POLYBORI

advanced normal form computations

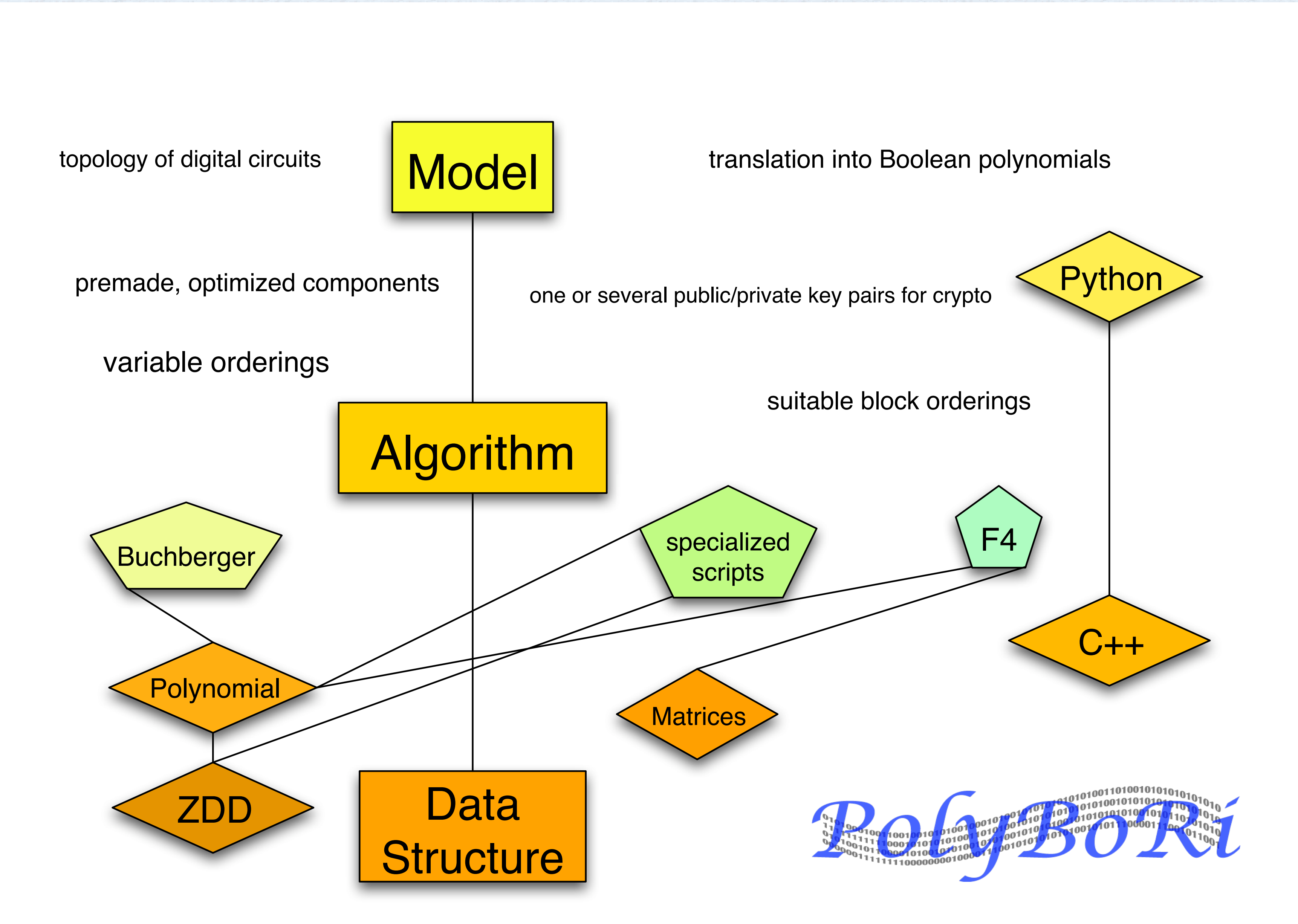
Overview

- What is PolyBoRi
- Specialized Normal Forms
  - against monomials (warmup)
  - against polynomials with linear leads
  - surprise

Goals of this talk

- Using some basic, but interesting math to show
  - what is possible in PolyBoRi
  - how to think in PolyBoRi
- No deeper knowledge of Gröbner bases required

PolyBoRi as Framework



Basic Data

PolyBoRi	***Poly***nomials over ***Bo***olean ***Ri***ngs
DFG Project	"Development, implementation and application of mathematical-algebraic algorithms for formal verification of digital systems with arithmetic blocks"
Fraunhofer ITWM	Alexander Dreyer (Department Adaptive Systems)
University of Kaiserslautern	Algebra, Geometry and Computer Algebra Group (Prof. Greuel, Department of Mathematics)
Doctoral thesis	Michael Brickenstein (Mathematisches Forschungsinstitut Oberwolfach)

