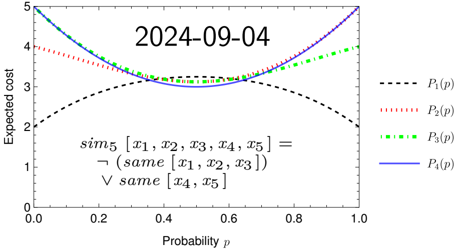
Level-p-complexity of Boolean Functions

Using thinning, memoization, and polynomials

Patrik Jansson[1]

Functional Programming unit, Chalmers University of Technology



DSL → δσλ
DSLsofMath

[1] Joint with Julia Jansson, Math@Chalmers. Paper: doi:10.1017/S0956796823000102, code@GitHub.

Explain the setting: BoFun, DecTree, cost, expected cost

- Start with some boolean function $f : \mathbb{B}^n \to \mathbb{B}$

Examples: sim_5, maj_3, maj_3^2

Explain the setting: BoFun, DecTree, cost, expected cost

- Start with some boolean function $f : \mathbb{B}^n \to \mathbb{B}$
- This function can be evaluated on an n-bit bool vector (word) in different ways.

Examples: sim_5, maj_3, maj_3^2

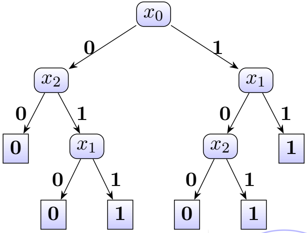
Explain the setting: BoFun, DecTree, cost, expected cost

- Start with some boolean function $f : \mathbb{B}^n \to \mathbb{B}$
- This function can be evaluated on an n-bit bool vector (word) in different ways.
- We capture an evaluation order using a $\mathit{DecTree}$:
  - query one bit (index) at a time
  - until the answer if clear (the "remaining function" is constant)

$$
\begin{array}{l}
\textbf{type}\ \mathit{Index} = \mathbb{N} \\
\textbf{data}\ \mathit{DecTree}\ \textbf{where} \\
\quad \mathit{Pick} :: \mathit{Index} \to \mathit{DecTree} \to \mathit{DecTree} \to \mathit{DecTree} \\
\quad \mathit{Res}\ \ :: \mathbb{B} \to \mathit{DecTree}
\end{array}
$$

Examples: sim_5, maj_3, maj_3^2

Example: a $\mathit{DecTree}$ for maj_3:



Explain the setting: BoFun, DecTree, cost, expected cost

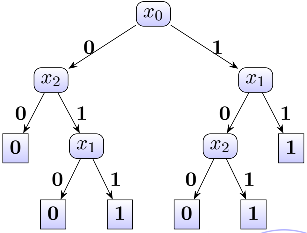
- Start with some boolean function $f : \mathbb{B}^n \to \mathbb{B}$
- This function can be evaluated on an n-bit bool vector (word) in different ways.
- We capture an evaluation order using a $\mathit{DecTree}$:
  - query one bit (index) at a time
  - until the answer if clear (the "remaining function" is constant)

$$
\begin{array}{l}
\mathbf{type}\ \mathit{Index} = \mathbb{N} \\
\mathbf{data}\ \mathit{DecTree}\ \mathbf{where} \\
\quad \mathit{Pick} :: \mathit{Index} \to \mathit{DecTree} \to \mathit{DecTree} \to \mathit{DecTree} \\
\quad \mathit{Res}\ \ :: \mathbb{B} \to \mathit{DecTree}
\end{array}
$$

- We compute the cost of a dectree (for our boolean function) $\mathit{cost} : \mathit{DecTree} \to \mathbb{B}^n \to \mathbb{N}$
  - for every n-bit word (there are 2^n such words)

Examples: sim_5, maj_3, maj_3^2

Example: a $\mathit{DecTree}$ for maj_3:



Explain the setting: BoFun, DecTree, cost, expected cost

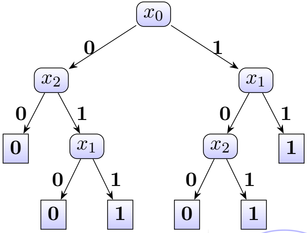
- Start with some boolean function $f : \mathbb{B}^n \to \mathbb{B}$
- This function can be evaluated on an n-bit bool vector (word) in different ways.
- We capture an evaluation order using a $DecTree$:
  - query one bit (index) at a time
  - until the answer if clear (the "remaining function" is constant)

  $$\begin{aligned}&\mathbf{type}\ Index = \mathbb{N}\\&\mathbf{data}\ DecTree\ \mathbf{where}\\&\quad Pick :: Index \to DecTree \to DecTree \to DecTree\\&\quad Res\ \ :: \mathbb{B} \to DecTree\end{aligned}$$

