Boolean Evaluation with a Pairing and Unpairing Function

Paul Tarau\textsuperscript{1} Brenda Luderman\textsuperscript{2}

University of North Texas\textsuperscript{1}
Texas Instruments Inc.\textsuperscript{2}

SYNASC’2012, Logic and Programming, Saturday, Sept 29, 11:40-12:00
by using pairing functions (bijections $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$) on natural number representations of truth tables, we derive an encoding for Ordered Binary Decision Trees (OBDTs)

boolean evaluation of an OBDT mimics its structural conversion to a natural number through recursive application of a matching pairing function

also: we derive ranking and unranking functions for OBDTs, generalize to arbitrary variable order and multi-terminal OBDTs

literate Haskell program, code at http://logic.csci.unt.edu/tarau/research/2012/hOBDT.hs
“pairing function”: a bijection $J : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$K(J(x, y)) = x,$
$L(J(x, y)) = y$
$J(K(z), L(z)) = z$

examples:
- Cantor’s pairing function: geometrically inspired (100+++ years ago - possibly also known to Cauchy - early 19-th century)
- the Pepis-Kalmar Pairing Function (1938):

$$f(x, y) = 2^x(2y + 1) - 1 \quad (1)$$
a pairing/unpairing function based on boolean operations

type N = Integer

bitunpair :: N→(N,N)
bitpair :: (N,N) → N

bitunpair z = (deflate z, deflate' z)
bitpair (x,y) = inflate x .|.. inflate' y

inflate : abcde-> a0b0c0d0e
inflate': abcde-> 0a0b0c0d0e
inflation/deflation in terms of boolean operations

\[ \text{inflation}, \text{ deflation} : \mathbb{N} \to \mathbb{N} \]

\[ \text{inflation} \ 0 = 0 \]
\[ \text{inflation} \ n = (\text{twice} \ . \ \text{twice} \ . \ \text{inflation} \ . \ \text{half}) \ n \ || \ . \ \text{firstBit} \ n \]

\[ \text{deflation} \ 0 = 0 \]
\[ \text{deflation} \ n = (\text{twice} \ . \ \text{deflation} \ . \ \text{half} \ . \ \text{half}) \ n \ || \ . \ \text{firstBit} \ n \]

\[ \text{deflation}' = \text{half} \ . \ \text{deflation} \ . \ \text{twice} \]
\[ \text{inflation}' = \text{twice} \ . \ \text{inflation} \]

\[ \text{half} \ n = \text{shiftR} \ n \ 1 : : \ N \]
\[ \text{twice} \ n = \text{shiftL} \ n \ 1 : : \ N \]
\[ \text{firstBit} \ n = n \ .&. \ 1 : : \ N \]
bitpair/bitunpair: an example

the transformation of the bitlists – with bitstrings aligned:

*BP> bitunpair 2012
(62, 26)

-- 2012: [0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1]
-- 62: [0, 1, 1, 1, 1, 1]
-- 26: [0, 1, 0, 1, 1]

Note that we represent numbers with bits in reverse order.
Also, some simple algebraic properties:

bitpair (x, 0) = inflate x
bitpair (0, x) = 2 * (inflate x)
bitpair (x, x) = 3 * (inflate x)
Given that unpairing functions are bijections from $\mathbb{N} \to \mathbb{N} \times \mathbb{N}$ they will progressively cover all points having natural number coordinates in the plan.

Pairing can be seen as a function $z=f(x,y)$, thus it can be displayed as a 3D surface.

Recursive application – the unpairing tree can be represented as a DAG – by merging shared nodes.
Figure: 2D curve connecting values of bitunpair \( n \) for \( n \in [0..2^{10} - 1] \)
Figure: Graph obtained by recursive application of \texttt{bitunpair} for 2012
Unpairing Trees: seen as OBDTs

data BT = O | I | D BT BT deriving (Eq, Ord, Read, Show)

unfold_bt :: (N,N) → BT
unfold_bt (n,tt) = if tt < 2^2^n
    then unfold_with bitunpair n tt
    else undefined where
        unfold_with _ n 0 | n<1 = O
        unfold_with _ n 1 | n<1 = I
        unfold_with f n tt =
            D (unfold_with f k tt1) (unfold_with f k tt2) where
            k=n-1
            (tt1,tt2)=f tt
Folding back Trees to Natural Numbers