ZDD-Zero Supressed Decisiion Diagrams

- Underlying data structure
- special kind of decision diagram

Boolean Polynomials

Interprete Boolean expressions as polynomials over $\mathbb{Z}_2$

logical operations $\rightarrow$ arithmetical operations and $\{\text{true}, \text{false}\} \rightarrow \{0, 1\}$

$$p \in \mathbb{Z}_2[x_1, \cdots, x_n]/\langle x_1^2 - x_1, \cdots, x_n^2 - x_n \rangle$$

Polynomials as sets

$$p = a_1 \cdot x_1^{\nu_{11}} \cdot \ldots \cdot x_n^{\nu_{1n}} + \ldots + a_{2^n} \cdot x_1^{\nu_{2^n 1}} \cdot \ldots \cdot x_n^{\nu_{2^n n}}$$

$$= \textstyle\sum_{s \in S_p} \left(\prod_{x_\nu \in s} x_\nu \right),$$

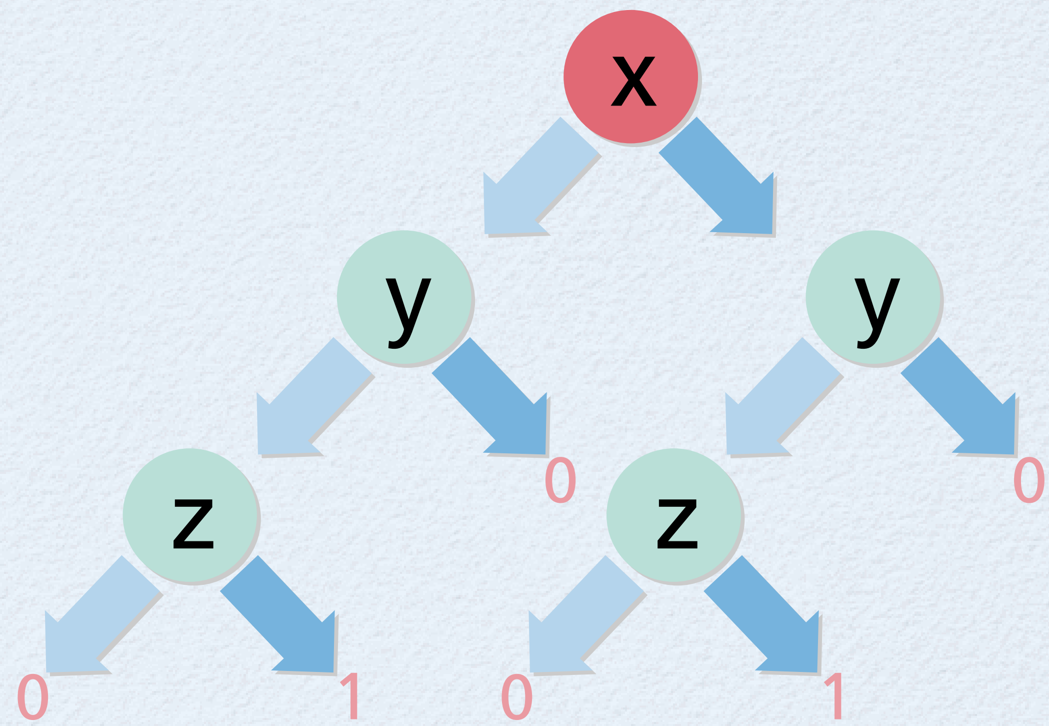
with $S_p = \{\{x_{i_1}, \cdots, x_{i_{n_1}}\}, \cdots, \{x_{i_m}, \cdots, x_{i_{n_m}}\}\}$

$$\subseteq \text{PowerSet}(x_1, \cdots, x_n)$$

Zero-Suppresed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\triangleq$ Boolean variables)