- We compute the cost of a dectree (for our boolean function) $cost : DecTree \to \mathbb{B}^n \to \mathbb{N}$
  - for every n-bit word (there are 2^n such words)
- Then compute the *expected* cost over all 2^n words $\to$ a polynomial in the probability a bit is 1

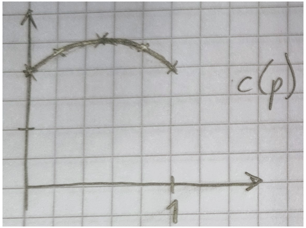
Examples: sim_5, maj_3, maj_3^2

Example: a $DecTree$ for maj_3:



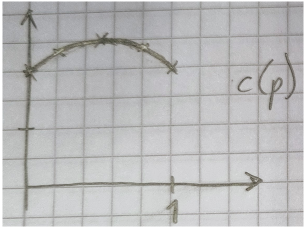
Examples of expected cost and level-p-complexity: $D_f(p)$

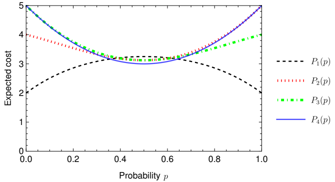
- Example 1: maj_3, $expCost\ t = 2 + 2p(1-p)$ (for all t)



Examples of expected cost and level-p-complexity: $D_f(p)$

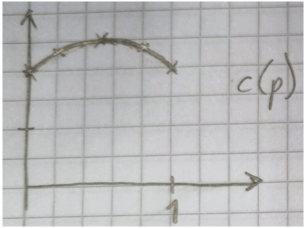
- Example 1: maj_3, $expCost\ t = 2 + 2p(1-p)$ (for all t)
- Example 2: sim_5,
  - $P_1\ p = expCost\ t_1 = 2 + 6p - 10p^2 + 8p^3 - 4p^4$
  - $\ldots$
  - $P_4\ p = expCost\ t_4 = 5 - 8p + 8p^2$

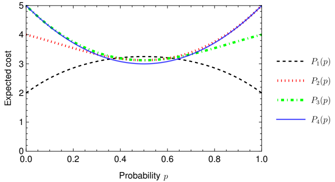




Examples of expected cost and level-p-complexity: $D_f(p)$

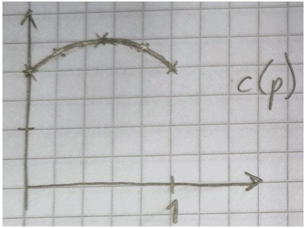
- Example 1: maj_3, $expCost\ t = 2 + 2p(1-p)$ (for all t)
- Example 2: sim_5,
  - $P_1\ p = expCost\ t_1 = 2 + 6p - 10p^2 + 8p^3 - 4p^4$
  - $\ldots$
  - $P_4\ p = expCost\ t_4 = 5 - 8p + 8p^2$
- Finally we compute the *minimum* over all decision trees. This gives a *piecewise* polynomial function. (Approximated by a set of pairwise intersecting polynomials.)

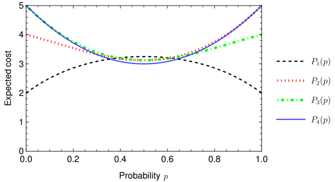




Examples of expected cost and level-p-complexity: $D_f(p)$

- Example 1: maj_3, $expCost\ t = 2 + 2p(1-p)$ (for all t)
- Example 2: sim_5,
  - $P_1\ p = expCost\ t_1 = 2 + 6p - 10p^2 + 8p^3 - 4p^4$
  - ...
  - $P_4\ p = expCost\ t_4 = 5 - 8p + 8p^2$
- Finally we compute the *minimum* over all decision trees. This gives a *piecewise* polynomial function. (Approximated by a set of pairwise intersecting polynomials.)
- This minimum is the level-p-complexity $D_f(p)$ of f.
- Example 1: maj_3, $D_f(p) = 2 + 2p(1-p)$
- Example 2: sim_5, $D_f(p) =$ three pieces, from P_1, P_4, P_1.





