1 Modelo Computacional de Programas Lógicos

Considere o seguinte programa:
$resistor(power, nl)$.
$resistor(power, n2)$.
$transistor(n2, ground, nl)$.
$transistor(n3, n4, n2)$.
$transistor(n5, ground, n4)$.

inverter(Input, Output) :-
transistor(input, ground, Output),
resistor(power, Output).

nand_gate(Input 1 ,Input2,Output) :-
transistor(Inputl,X,Output),
transistor(Input2,ground,X),
resistor(power,Output).

and_gate(Inputl,Input2,Output) :-
nand_gate(Inputl,Input2,X),
inverter(X,Output).

Qual será o resultado do goal $and_gate(In1, In2, Out)$?, a leitora pode se perguntar. Mais do que isso, ela pode se perguntar "Será que, dado um programa qualquer e um goal qualquer dá para *calcular* o resultado do goal?". Te convido a refletir por alguns momentos sobre essa questão.

A leitora pode imaginar que, se houvessem muitos programas com goals de resultados incalculáveis, programação lógica não seria tão útil, então esse não deve ser o caso. Mas mais do que isso, com algumas hipóteses (mais ou menos) razoáveis, temos o seguinte resultado (estabelecido por A. Robinson):

çãoΩção

Existe um algoritmo, chamado algoritmo de unificação, que checa se um goal é consistente com um programa lógico (isto é, se ele *sucede* ou *falha*) e, se for, calcula uma substituição para ele.

A leitora atenta vai notar que ainda não disse o que quero dizer com "uma substituição", mas isso vai ficar claro logo. Ela vai notar também que esse resultado não nos provê um método para resolver o nosso problema. Isso é resolvido pelo **algoritmo de Martelli-Montanari**:

(a) Escolha uma das proposições $p_i(t_1, ..., t_n)$ presentes no goal (lembre-se que um goal é entendido como uma conjunção de proposições);

(b) Das proposições presentes no programa, escolha não-determinísticamente uma que tenha a forma de uma das abaixo e realize a ação especificada:

1. $f(k_1, ..., k_m) = p_i(t_1, ..., t_n)$, onde $m = n$, $f = p_i$: troque t_l por s_l;

2. $f(k_1, ..., k_m) = p_i(t_1, ..., t_n)$, onde $f \neq p_i$: delete essa equação e pare, retorne com falha;
3. $x = x$: delete a equação;
4. $x = y$, onde x é uma variável, mas y não: troque x por y em todas as proposições consideradas;
5. $x = y$, onde x é variável de y (isto é, y é algo como $y(a_1, ..., x, ..., a_n)$): pare e retorne com falha.

(c) Se existe mais algum p_i a ser escolhido, volte ao item (a), se não retorne com sucesso.

Antes de estudarmos mais a fundo o funcionamento do algoritmo de Martelli-Montanari, precisamos saber o que é uma *substituição*. Intuitivamente, ela é o que parece ser: a substituição de uma variável por um valor atômico ou, possívelmente, uma outra variável. Mais em geral, dada a cláusula **P**: `p(`A_1`, ..., `A_n`)`, uma substituição ŏ sobre P, denotada Pŏ, é um conjunto (possívelmente vazio) de atribuição de valores às variáveis de A_i de P. Uma atribuição é escrita como $A_i = B$, onde o símbolo "=" deve ser entendido como usado em álgebra (isto é, como denotando uma relação simétrica de igualdade entre A_i e B, não como geralmente usado em programação, como um operador de atribuição). Não pense em "=" como um operador de atribuição, mas sim como algo que expressa a relação de dois termos terem o mesmo valor.

Até então, evitamos lidar com variáveis muito a fundo. Isso porque elas adicionam complicações importantes (na verdade, essenciais, como logo ficará patente). Mas voltando ao algoritmo de unificação...

Intuitivamente, ele tenta provar o goal a partir do programa de forma construtiva: tenta construir uma solução por meio de substituições e, se não chegar a uma contradição séria demais no processo (uma contradição que não pode ser corrigida. continue lendo), termina com sucesso, "retornando" (não no sentido de uma função que retorna um valor, mas no de mostrar ao usuário do programa) a substituição realizada (na verdade, extritamente falando, ele não precisa "retornar" as substituições, mas assumiremos que retorna).

No item (a), escolhemos um dos termos do goal para provar. Isso porque, como discutido anteriormente, o goal é verdadeiro se cada um de seus termos o for: Provando todos os p_i, provamos o goal.

Em seguida, no item (b), escolhemos não-determínisticamente[1] uma das cláusulas (digamos, a cláusula m_j) do programa para provar p_i. Essa operação pode ser vista como a tentativa de unificação de p_i com m_j, e assim nos referiremos a ela.

[1] No geral, podem existir várias escolhas possíveis e pode ser que por algumas sequências de escolhas de cláusulas podemos nunca chegar a uma prova do goal, apesar de ele ser deduzível a partir de outras cláusulas. Quando dizemos que a escolha é não-determinística, queremos dizer que, se existem conjuntos de escolhas que provem o goal, um desses conjuntos é escolhido (a escolha é feita entre as cláusulas que podem provar o goal). Na prática, isso pode ser implementado apenas aproximadamente, mas, aind assim, é uma abstração que pode levar a aplicações interessantes, em particular de, assim chamados, *programas não-determinísticos*.

Os itens de 1 a 5 são referentes a um passo da unificação.
Veja o seguinte exemplo. Digamos que se tenha o seguinte programa P: