ASSINATURA DE EXPRESSÕES SÉRIE-PARALELAS READ-ONCE

Lucas Carraro e André Inácio Reis
UFRGS, Porto Alegre, Brasil
{lcarraro, andreis}@inf.ufrgs.br

INTRODUÇÃO

Na síntese lógica, presente no fluxo de projeto de circuitos integrados digitais, o processo conhecido como mapeamento tecnológico é essencial. Essa etapa visa a construção de blocos lógicos produzidos a partir de bibliotecas de células, que serão usadas nos algoritmos de otimização do circuito final. Um procedimento para gerar essas bibliotecas foi desenvolvido previamente pelo nosso grupo[1]. Esse algoritmo utiliza o método aqui apresentado para resolver um dos principais obstáculos na sua execução.

A proposta deste trabalho é demonstrar uma técnica para canonizar árvores de expressões booleanas “read-once”(RO). Uma expressão RO é uma expressão onde cada variável aparece apenas uma vez. A forma canônica é, então, utilizada para gerar, em tempo linear, uma assinatura que possibilita p-matching[2].

PROCEDIMENTO

Uma expressão booleana RO pode ser representada como uma árvore, onde as folhas dessa árvore são os literais da expressão. Tomemos como exemplo a seguinte expressão (na qual “+”, “*” e “!” correspondem aos operadores booleanos “OU”, “E” e “NÃO”, respectivamente):

d + e + !(a * b) * c (1)

Uma representação em árvore de (1) tem grau 2, conforme ilustrado na figura 1a. O primeiro passo é **colapsar os nodos operadores** da árvore de modo que não existam dois nodos iguais conectados. Para isso, caso dois operadores idênticos estejam conectados, o de menor profundidade adota os filhos do outro, que é eliminado. Além disso, é necessário **propagar os operadores de negação para as folhas**.

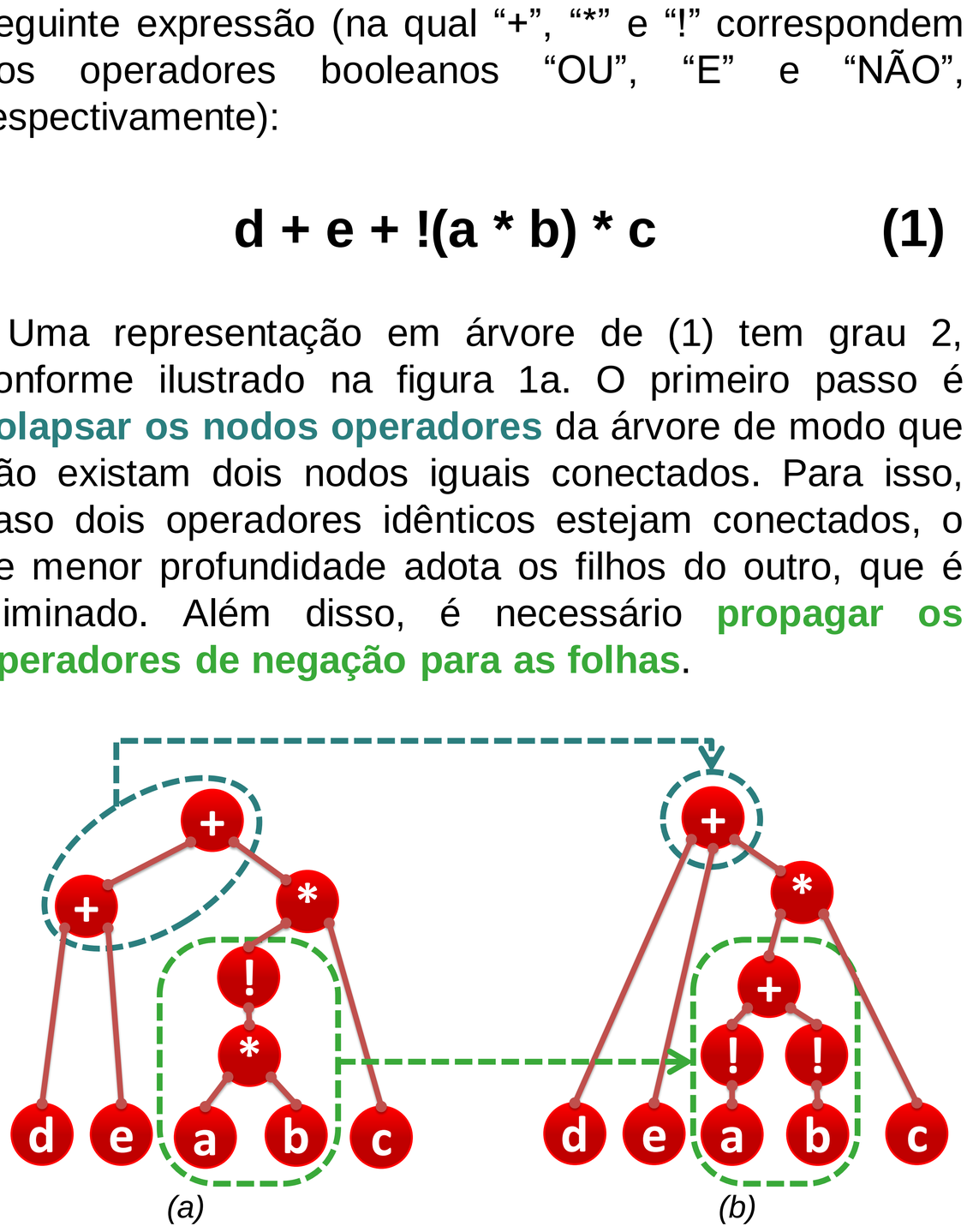

Figura 1- a: árvore correspondente a (1); *b: árvore colapsada com operadores “NÃO” propagados.*

Após essa redução, podemos afirmar que:

- Filhos de nodos “E” são “OU”, “NÃO” ou literais.
- Filhos de nodos “OU” são “E”, “NÃO” ou literais.
- Filhos de nodos “NÃO” são folhas.

Ou seja, dado o operador na raiz e a estrutura da árvore, todo o resto é redundante, visto que dois operadores iguais nunca se conectam e que o operador “NÃO” é o único com somente uma entrada. Portanto, o passo seguinte permite que os nodos, exceto a raiz, sejam substituídos pelas suas profundidades sem perda de informação.

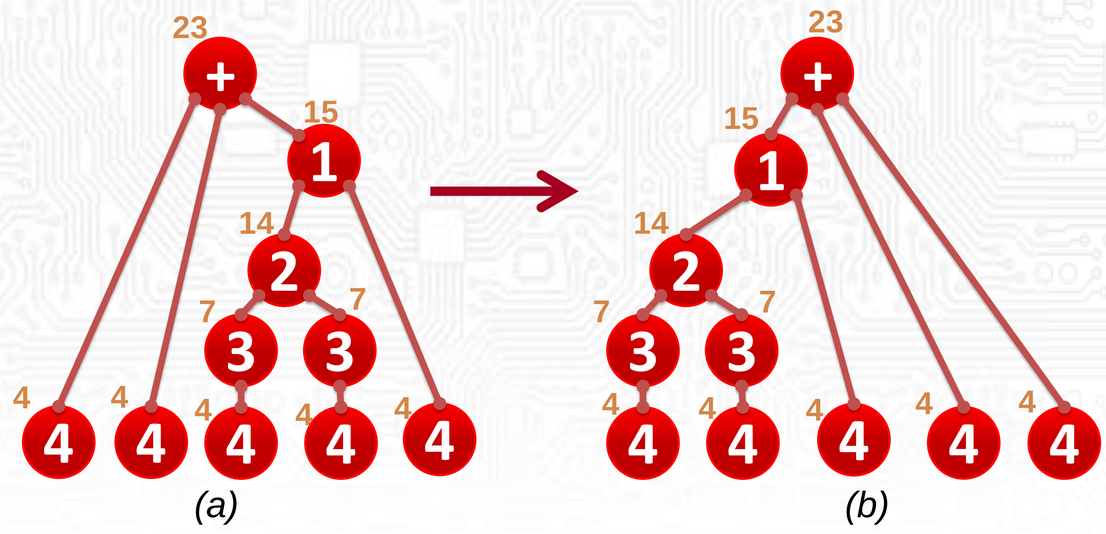

Figura 2 – a: árvore da figura 1ª com os pesos e as profundidades de cada nodo representadas; b: árvore da figura 2a balanceada (ramos mais pesados ficam à esquerda)

Após a substituição dos nodos, a árvore deve ser reordenada de acordo com os **pesos dos ramos**, como ilustrado nas figuras 2a e 2b.

A árvore gerada a partir desse processo pode ser varrida por profundidade, resultando na assinatura (2). Esse método foi inspirado no trabalho proposto por H. I. Scoins[3].

***+123434444* (2)**

Essa é a assinatura correspondente a expressão booleana dada na entrada do procedimento. Se uma expressão equivalente tivesse sido fornecida a assinatura seria a mesma. Portanto, a comparação entre duas assinaturas equivale a testar o p-matching das expressões correspondentes.

RESULTADOS

Foram executados testes sobre conjuntos de expressões com número de literais variando de 4 até 16. Para esses conjuntos, foi amostrado o tempo de execução(em microssegundos) investido na assinatura de cada uma dessas funções, conforme a figura 3.

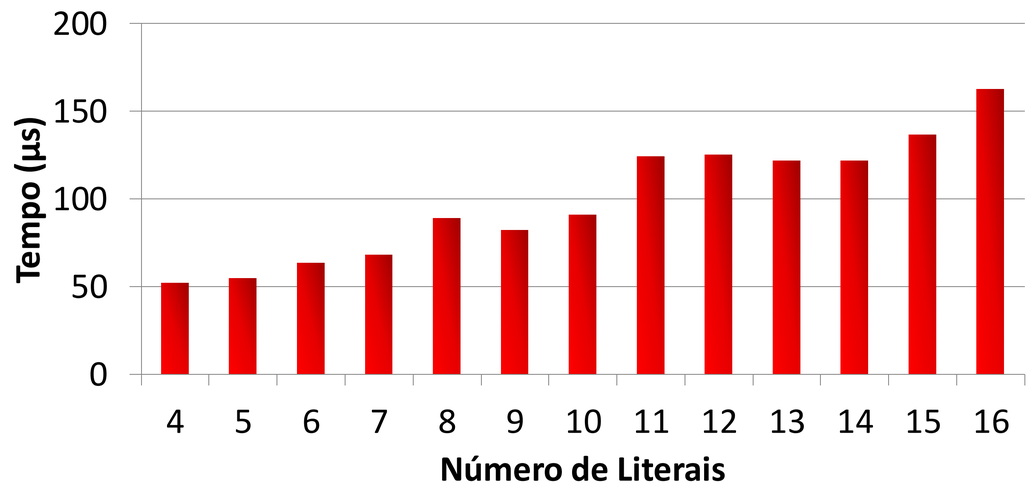

Figura 3: Gráfico de tempos de execução para expressões de 4 a 16 literais

Na execução sobre a biblioteca conhecida como genlib-44-6 [4], o método leva aproximadamente 132 milisegundos para assinar todas as 3503 expressões.

CONCLUSÃO

O método proposto é capaz de gerar uma assinatura para expressões RO em um tempo muito curto e com complexidade linear. Assim sendo, fica evidente que mesmo para conjuntos muito grandes o algoritmo é bastante eficiente.

REFERÊNCIAS

[1] L. Carraro, V. Callegaro, R. P. Ribas e A. I. Reis, *Series-Parallel Switch Network Generator for Read-Once Functions*, 30th South Symposium on Microelectronics (SIM), 2015;

[2] U. Hinsberger e R. Kolla. *Boolean Matching for Large Libraries*, DAC '98 Proceedings of the 35th annual Design Automation Conference, pp 206-211;

[3] H. I. Scoins, *Placing Trees in Lexicographic Order*, Machine Intelligence, vol. 3, pp. 43-60;

[4] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vicentelli e A. Wang, *Technology Mapping in MIS*, Proceedings of the ICCAD, pp. 116-119, November 1987..