1 Programas como árvores

Será conveniente termos uma interpretação mais visual de programas lógicos. Isso nos dará uma ideia melhor de em qual ordem ocorrem as tentativas de unificações e como lidar com elas. Considere o seguinte programa, representando um pedaço da árvore genealógica da dinastia Julio-Claudiana, junto com a relação ancestral[1]

```
filhx(julia_caesaris, augustus).
filhx(julia_caesaris, scribonia).

filhx(agrippina, marcus_vipsanius).
filhx(agrippina, julia_caesaris).

filhx(caius_caesar, agrippina).          % Calígula
filhx(caius_caesar, germanicus).

filhx(julia_drusilla, caius_caesar).
filhx(julia_drusilla, caesonia).

filhx(nero, agrippina).                  % Nero
filhx(nero, gnaeus_ahenobarbus).

ancestral(A, B) := filhx(B, A), !.
ancestral(A, B) := filhx(B, C), ancestral(A,C).
```

Código 1: Ancestral 0

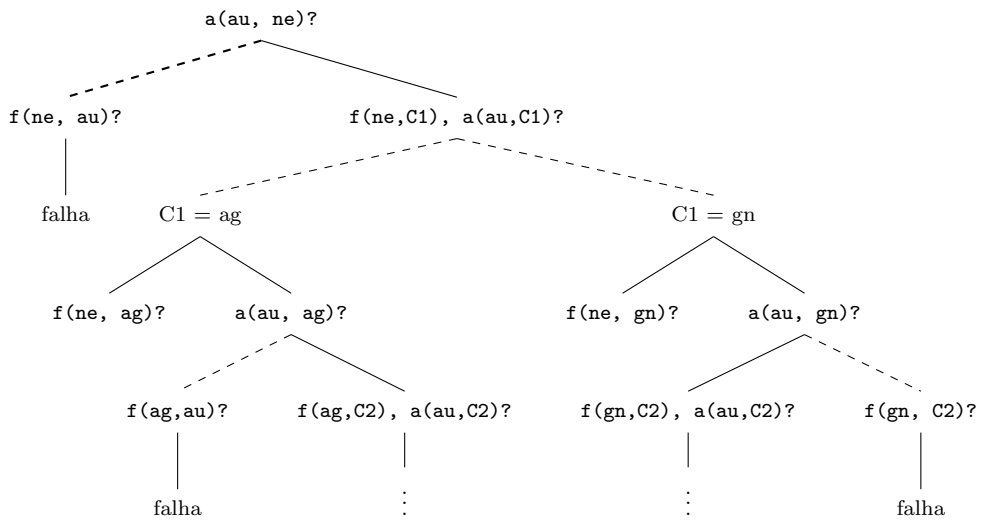
Por conveniência, no lugar das relações filhx/2 e ancestral/2 e dos nomes mostrados acima, usaremos os mostrados no Código 2

[1] Os romanos, assim como outros povos, tinham o hábito de atribuir outro nome às pessoas, com base em características físicas, psicológicas, de caráter ou de hábito. Assim, por exemplo, Claudius significa "manco", Calígula significa "bota de (pequeno) soldado", Augustus significa "majestico" ou "venerável" e, Sulla, significa "manchado".

```
f(ju, au).
f(ju, sc).

f(ag, ma).
f(ag, ju).

f(ca, ag).
f(ca, ge).

f(ju_d, ca).
f(ju_d, cae).

f(ne, ag).
f(ne, gn).

a(A, B) := f(B, A), !.
a(A, B) := f(B, C), a(A,C).
```

Código 2: Ancestral 1

O goal `a(au, ne)?` define implicitamente uma árvore, que começa da forma a seguir.



A árvore inteira não será colocada por questões de espaço, mas continua de maneira semelhante. Os ramos tracejados indicam um "ou" lógico, enquanto que os sólidos indicam um "e" lógico. O que isso significa é que falha em um dos ramos dos arcos sólidos indica que todo o arco é falho, mas uma falha em

um ramo tracejado só afeta o ramo tracejado.

A execução de um programa lógico consiste em uma busca, em uma árvore como essa, por uma folha de "sucesso". Se, ao invés de "sucesso", é encontrada "falha", é feito o *backtracking*. Quando alguém fala sobre um "modelo computacional de programa lógico", fala essencialmente sobre como percorrer essa árvore e como fazer esse *backtracking*. Como pode ver, ela pode crescer muito em largura a cada nível (não cresceu tanto no exemplo acima porque colocamos uma quantidade reduzida de substituições). Por isso, é compreensível que prefiramos uma busca em profundidade do que em largura. Isso pode, é claro, levar a problemas técnicos e filosóficos, já que impõe uma escolha arbitrária sobre a ordem de avaliação das cláusulas e dos termos de cada cláusula, o que pode levar a comportamentos inesperados ao programador desatento a esse modelo[2]. É claro, dizer apenas que a busca é em profundidade não é o suficiente: precisamos dizer que ela é em profundidade e à esquerda.

