exemplo,
← ordenação([5,4,6],X)?

***Definição*.**

Seja S um conjunto finito de fórmulas atómicas.

Uma substituição θ diz-se *unificador* de S se e só se θS é um conjunto singular.

S diz-se *unificável* se e só se admite um unificador.

Um unificador θ de S diz-se um *unificador mais geral* (UMG) se e só se, para todo o unificador σ de S, existe uma substituição γ tal que $\sigma=\gamma\theta$.

***Exemplo*.**

S={p(X,f(Y)), p(f(U),f(Z))} é unificável:
σ=[X/f(a), Y/Z, U/a] é um unificador pois σS={p(f(a),f(Z))};

θ=[X/f(U), Y/Z] é um UMG tal que θS={p(f(U),f(Z))} e σ=[U/a]θ.

S={p(X,f(Y)), p(f(U),g(Z))} não é unificável.

***Definição*.**

Uma *mudança de variáveis* é uma substituição injectiva cujo contradomínio consiste apenas de variáveis (isto é, substitui variáveis por variáveis).

***Proposição*.**

Um UMG é único a menos de uma mudança de variáveis. Quer dizer, se θ e σ são UMGs de um conjunto S, existe uma mudança de variáveis γ tal que $\sigma=\gamma\theta$.

***Proposição*.**

Todo o conjunto unificável admite um unificador mais geral.

<u>prova</u>
Existe um algoritmo que determina se um dado conjunto é unificável e, no caso afirmativo, determina um UMG.

O algoritmo permite encontrar um UMG para duas expressões $p(t_1,\ldots,t_n)$ e $p(r_1,\ldots,r_n)$. Usa uma pilha S na qual guarda os pares de termos a unificar, inicializada com os pares $<t_i,r_i>$.

Recorre-se ainda a uma variável Booleana para indicar a impossibilidade de unificar as duas expressões.

O UMG θ é construído progressivamente começando com a identidade.

begin

θ:=id; b:=true; S:=[<t_n,r_n>,…,<t_1,r_1>];

while (S$\neq$[] **and** b) **do**

<t,r>:=top(S); S:=pop(S); <t,r>:=<θt,θr>;

if t e r são da forma f(t_1,…,t_n), g(r_1,…,r_m) com (f,n) e (g,m) distintos (n,m$\geq$0)
then b:=false

else if t e r são da forma f(t_1,…,t_n), f(r_1,…,r_n)
then S:=push(<t_n,r_n>,push(…,push(<t_1,r_1>,S)…)

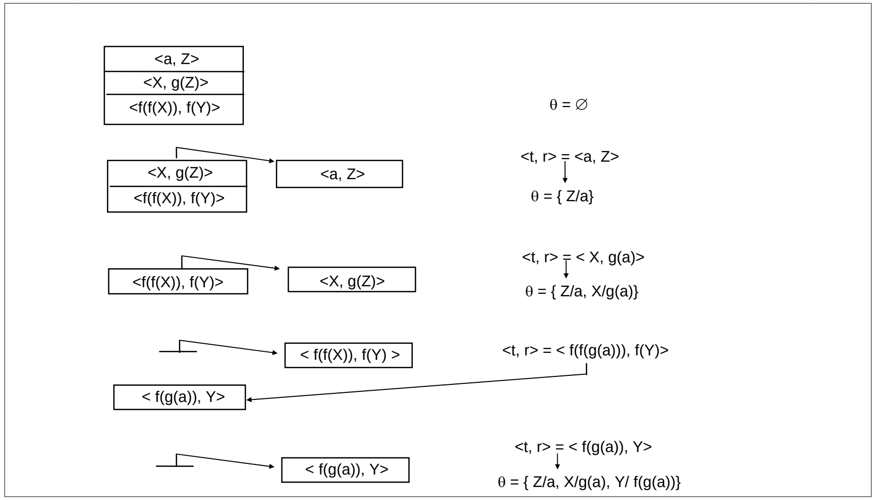
else if t ocorre estritamente em r ou r ocorre estritamente em t
then b:=false

else if t é uma variável **then** θ:=[t/r]θ

else if r é uma variável **then** θ:=[r/t]θ

end

Exemplo: UMG para p(f(f(X)), X, a) e p(f(Y), g(Z), Z)



Observação.

Occur Check
else if t ocorre estritamente em r ou r ocorre estritamente em t

Pretende-se, desta maneira, detectar situações de auto-referência, como em {p(f(X),f(f(X))),p(Y,Y)}.