Core algorithm for computing the level-p-complexity of f

"generate all decision trees and pick the best one(s)"

- Informally: to generate all the decision trees:
  - for every position i in $\{0 \mathinner{..} n-1\}$,
    - compute the two "immediate subfunctions" (when bit i is 0 or 1).
    - generate trees for both cases
    - combine them in all ways (cross product)

Core algorithm for computing the level-p-complexity of f

"generate all decision trees and pick the best one(s)"

- Informally: to generate all the decision trees:
  - for every position i in $\{0 \,..\, n-1\}$,
    - compute the two "immediate subfunctions" (when bit i is 0 or 1).
    - generate trees for both cases
    - combine them in all ways (cross product)
- ...or in (prettified) Haskell (using type classes $BoFun$, $TreeAlg$ explained in the paper)

$$
\begin{aligned}
&genAlg_n :: (BoFun\ bf, TreeAlg\ ta) \Rightarrow bf \ \rightarrow\ Set\ ta\\
&genAlg_n \quad f \mid Just\ b \leftarrow isConst\ f\\
&\qquad\qquad\qquad = \{res\ b\}\\
&genAlg_{n+1}\ f \ \ = \{pic\ i\ t_0\ t_1 \mid i \leftarrow \{0 \,..\, n-1\}, t_0 \leftarrow genAlg_n\ f^i_{\mathbf{0}}, t_1 \leftarrow genAlg_n\ f^i_{\mathbf{1}}\}
\end{aligned}
$$

Core algorithm for computing the level-p-complexity of f

"generate all decision trees and pick the best one(s)"

- Informally: to generate all the decision trees:
  - for every position i in $\{0 \mathinner{.\,.} n-1\}$,
    - compute the two "immediate subfunctions" (when bit i is 0 or 1).
    - generate trees for both cases
    - combine them in all ways (cross product)
- ...or in (prettified) Haskell (using type classes BoFun, $\mathit{TreeAlg}$ explained in the paper)

$$
\begin{aligned}
&\mathit{genAlg}_n :: (\mathit{BoFun}\ \mathit{bf}, \mathit{TreeAlg}\ \mathit{ta}) \Rightarrow \mathit{bf} \rightarrow \mathit{Set}\ \mathit{ta}\\
&\mathit{genAlg}_n \quad f \mid \mathit{Just}\ b \leftarrow \mathit{isConst}\ f\\
&\qquad\qquad\quad = \{\mathit{res}\ b\}\\
&\mathit{genAlg}_{n+1}\ f \quad = \{\mathit{pic}\ i\ t_0\ t_1 \mid i \leftarrow \{0 \mathinner{.\,.} n-1\}, t_0 \leftarrow \mathit{genAlg}_n\ f^i_{\mathbf{0}}, t_1 \leftarrow \mathit{genAlg}_n\ f^i_{\mathbf{1}}\}
\end{aligned}
$$

- How many $\mathit{DecTree}$s can there be?

Core algorithm for computing the level-p-complexity of f

"generate all decision trees and pick the best one(s)"

- Informally: to generate all the decision trees:
  - for every position i in $\{0 \mathinner{..} n-1\}$,
    - compute the two "immediate subfunctions" (when bit i is 0 or 1).
    - generate trees for both cases
    - combine them in all ways (cross product)
- ...or in (prettified) Haskell (using type classes BoFun, $\mathit{TreeAlg}$ explained in the paper)

$$\begin{aligned}
&\mathit{genAlg}_n :: (\mathit{BoFun}\ \mathit{bf}, \mathit{TreeAlg}\ \mathit{ta}) \Rightarrow \mathit{bf} \to \mathit{Set}\ \mathit{ta}\\
&\mathit{genAlg}_n \quad f \mid \mathit{Just}\ b \leftarrow \mathit{isConst}\ f\\
&\qquad\qquad\quad = \{\mathit{res}\ b\}\\
&\mathit{genAlg}_{n+1}\ f \quad = \{\mathit{pic}\ i\ t_0\ t_1 \mid i \leftarrow \{0 \mathinner{..} n-1\}, t_0 \leftarrow \mathit{genAlg}_n\ f^i_{\mathbf{0}}, t_1 \leftarrow \mathit{genAlg}_n\ f^i_{\mathbf{1}}\}
\end{aligned}$$

- How many $\mathit{DecTree}$s can there be? In the worst case we get $T_{n+1} = n \cdot T_n^2$
- An estimate of the number of different $\mathit{DecTree}$s for a function of n bits is $2^{2^{n-1}}$ or simply **FAR TOO MANY**.

Fusion and thinning (towards efficient / tractable algorithm)

- We aim to filter (minimize) at the end - can we take advantage of this?
- We compare two dectrees by computing their polynomials and comparing the polynomials.

Fusion and thinning (towards efficient / tractable algorithm)

- We aim to filter (minimize) at the end - can we take advantage of this?
- We compare two dectrees by computing their polynomials and comparing the polynomials.
- Yes 1: We can push the computation of the polynomials down all the way, so that we don't need to compute any dectrees at all.
  This is already a good gain, but still doubly exponential.