O seguinte exemplo, um homem reclamando de sua vida, mostra que questões de parentesco podem ser bem mais complicadas do que isso (o exemplo é baseado na história retirada de [1], frequentemente atribuída a Mark Twain):

Me casei com uma viúva com uma filha crescida. Meu pai, que nos visitava frequentemente, se apaixonou com a filha e a tomou como sua esposa. Isso fez do meu pai o meu filho adotado, e, de minha filha adotada, minha madrasta.
Depois de um ano, minha esposa deu à luz um filho, que se tornou o irmão adotado do meu pai e, ao mesmo tempo, meu tio, já que ele era o irmão de minha madrasta.
Mas a esposa de meu pai, isto é, minha filha adotada, também deu à luz um filho. Então, ele era meu irmão e também meu neto, já que ele era o filho de minha filha.
Isso quer dizer que eu me casei com minha avó, já que ela era a mãe de minha mãe. Como o marido de minha esposa, eu também era o neto adotado dela.
Nossos amigos dizem que eu sou meu próprio avô. Isso é verdade?

O personagem dessa história, não sabendo Prolog, teve uma certa dificuldade em responder a essa questão. Mas nós, como que por reflexo, fazemos o seguinte programa:

[2]Convém lembrar que não estamos lidando com programação paralela. No modelo de programação paralela a avaliação pode ocorrer de maneira diferente.

```
%%
% pai(?Pai, ?Filhx).
% mae(?Mae, ?Filhx).
% avoh(?Avó, ?Netx).
% avo(?Avô, ?Netx).
% irmao(?Pai, ?Irmao1, ?Irmao2).
% tio(?Pai, ?Tio, ?Sobrinhx).


avo(Avo, Neto):-
         pai(Avo, Avo_filho),
         pai(Avo_filho, Neto).

avoh(Avoh,Neto):-
         mae(Avoh, Filha_da_avoh),
         mae(Filha_da_avoh, Neto).

irmao(Pai, Irmao1, Irmao2):-
       pai(Pai, Irmao1),
       pai(Pai, Irmao2).

tio(Pai, Tio, Sobrinho):-
       irmao(_,Pai,Tio),
       pai(Tio, Sobrinho).


% Eu sou o filho do meu pai
pai(meu_pai, eu).

% Meu pai é meu filho adotado
pai(eu, meu_pai).

% Eu sou o pai do filho da minha esposa
pai(eu, filho_meu_e_de_minha_esposa).

% Meu pai é o pai do filho_de_minha_filha_adotiva
pai(meu_pai, filho_de_minha_filha_adotiva).

% Minha filha adotiva é minha madastra
mae(minha_filha_adotiva, eu).

% Minha esposa é a mãe de minha filha adotiva
mae(minha_esposa, minha_filha_adotiva).



% O filho meu e de minha esposa é o irmão adotado de meu pai
% O filho meu e de minha esposa é meu tio
% Filho de minha filha adotiva é meu irmão
% Filho de minha irmã adotiva é meu neto
% Minha esposa é minha avó
% Eu sou meu avô
              irmao(_, meu_pai, filho_meu_e_de_minha_esposa).
              tio(filho_meu_e_de_minha_esposa, meu_pai, eu),
              irmao(_, filho_de_minha_filha_adotiva, eu),
              avo(eu, filho_de_minha_filha_adotiva),
```

O último bloco de código na verdade não faz parte do programa, mas foi deixado por conveniência. Ele indica o goal (na verdade, só queremos saber se o pobre coitado é avô de si mesmo, mas os outros termos foram deixados por completeza). Você tem ideia de como é a árvore de busca para esse goal?

Leituras adicionais

[1] Niederliński, Antoni, "A gentle guide to constraint logical programming via Eclipse", 3rd edition, Jacek Skalmierski Computer Studio, Gliwice, 2014

[2] Russell, Bertrand (1919), Introduction to Mathematical Philosophy, George Allen and Unwin, London, UK. Reprinted, John G. Slater (intro.), Routledge, London, UK, 1993
Esse livro está datado em alguns pontos, mas permanece interessante. Está gratuitamente disponível (em inglês) em `http://people.umass.edu/klement/russell-imp.html`