Na ausência deste teste, obter-se-ia a substituição X/f(X), que corresponderia a substituir X pelo termo infinito f(f(f…)), o qual não pertence à linguagem de primeira ordem.

A verificação da condição é "cara" pois exige um exame à estrutura do termo a fim de se detectar a ocorrência da variável. Por isso, ela é frequentemente omitida nas implementações do PROLOG, com a possível perda da correcção! (Exemplo, resposta a $\leftarrow$ igual(X,suc(X)).)

2.3 RESOLUÇÃO DE 1ª ORDEM

Definição.

Um par de *separadores* para cláusulas c e c' é um par (θ,θ') de mudanças de variável tais que θc e θ'c' não possuem variáveis em comum.

Definição.

Dadas duas cláusulas $c_1 = P_1 \leftarrow N_1$ e $c_2 = P_2 \leftarrow N_2$, a cláusula $P \leftarrow N$ é um resolvente de c_1 e de c_2 se e só se, sendo (θ_1,θ_2) um par de separadores para c_1 e c_2:

- existem $P' \subseteq P_i$ e $N' \subseteq N_j$, com $i \neq j$, tais que $\theta_i P' \cup \theta_j N'$ é unificável;
- para algum UMG σ de $\theta_i P' \cup \theta_j N'$ tem-se
  $P = \sigma(\theta_i(P_i \setminus P') \cup \theta_j P_j)$ e $N = \sigma(\theta_j(N_j \setminus N') \cup \theta_i N_i)$.

Observação.

Cada passo de resolução fica caracterizado pelo quíntuplo (c_1, c_2, θ_1, θ_2, σ).

$$\frac{\begin{array}{l} P'_1,\ \underline{\mathbf{P'}} \ \leftarrow\ N_1 \\ \qquad P_2 \ \leftarrow\ N'_2,\ \underline{\mathbf{N'}} \end{array}}{\sigma\theta_1 P'_1,\ \sigma\theta_2 P_2\ \leftarrow\ \sigma\theta_1 N_1,\ \sigma\theta_2 N'_2}$$

Proposição. (Correcção)

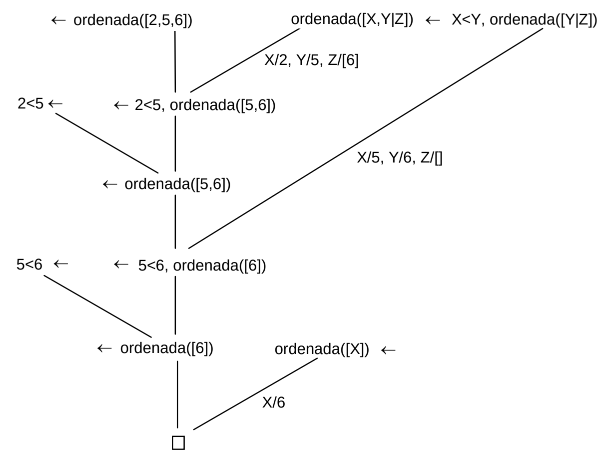
Todo o conjunto de cláusulas refutável é contraditório.

Proposição. (Completude)

Todo o conjunto de cláusulas contraditório é refutável.

(Esta propriedade depende de P' e N' poderem ser conjuntos não singulares).

Exemplo.



2.4 RESOLUÇÃO SLD

As refutações por resolução são "pesadas" e "caras", no caso geral.

No entanto, em certos fragmentos, podemos definir procedimentos de refutação eficientes.

É este o caso das **cláusulas de Horn**:

$q \leftarrow p_1, \ldots, p_n$	cláusula de programa
$\leftarrow p_1, \ldots, p_n$	objectivo

SL**<u>D</u>**: D de <u>d</u>elimitada (<u>d</u>efinite)

Programa é um conjunto (finito) de cláusulas delimitadas (cláusulas de programa).

S**<u>L</u>**D: L de <u>l</u>inear

Cada passo de resolução (aplicação da regra) usa como premissa (*cláusula central*) o resolvente mais recente.

Observação.

Partindo de um objectivo e um programa, uma derivação (S)LD só gera objectivos como resolventes.

<u>prova</u>: por indução no comprimento da derivação, notando que, em cada passo, sendo a resolução linear, uma das premissas é um objectivo (o resolvente anterior) e que os objectivos só admitem como segunda premissa uma cláusula de programa. Isto é, num passo de resolução (S)LD, a cláusula central é um objectivo e a outra premissa (*cláusula lateral*) é uma das cláusulas do programa. Ora, o resolvente de um objectivo e de uma cláusula de programa é ainda um objectivo.