Fusion and thinning (towards efficient / tractable algorithm)

- We aim to filter (minimize) at the end - can we take advantage of this?
- We compare two dectrees by computing their polynomials and comparing the polynomials.
- Yes 1: We can push the computation of the polynomials down all the way, so that we don't need to compute any dectrees at all.
  This is already a good gain, but still doubly exponential.
- Next, can we do some "thinning": push some of the filtering down?

Fusion and thinning (towards efficient / tractable algorithm)

- We aim to filter (minimize) at the end - can we take advantage of this?
- We compare two dectrees by computing their polynomials and comparing the polynomials.
- Yes 1: We can push the computation of the polynomials down all the way, so that we don't need to compute any dectrees at all.
  This is already a good gain, but still doubly exponential.
- Next, can we do some "thinning": push some of the filtering down?
- Yes 2: If we are careful, the comparison order is monotone in the operation used to combine smaller polynomials into larger ones.
  Thus, we can throw out many polynomials early, because they have no chance of contributing to the final result.

Fusion and thinning (towards efficient / tractable algorithm)

- We aim to filter (minimize) at the end - can we take advantage of this?
- We compare two dectrees by computing their polynomials and comparing the polynomials.
- Yes 1: We can push the computation of the polynomials down all the way, so that we don't need to compute any dectrees at all.
  This is already a good gain, but still doubly exponential.
- Next, can we do some "thinning": push some of the filtering down?
- Yes 2: If we are careful, the comparison order is monotone in the operation used to combine smaller polynomials into larger ones.
  Thus, we can throw out many polynomials early, because they have no chance of contributing to the final result.
- Success? Not yet.

Reuse of intermediate results (memoization)

- We can now easily compute up to n=5 but n=9 is still beyond reach.
- In a motivating example (iterated majority on $3 \cdot 3 = 9$ bits) we now get just 1 polynomial at the end, but we still have $2 \cdot n$ immediate subfunctions (recursive calls) and around **100 million** recursive calls in total.

Reuse of intermediate results (memoization)

- We can now easily compute up to n=5 but n=9 is still beyond reach.
- In a motivating example (iterated majority on $3 \cdot 3 = 9$ bits) we now get just 1 polynomial at the end, but we still have $2 \cdot n$ immediate subfunctions (recursive calls) and around **100 million** recursive calls in total.
- But perhaps some of those recursive calls are for the same function - if so we could reuse the result (use memoization).
- This turns out to be a great idea - for our running example the number of subfunctions (unique recursive calls) is only 215.

Reuse of intermediate results (memoization)

- We can now easily compute up to n=5 but n=9 is still beyond reach.
- In a motivating example (iterated majority on $3 \cdot 3 = 9$ bits) we now get just 1 polynomial at the end, but we still have $2 \cdot n$ immediate subfunctions (recursive calls) and around **100 million** recursive calls in total.
- But perhaps some of those recursive calls are for the same function - if so we could reuse the result (use memoization).
- This turns out to be a great idea - for our running example the number of subfunctions (unique recursive calls) is only 215.
- Finally we can compute the Level-p-complexity for maj_3^2 in around a few seconds. (Still 100 million recursive calls, but almost all are just a lookup in a memo-table.)

What I glossed over + future work

- The actual implementation of memoization is a bit tricky - we need to compare functions for equality (efficiently). But Binary Decision Diagrams come to our rescue (see paper).

What I glossed over + future work

- The actual implementation of memoization is a bit tricky - we need to compare functions for equality (efficiently). But Binary Decision Diagrams come to our rescue (see paper).
- To implement comparison of polynomials we need to know (exactly) where they intersect. In effect this requires computing with Algebraic numbers (symbolic computations with polynomials, root counting, interval halving, etc). Details in the paper and the code.

What I glossed over + future work

- The actual implementation of memoization is a bit tricky - we need to compare functions for equality (efficiently). But Binary Decision Diagrams come to our rescue (see paper).
- To implement comparison of polynomials we need to know (exactly) where they intersect. In effect this requires computing with Algebraic numbers (symbolic computations with polynomials, root counting, interval halving, etc). Details in the paper and the code.
- To go further, beyond $n = 9$, we have used more properties of special classes of boolean functions (iterated threshold functions) to reduce the computational cost even further. (In the repo we have reached $n = 27$ - thanks to help from Christian Sattler.)

Conclusions

If you run into an intractable (but algebraically nice) problem - some algorithmic design techniques can help *a lot*.

- **Thinning**:
  - reduce the size of the candidate set as early as possible to avoid exponential blowup.

Conclusions

If you run into an intractable (but algebraically nice) problem - some algorithmic design techniques can help *a lot*.