\[
\text{fold} \_\text{bt} :: \text{BT} \to (\text{N}, \text{N}) \\
\text{fold} \_\text{bt} \text{ bt} = (\text{bdepth bt}, \text{fold} \_\text{with bitpair bt}) \text{ where} \\
\quad \text{fold} \_\text{with f O} = 0 \\
\quad \text{fold} \_\text{with f I} = 1 \\
\quad \text{fold} \_\text{with f (D l r)} = f (\text{fold} \_\text{with f l}, \text{fold} \_\text{with f r})
\]

\[
\text{bdepth O} = 0 \\
\text{bdepth I} = 0 \\
\text{bdepth (D x _)} = 1 + (\text{bdepth x})
\]

This is a purely structural operation - no boolean evaluation involved!

\*BP\> unfold\_bt (3,42)
\D (D (D O O) (D O O)) (D (D I I) (D I O))
\*BP\> fold\_bt it
(3,42)
### Truth tables as natural numbers

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| x | y | z |   |   | f | x | y | z |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 |   |   | → | 0 |
| 1 | 0 | 1 |   |   | → | 1 |
| 2 | 0 | 1 |   |   | → | 0 |
| 3 | 0 | 1 |   |   | → | 1 |
| 4 | 1 | 0 |   |   | → | 0 |
| 5 | 1 | 0 |   |   | → | 1 |
| 6 | 1 | 1 |   |   | → | 1 |
| 7 | 1 | 1 |   |   | → | 0 |

::

\{1, 3, 5, 6\} :: 106 = 2^1 + 2^3 + 2^5 + 2^6 = 2 + 8 + 32 + 64

01010110 (right to left)
Proposition

Let \( v_k \) be a variable for \( 0 \leq k < n \) where \( n \) is the number of distinct variables in a boolean expression. Then column \( k \) in the matrix representation of the inputs in the truth table represents, as a bitstring, the natural number:

\[
v_k = \frac{2^n - 1}{2^k + 1}
\]  

(2)

For instance, if \( n = 2 \), the formula computes \( v_0 = 3 = [0, 0, 1, 1] \) and \( v_1 = 5 = [0, 1, 0, 1] \).
The function $v_n$, working with arbitrary length bitstrings are used to evaluate the $[0..n-1]$ projection variables $v_k$ representing encodings of columns of a truth table, while $v_m$ maps the constant 1 to the bitstring of length $2^n$, 111...1:

\[
v_n \colon \mathbb{N} \rightarrow \mathbb{N} \\
v_n 1 0 = 1 \\
v_n n q \mid q = n-1 = \text{bitpair} (v_n n 0,0) \\
v_n n q \mid q \geq 0 \land q < n' = \text{bitpair} (q',q') \text{ where} \\
\quad n' = n-1 \\
\quad q' = v_n n' q \\
\]

\[
v_m \colon \mathbb{N} \rightarrow \mathbb{N} \\
v_m n = v_n (n+1) 0
\]
an ordered binary decision diagram (OBDT) is a rooted ordered binary tree obtained from a boolean function, by assigning its variables, one at a time, to 0 (left branch) and 1 (right branch).

deriving a OBDT of a boolean function \( f \): repeated Shannon expansion

\[
f(x) = (\bar{x} \land f[x \leftarrow 0]) \lor (x \land f[x \leftarrow 1])
\] (3)

with a more familiar notation:

\[
f(x) = \text{if } x \text{ then } f[x \leftarrow 1] \text{ else } f[x \leftarrow 0]
\] (4)
Boolean Evaluation of OBDTs

- OBDTs ⇒ ROBDDs by sharing nodes + dropping identical branches
- \texttt{fold_obdt} might give a different result as it computes different pairing operations!
- however, we obtain a truth table if we evaluate the OBDT tree as a boolean function
- can we relate this to the original truth table from which we unfolded the OBDT?