<u>**S**</u>LD: S de <u>s</u>elecção

Regra de computação: selecciona em cada passo a fórmula atómica do objectivo que vai ser eliminada.

***Definição*.**

Um objectivo diz-se *derivável de um programa **P** e de um objectivo **G** via uma regra de computação **S*** quando é o resolvente de um passo de resolução (**G**,c,Id,θ,σ), tal que c$\in$**P** e a fórmula eliminada é **S(G)**.

Quer dizer, se **G** for $\leftarrow p_1,\ldots,p_k,\ldots p_m$, tal que p_k=**S**(**G**), e c for $q \leftarrow q_1,\ldots,q_n$ (o que exige que $\{p_k,\theta q\}$ seja unificável para o par (Id,θ) de separadores), o resolvente é

$\leftarrow \sigma(p_1,\ldots,p_{k-1},\theta q_1,\ldots,\theta q_n,p_{k+1},\ldots,p_m)$

onde σ é um UMG de $\{p_k,\theta q\}$.

***Definição*.**

Uma *derivação SLD para um programa **P** e um objectivo $\boldsymbol{G_0}$ via uma regra de computação **S*** é uma sequência $\mathbf{G_0}$, G_1, … G_l ($l\leq\omega$) de objectivos tal que, para cada i<l, G_{i+1} é derivável de **P** e G_i via **S**.

Cada derivação tem associada a sequência $(G_i,c_i,Id,\theta_i,\sigma_i)$ dos quíntuplos para cada passo de resolução (*historial* da derivação).

Uma *refutação SLD para **P** e $\boldsymbol{G_0}$ via **S*** é uma derivação SLD para **P** e $\mathbf{G_0}$ via **S** que termina com □ (derivação *bem sucedida*).

Se l é o comprimento da refutação, a composição $\sigma_{l-1}\ldots\sigma_0$ (restringida às varáveis que ocorrem em $\mathbf{G_0}$) diz-se *resposta calculada* pela refutação.

Uma *derivação falhada* é uma derivação finita que não termina com □e que não pode ser continuada (identificada com n).

***Definição*.**

Uma *resposta correcta* para um programa **P** e um objectivo **G** é uma substituição θ tal que, para cada fórmula atómica f em **G**, **P**$\models(\forall)\theta f$.

***Proposição*.**

Seja **P** um programa, **G** um objectivo e **S** uma regra de computação.

1– Correcção:
Toda a resposta calculada por uma refutação SLD para **P** e **G** via **S** é correcta.

2– Completude:
Se θ é uma resposta correcta para **P** e **G**, então existe uma substituição γ e uma refutação SLD de **P** e **G** via **S** que calcula uma resposta σ tal que $\theta=\gamma\sigma$.

<u>prova</u>: no capítulo 4.

***Observação*.**

A regra de computação **S** é qualquer.

Fixando uma regra de computação **S**, pode haver mais do que uma refutação para **P** e **G** via **S** (*não-determinismo*).

***Definição*.**

A *árvore SLD (de computação) para um programa **P** e um objectivo **G** via uma regra de computação **S*** define-se do seguinte modo

- a sua raiz é **G**
- cada nó tem um filho para cada um dos objectivos deriváveis de **P** via **S** (um filho por cada uma das cláusulas do programa cuja cabeça pode ser unificada com a fórmula do objectivo seleccionada por **S**).

Observação.

Cada ramo da árvore de computação é uma derivação para **P** e **G** via **S**.

Um ramo finito com folha □(sucesso) indica uma derivação bem sucedida (refutação).

Um ramo finito sem folha □indica uma derivação falhada (completa-se com a folha n de insucesso).

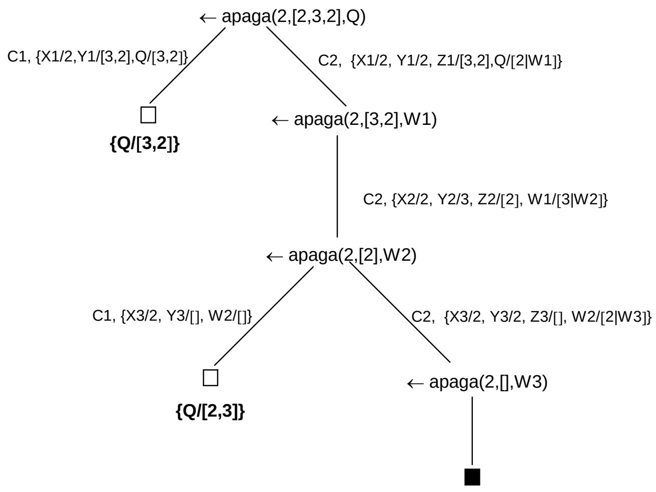
Pode haver ramos infinitos!