- **Thinning**:
  - reduce the size of the candidate set as early as possible to avoid exponential blowup.
- **Memoization**:
  - If your recursive call structure ends up recomputing a lot - store the results in a table.

Conclusions

If you run into an intractable (but algebraically nice) problem - some algorithmic design techniques can help *a lot*.

- **Thinning**:
  - reduce the size of the candidate set as early as possible to avoid exponential blowup.
- **Memoization**:
  - If your recursive call structure ends up recomputing a lot - store the results in a table.
- Representing functions
  - **Binary Decision Diagrams (BDDs)**: If you need to work seriously with boolean functions, you should represent them as BDDs. (Roughly $DecTree$s with sharing)

Conclusions

If you run into an intractable (but algebraically nice) problem - some algorithmic design techniques can help *a lot*.

- **Thinning**:
  - reduce the size of the candidate set as early as possible to avoid exponential blowup.
- **Memoization**:
  - If your recursive call structure ends up recomputing a lot - store the results in a table.
- Representing functions
  - **Binary Decision Diagrams (BDDs)**: If you need to work seriously with boolean functions, you should represent them as BDDs. (Roughly $DecTree$s with sharing)
  - **Polynomials**: if you compute with (numeric) functions of a special structure (like polynomials), symbolic computation helps a lot.

Conclusions / Questions?

If you run into an intractable (but algebraically nice) problem - some algorithmic design techniques can help *a lot*.

- **Thinning**:
  - reduce the size of the candidate set as early as possible to avoid exponential blowup.
- **Memoization**:
  - If your recursive call structure ends up recomputing a lot - store the results in a table.
- Representing functions
  - **Binary Decision Diagrams (BDDs)**: If you need to work seriously with boolean functions, you should represent them as BDDs. (Roughly $DecTree$s with sharing)
  - **Polynomials**: if you compute with (numeric) functions of a special structure (like polynomials), symbolic computation helps a lot.

Paper: doi:10.1017/S0956796823000102, code: github:juliajansson/BoFunComplexity.

Joint work with Julia Jansson (PhD student @ Math.chalmers.se)

Motivating example: two-level iterated majority

Our running example is a simple case of two-level majority ($maj_3^2 : \mathbb{B}^9 \to \mathbb{B}$).

$$\underbrace{\underbrace{x_{(1,1)}, x_{(1,2)}, x_{(1,3)}}_{m_1 = maj_3\ (\ldots)}, \underbrace{x_{(2,1)}, x_{(2,2)}, x_{(2,3)}}_{m_2 = maj_3\ (\ldots)}, \underbrace{x_{(3,1)}, x_{(3,2)}, x_{(3,3)}}_{m_3 = maj_3\ (\ldots)}}_{maj_3(m_1, m_2, m_3)}$$

Motivating example: two-level iterated majority

Our running example is a simple case of two-level majority ($maj_3^2 : \mathbb{B}^9 \to \mathbb{B}$).

$$\underbrace{\underbrace{x_{(1,1)}, x_{(1,2)}, x_{(1,3)}}_{m_1 = maj_3\ (\ldots)}, \underbrace{x_{(2,1)}, x_{(2,2)}, x_{(2,3)}}_{m_2 = maj_3\ (\ldots)}, \underbrace{x_{(3,1)}, x_{(3,2)}, x_{(3,3)}}_{m_3 = maj_3\ (\ldots)}}_{maj_3(m_1, m_2, m_3)}$$

Example 1: Five $\mathbf{0}$ votes, four $\mathbf{1}$ votes, even distribution:

$$\underbrace{\underbrace{\mathbf{0}, \mathbf{1}, \mathbf{0}}_{m_1 = \mathbf{0}}, \underbrace{\mathbf{1}, \mathbf{0}, \mathbf{1}}_{m_2 = \mathbf{1}}, \underbrace{\mathbf{0}, \mathbf{1}, \mathbf{0}}_{m_3 = \mathbf{0}}}_{maj_3 = \mathbf{0}}$$

Motivating example: two-level iterated majority

Our running example is a simple case of two-level majority ($maj_3^2 : \mathbb{B}^9 \to \mathbb{B}$).

$$\underbrace{\underbrace{x_{(1,1)}, x_{(1,2)}, x_{(1,3)}}_{m_1 = maj_3\ (\ldots)}, \underbrace{x_{(2,1)}, x_{(2,2)}, x_{(2,3)}}_{m_2 = maj_3\ (\ldots)}, \underbrace{x_{(3,1)}, x_{(3,2)}, x_{(3,3)}}_{m_3 = maj_3\ (\ldots)}}_{maj_3(m_1, m_2, m_3)}$$