Paul Tarau, Brenda Luderman
University of North Texas\textsuperscript{1}, Texas Instruments Inc.\textsuperscript{2}
evaluating an OBDT with given variable order $vs$

eval_obdt_with :: [N] → BT → N

eval_obdt_with vs bt =
    eval_with_mask (vm n) (map (vn n) vs) bt where
    n = genericLength vs

eval_with_mask m _ O = 0

eval_with_mask m _ I = m

eval_with_mask m (v:vs) (D l r) =
    ite_ v (eval_with_mask m vs l) (eval_with_mask m vs r)

ite_ x t e = ((t `xor` e) .&. x) `xor` e
The Equivalence of boolean evaluation and recursive pairing

SURPRISINGLY, it turns out that:

- boolean evaluation `eval_obdt` faithfully emulates `fold_obdt`
- and it also works on reduced OBDTs, ROBDDs, BDDs as they represent the same boolean formula

*BP> unfold_bt (3,42)
D (D (D O O) (D O O)) (D (D I I) (D I O))
*BP> eval_obdt it
42

Paul Tarau, Brenda Luderman
University of North Texas\textsuperscript{1}, Texas Instruments Inc.\textsuperscript{2}

Boolean Evaluation with a Pairing and Unpairing Function
The Equivalence

Proposition

The complete binary tree of depth \( n \), obtained by recursive applications of \texttt{bitunpair} on a truth table computes an (unreduced) OBDT, that, when evaluated (reduced or not) returns the truth table, i.e.

\[
\text{fold}_\text{obdt} \circ \text{unfold}_\text{obdt} \equiv \text{id} \quad (5)
\]

\[
\text{eval}_\text{obdt} \circ \text{unfold}_\text{obdt} \equiv \text{id} \quad (6)
\]
Ranking/unranking: bijection to/from $\mathbb{N}$

- one more step is needed to extend the mapping between $OBDTs$ with $\mathbb{N}$ variables to a bijective mapping from/to $\mathbb{N}$:
  - we will have to “shift toward infinity” the starting point of each new block of OBDTs in $\mathbb{N}$ as OBDTs of larger and larger sizes are enumerated
- we need to know by how much - so we compute the sum of the counts of boolean functions with up to $\mathbb{N}$ variables.
Ranking/unranking of OBDTs

\[
\begin{align*}
\text{bsum} &:: \mathbb{N} \rightarrow \mathbb{N} \\
\text{bsum} \; 0 & = 0 \\
\text{bsum} \; n \mid n > 0 & = \text{bsum1} \; (n-1) \text{ where} \\
\text{bsum1} \; 0 & = 2 \\
\text{bsum1} \; n \mid n > 0 & = \text{bsum1} \; (n-1) + 2^{2^n} \\
\end{align*}
\]

*BP*> genericTake 7 bsums
[0, 2, 6, 22, 278, 65814, 4295033110]

**A060803** in the Online Encyclopedia of Integer Sequences

*BP*> nat2obdt 42
D (D (D O I) (D I O)) (D (D O O) (D O O))

*BP*> obdt2nat it
42
Generalizations

Given a permutation of $n$ variables represented as natural numbers in $[0..n-1]$ and a truth table $tt \in [0..2^{2^n} - 1]$ we can define:

\[
\text{to_obdt vs } tt \mid 0 \leq tt \&\& tt \leq m = \text{to_obdt}_\text{mn vs } tt \text{ m n where } \\
\quad n=\text{genericLength vs } \\
\quad m=\text{vm n}
\]

\[
\text{to_obdt}_\text{mn} [\] 0 _ _ = 0 \\
\text{to_obdt}_\text{mn} [\] _ _ _ = I \\
\text{to_obdt}_\text{mn} (v:vs) tt \text{ m n} = D l r \text{ where } \\
\quad \text{cond} = vn n v \\
\quad f0 = (m \ 'xor' \ \text{cond}) \ .&. tt \\
\quad f1 = \text{cond} \ .&. tt \\
\quad l = \text{to_obdt}_\text{mn vs } f1 \text{ m n} \\
\quad r = \text{to_obdt}_\text{mn vs } f0 \text{ m n}
\]
Applications

- possible applications to (RO)BDDs: circuit synthesis/verification
- BDD minimization using our generalization to arbitrary variable order
- combinatorial enumeration and random generation of circuits
- succinct data representations derived from our OBDT encodings
- an interesting “mutation”: use integers/bitstrings as genotypes, OBDTs as phenotypes in Genetic Algorithms
Conclusion

- **NEW:** the connection of pairing/unpairing functions and boolean evaluation of OBDTs
- synergy between concepts borrowed from *foundation of mathematics, combinatorics, boolean logic, circuits*
- Haskell as sandbox for experimental mathematics: type inference helps clarifying complex dependencies between concepts quite nicely - moving to a functional subset of Mathematica, after that, is routine.