É costume anotar a árvore com os UMGs.

Exemplo.

C1: apaga(X,[X|Y],Y) ←

C2: apaga(X,[Y|Z],[Y|W]) ← apaga(X,Z,W)



Proposição. (Independência da Regra de Computação)

Duas árvores SLD para o mesmo programa e objectivo, mas via regras de computação diferentes, têm o mesmo número de ramos bem sucedidos e calculam as mesmas respostas.

Observação.

As árvores podem ter formas muito distintas, diferindo em particular nos ramos falhados ou infinitos.

Exemplo. *Dinastia_1*

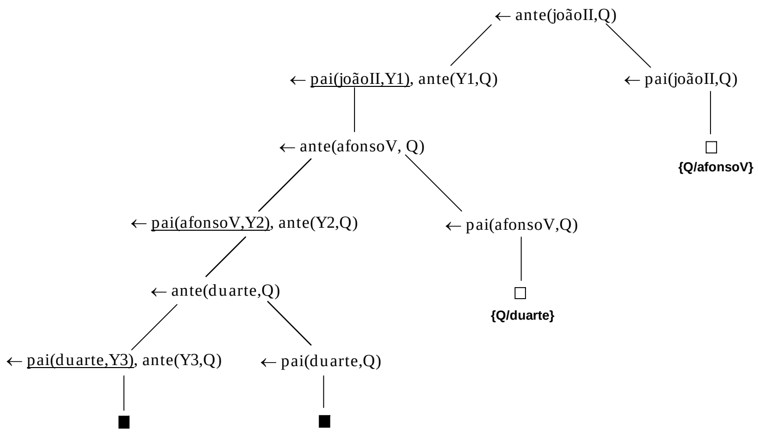
ante(X,Z) ← pai(X,Y), ante(Y,Z)

ante(X,Z) ← pai(X,Z)

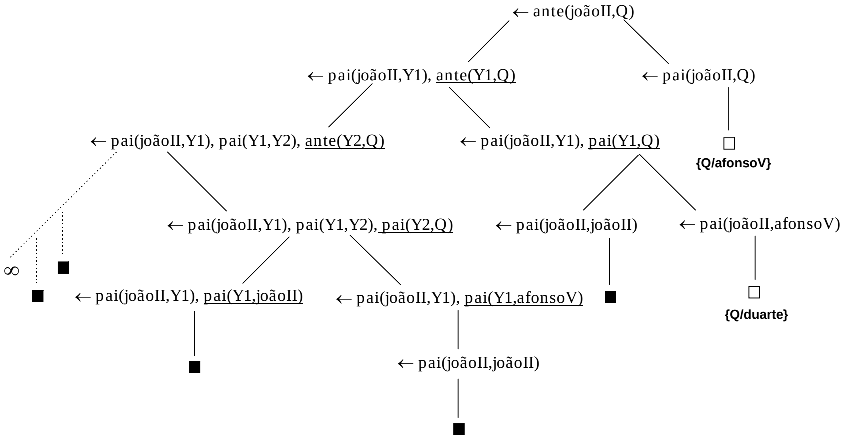
pai(joãoII,afonsoV) ←

pai(afonsoV,duarte) ←

Selecção do Primeiro



Selecção do Último



Execução (Pesquisa na Árvore de Computação)

Meta.

Determinar a existência de um ramo bem sucedido (refutação). Possivelmente, determinar todas as respostas calculáveis (uma por cada ramo bem sucedido).

Questão.

Que estratégia usar para explorar a árvore? (Regra de Pesquisa.)
Como construir a árvore ?

Observação.

A necessidade de uma tal estratégia é característica dos formalismos *relacionais* (existência de mais de uma resposta) em oposição aos formalismos *funcionais*.

Execução Sequencial Descendente, em Profundidade, com Retrocesso

- Relativamente a cada nó, atribui-se uma prioridade a cada um dos seus filhos (atribuir uma prioridade a cada uma das cláusulas do programa cuja cabeça pode ser unificada com a fórmula do objectivo seleccionada pela regra de computação).
- A execução começa com a geração da raiz.
- Encontrando-se o controlo num nó, gera-se, e passa-se-lhe o controlo, o filho de maior prioridade entre os que ainda não foram gerados.
- Se não houver filhos, ou se todos os filhos já tiverem sido gerados, então o controlo é passado ao ascendente mais próximo que ainda tem filhos por gerar. Se tal ascendente não existe, é porque a árvore já foi totalmente gerada, terminando-se então a execução.