Example 1: Five $\mathbf{0}$ votes, four $\mathbf{1}$ votes, even distribution:

$$\underbrace{\underbrace{\mathbf{0}, \mathbf{1}, \mathbf{0}}_{m_1 = \mathbf{0}}, \underbrace{\mathbf{1}, \mathbf{0}, \mathbf{1}}_{m_2 = \mathbf{1}}, \underbrace{\mathbf{0}, \mathbf{1}, \mathbf{0}}_{m_3 = \mathbf{0}}}_{maj_3 = \mathbf{0}}$$

Example 2: Five $\mathbf{0}$ votes, four $\mathbf{1}$ votes, regrouped (perhaps through gerrymandering):

$$\underbrace{\underbrace{\mathbf{1}, \mathbf{1}, \mathbf{0}}_{m_1 = \mathbf{1}}, \underbrace{\mathbf{1}, \mathbf{0}, \mathbf{1}}_{m_2 = \mathbf{1}}, \underbrace{\mathbf{0}, \mathbf{0}, \mathbf{0}}_{m_3 = \mathbf{0}}}_{maj_3 = \mathbf{1}}$$

Result for maj_3^2 and sim_5

$$
\begin{aligned}
&ps_9 = genAlgThinMemo\ 9\ \ maj_3^2 :: Set\ (Poly\ \mathbb{Q})\\
&check_9 = \ ps_9 == fromList\ [\ P\ [4, 4, 6, 9, -61, 23, 67, -64, 16]]
\end{aligned}
$$

$$
\begin{aligned}
&ps5\ = genAlgThinMemo\ 5\ sim_5\ :: Set\ (Poly\ \mathbb{Q})\\
&check5\ = ps5\ == fromList\ [P\ [2, 6, -10, 8, -4], P\ [4, -2, -3, 8, -2],\\
&\qquad\qquad\qquad\qquad\qquad P\ [5, -8, 9, 0, -2],\ \ P\ [5, -8, 8]]
\end{aligned}
$$

Subfunctions illustrated

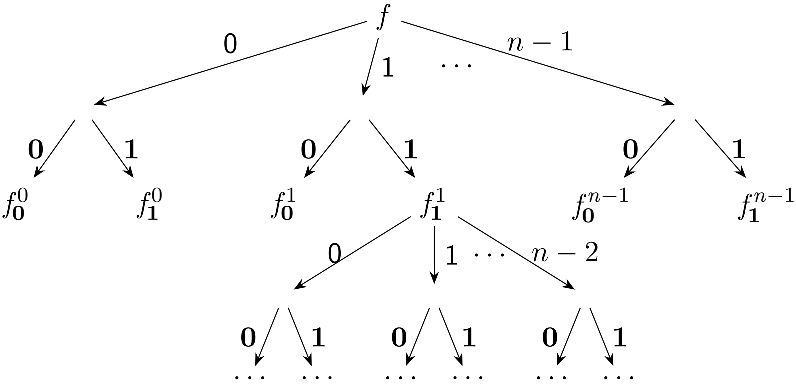


Figure: The tree of subfunctions of a Boolean function f. For brevity $setBit\ i\ b\ f$ is denoted f^i_b. This tree structure is also the call-graph for our generation of decision trees. Note that this tree structure is related to, but not the same as, the decision trees.

A class of Boolean functions

We abstract from the actual type for Boolean functions to a type class:

$$
\begin{array}{l}
\mathbf{class}\ \mathit{BoFun}\ \mathit{bf}\ \mathbf{where} \\
\quad \mathit{isConst} :: \mathit{bf} \to \mathit{Maybe}\ \mathbb{B} \\
\quad \mathit{setBit} \ \ :: \mathit{Index} \to \mathbb{B} \to \mathit{bf} \to \mathit{bf}
\end{array}
$$

We only use one instance: Binary Decision Diagrams (BDDs) which allows good sharing and fast operations.

A class of decision tree algebras

Similarly, for decision trees, we use a class of tree algebras:

$$
\begin{array}{l}
\textbf{class}\ \mathit{TreeAlg}\ a\ \textbf{where} \\
\quad \mathit{res} :: \mathbb{B} \to a \\
\quad \mathit{pic} :: \mathit{Index} \to a \to a \to a
\end{array}
$$

$$
\begin{array}{l}
\mathit{fold}_{DT} :: \mathit{TreeAlg}\ a \Rightarrow \mathit{DecTree} \to a \\
\mathit{fold}_{DT}\ (\mathit{Res}\ b) \qquad\quad = \mathit{res}\ b \\
\mathit{fold}_{DT}\ (\mathit{Pick}\ i\ t_0\ t_1) = \mathit{pic}\ i\ (\mathit{fold}_{DT}\ t_0)\ (\mathit{fold}_{DT}\ t_1)
\end{array}
$$