Observação.

Esta é a estratégia implementada pelo PROLOG, sendo a prioridade determinada pela ordem das cláusulas no programa (*estratégia padrão*).

(Em certos sistemas PROLOG, a execução é suspendida quando se encontra a cláusula vazia.)

Exemplo.

Dinastia_1 com as regras padrão (selecção do primeiro predicado e da primeira cláusula).

Penúltima árvore.

Execução Descendente, em Largura

Todos os nós de uma geração são gerados antes de passar à geração seguinte (sendo **P** finito, cada geração é finita).

Observação.

Se a árvore é finita, a escolha da estratégia de pesquisa é inconsequente (apenas a ordem das respostas varia).

A pesquisa em profundidade optimiza a utilização de memória. No entanto, se a árvore possui um ramo infinito, a execução nunca sai desse ramo, e as respostas calculáveis nos outros ramos não serão geradas (inadequação).

A pesquisa em largura garante que cada resposta calculável é gerada em tempo finito (se bem que a pesquisa numa árvore infinita se prolongaria indefinidamente). No entanto, exige demasiada utilização de memória.

PROLOG

Regra de Computação
Escolha da primeira fórmula atómica do objectivo.

Regra de Pesquisa
Descendente, em profundidade, escolhendo as cláusulas do programa de acordo com a ordem pela qual ocorrem no programa.

Observação.

Uma vez fixadas, as *regras de computação e de pesquisa* permitem escolher, entre programas logicamente equivalentes, os que têm uma execução mais eficiente fazendo variar a ordem dos predicados no corpo das cláusulas, e a das cláusulas no corpo do programa.

É neste sentido que se *programa* (vs representação).

Ordem das Cláusulas

Afecta a prioridade dada aos filhos de cada nó. Logo, determina o percurso de construção.

Exemplo.

Comparar o percurso de construção da árvore de computação para o objectivo ← ante(joãoII,Q) e o programa

Dinastia_2:
ante(X,Z) ← pai(X,Z)
ante(X,Z) ← pai(X,Y), ante(Y,Z)
pai(joãoII,afonso_V) ←
pai(afonsoV,duarte) ←

com o caso anterior.

ORDEM DOS PREDICADOS:

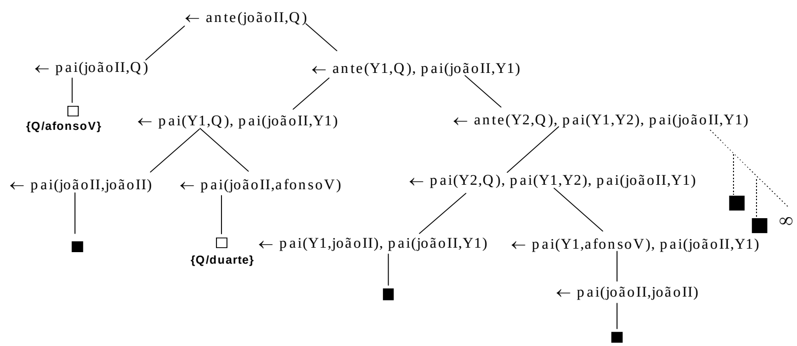
Condiciona a selecção da fórmula atómica a eliminar. Logo, determina a *forma da árvore*.

Exemplo.

Comparar a árvore de computação para ← ante(joãoII,Q) e o programa

Dinastia_3
ante(X,Z) ← pai(X,Z)
ante(X,Z) ← ante(Y,Z), pai(X,Y)
pai(joãoII,afonsoV) ←
pai(afonsoV,duarte) ←

com o caso anterior.



Questão.
Que aconteceria se a ordem das cláusulas do procedimento *ante* fosse trocada? (*Num programa recursivo, a base deve estar antes do passo!*)

Exemplo.

- Geração de permutações, aceitando as que estão ordenadas:

  ordenação_de(X,Y) ← perm(X,Y), ordenada(Y)
  ← ordenação_de([2,5,6,2,4,1,8,7],Q)

- Geração de listas ordenadas (*ad hoc*), aceitando as que são permutações (qual é o tamanho da árvore?):

  ordenação_de(X,Y) ← ordenada(Y), perm(X,Y)
  ← ordenação_de([2,5,6,2,4,1,8,7],Q)

- E se o objectivo fosse ← ordenação_de(Q,[2,5,6,2,4,1,8,7])? A eficácia da ordem depende do objectivo.