$$
\begin{array}{l}
\mathit{ex1} :: \mathit{TreeAlg}\ a \Rightarrow a \\
\mathit{ex1} = \mathit{pic}\ 0\ (\mathit{pic}\ 2\ (\mathit{res}\ \mathbf{0})\ (\mathit{pic}\ 1\ (\mathit{res}\ \mathbf{0})\ (\mathit{res}\ \mathbf{1}))) \\
\qquad\qquad\quad (\mathit{pic}\ 1\ (\mathit{pic}\ 2\ (\mathit{res}\ \mathbf{0})\ (\mathit{res}\ \mathbf{1}))\ (\mathit{res}\ \mathbf{1}))
\end{array}
$$

$$
\begin{array}{llll}
\textbf{instance} & \mathit{TreeAlg}\ \mathit{DecTree} & \textbf{where}\ \mathit{res} = \mathit{Res}; & \mathit{pic} = \mathit{Pick}; \\
\textbf{instance} & \mathit{TreeAlg}\ \mathit{CostFun} & \textbf{where}\ \mathit{res} = \mathit{resC}; & \mathit{pic} = \mathit{pickC} \\
\textbf{instance}\ \mathit{Ring}\ a \Rightarrow & \mathit{TreeAlg}\ (\mathit{ExpCost}\ a) & \textbf{where}\ \mathit{res} = \mathit{resPoly}; & \mathit{pic} = \mathit{pickPoly}
\end{array}
$$

A class of decision tree algebras

Similarly, for decision trees, we use a class of tree algebras:

$$\begin{aligned}
&\textbf{class}\ \mathit{TreeAlg}\ a\ \textbf{where}\\
&\quad \mathit{res} :: \mathbb{B} \to a\\
&\quad \mathit{pic} :: \mathit{Index} \to a \to a \to a
\end{aligned}$$

$$\begin{aligned}
&\mathit{fold}_{DT} :: \mathit{TreeAlg}\ a \Rightarrow \mathit{DecTree} \to a\\
&\mathit{fold}_{DT}\ (\mathit{Res}\ b) \qquad\quad = \mathit{res}\ b\\
&\mathit{fold}_{DT}\ (\mathit{Pick}\ i\ t_0\ t_1) = \mathit{pic}\ i\ (\mathit{fold}_{DT}\ t_0)\ (\mathit{fold}_{DT}\ t_1)
\end{aligned}$$

$$\begin{aligned}
&\mathit{ex1} :: \mathit{TreeAlg}\ a \Rightarrow a\\
&\mathit{ex1} = \mathit{pic}\ 0\ (\mathit{pic}\ 2\ (\mathit{res}\ \mathbf{0})\ (\mathit{pic}\ 1\ (\mathit{res}\ \mathbf{0})\ (\mathit{res}\ \mathbf{1})))\\
&\qquad\qquad\quad (\mathit{pic}\ 1\ (\mathit{pic}\ 2\ (\mathit{res}\ \mathbf{0})\ (\mathit{res}\ \mathbf{1}))\ (\mathit{res}\ \mathbf{1}))
\end{aligned}$$

$$\begin{aligned}
&\textbf{instance}\ \mathit{TreeAlg}\ \mathit{DecTree}\ \textbf{where}\ \mathit{res} = \mathit{Res}; \mathit{pic} = \mathit{Pick};\\
&\textbf{data}\ \mathit{DecTree}\ \textbf{where}\\
&\quad \mathit{Res}\ \ :: \mathbb{B} \to \mathit{DecTree}\\
&\quad \mathit{Pick} :: \mathit{Index} \to \mathit{DecTree} \to \mathit{DecTree} \to \mathit{DecTree}
\end{aligned}$$

A class of decision tree algebras

Similarly, for decision trees, we use a class of tree algebras:

$$\begin{aligned}
&\mathbf{class}\ \mathit{TreeAlg}\ a\ \mathbf{where}\\
&\quad \mathit{res} :: \mathbb{B} \to a\\
&\quad \mathit{pic} :: \mathit{Index} \to a \to a \to a
\end{aligned}$$

$$\begin{aligned}
&\mathit{fold}_{DT} :: \mathit{TreeAlg}\ a \Rightarrow \mathit{DecTree} \to a\\
&\mathit{fold}_{DT}\ (\mathit{Res}\ b) \qquad\ \ = \mathit{res}\ b\\
&\mathit{fold}_{DT}\ (\mathit{Pick}\ i\ t_0\ t_1) = \mathit{pic}\ i\ (\mathit{fold}_{DT}\ t_0)\ (\mathit{fold}_{DT}\ t_1)
\end{aligned}$$

$$\begin{aligned}
&\mathit{ex1} :: \mathit{TreeAlg}\ a \Rightarrow a\\
&\mathit{ex1} = \mathit{pic}\ 0\ (\mathit{pic}\ 2\ (\mathit{res}\ \mathbf{0})\ (\mathit{pic}\ 1\ (\mathit{res}\ \mathbf{0})\ (\mathit{res}\ \mathbf{1})))\\
&\qquad\qquad\quad (\mathit{pic}\ 1\ (\mathit{pic}\ 2\ (\mathit{res}\ \mathbf{0})\ (\mathit{res}\ \mathbf{1}))\ (\mathit{res}\ \mathbf{1}))
\end{aligned}$$

$$\begin{aligned}
&\mathbf{instance}\ \mathit{TreeAlg}\ \mathit{CostFun}\ \mathbf{where}\ \mathit{res} = \mathit{resC}; \mathit{pic} = \mathit{pickC}\\
&\mathbf{type}\ \mathit{CostFun} = \mathbb{B}^n \to \mathit{Int}\\
&\mathit{resC}\ b = \mathit{const}\ 0\\
&\mathit{pickC}\ i\ c_0\ c_1 = \lambda t \to 1 + \mathbf{if}\ \mathit{index}\ t\ i\ \mathbf{then}\ c_1\ t\ \mathbf{else}\ c_0\ t
\end{aligned}$$

A class of decision tree algebras

Similarly, for decision trees, we use a class of tree algebras:

$$\begin{aligned}&\textbf{class}\ \mathit{TreeAlg}\ a\ \textbf{where}\\&\quad \mathit{res} :: \mathbb{B} \to a\\&\quad \mathit{pic} :: \mathit{Index} \to a \to a \to a\end{aligned}$$

$$\begin{aligned}&\mathit{fold}_{DT} :: \mathit{TreeAlg}\ a \Rightarrow \mathit{DecTree} \to a\\&\mathit{fold}_{DT}\ (\mathit{Res}\ b) = \mathit{res}\ b\\&\mathit{fold}_{DT}\ (\mathit{Pick}\ i\ t_0\ t_1) = \mathit{pic}\ i\ (\mathit{fold}_{DT}\ t_0)\ (\mathit{fold}_{DT}\ t_1)\end{aligned}$$

$$\begin{aligned}&\mathit{ex1} :: \mathit{TreeAlg}\ a \Rightarrow a\\&\mathit{ex1} = \mathit{pic}\ 0\ (\mathit{pic}\ 2\ (\mathit{res}\ \mathbf{0})\ (\mathit{pic}\ 1\ (\mathit{res}\ \mathbf{0})\ (\mathit{res}\ \mathbf{1})))\\&\qquad\qquad (\mathit{pic}\ 1\ (\mathit{pic}\ 2\ (\mathit{res}\ \mathbf{0})\ (\mathit{res}\ \mathbf{1}))\ (\mathit{res}\ \mathbf{1}))\end{aligned}$$

$$\begin{aligned}&\textbf{instance}\ \mathit{Ring}\ a \Rightarrow \mathit{TreeAlg}\ (\mathit{ExpCost}\ a)\ \textbf{where}\ \mathit{res} = \mathit{resPoly}; \mathit{pic} = \mathit{pickPoly}\\&\textbf{type}\ \mathit{ExpCost}\ a = \mathit{Poly}\ a\\&\mathit{resPoly}\ \_b = \mathit{zero}\\&\mathit{pickPoly}\ \_\ p_0\ p_1 = \mathit{one} + (\mathit{one} - \mathit{xP}) \times p_0 + \mathit{xP} \times p_1\end{aligned}$$

Memoization reminder

- Naively computing $\mathit{fib}\ (n+2) = \mathit{fib}\ (n+1) + \mathit{fib}\ n$ leads to exponential growth in the number of function calls.
- But if we fill in a table indexed by n with already computed results we get a the result in linear time.

Conjecture from the MSc thesis

These are the polynomials from the MSc thesis (P_t) and the best one (P_*):

$$P_t = P\,[4, 4, 7, 6, -57, 20, 68, -64, 16]$$
$$P_* = P\,[4, 4, 6, 9, -61, 23, 67, -64, 16]$$

Comparing the two polynomials shows that the new one has strictly lower expected cost than the one from the thesis. The difference is $p^2(1-p)^2(1-p+p^2)$ is illustrated here:
It is non-negative in the whole interval.
The value of both polynomials at the endpoints is 4 and the maximum of P_* is ≈ 6.14 compared to the maximum of P_t which is ≈ 6.19.