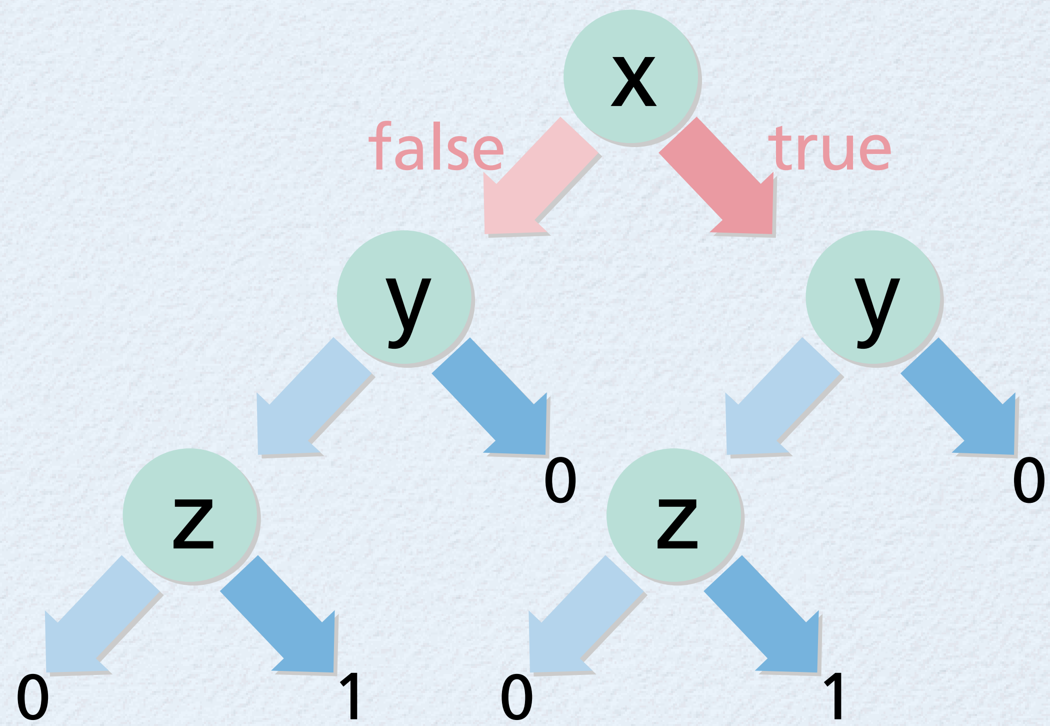


Zero-Suppresed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\widehat{=}$ Boolean variables)

Two decending edges per node (high/low or then/else) $\widehat{=}$ assign variable to true/false



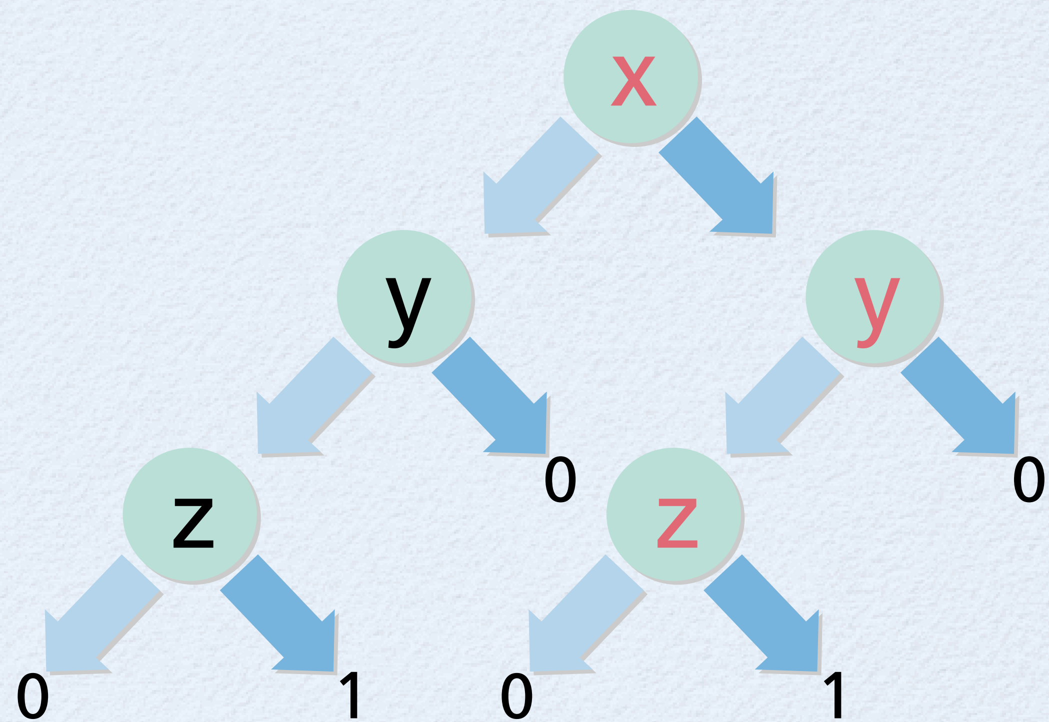
Zero-Suppresed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\widehat{=}$ Boolean variables)

Two decending edges per node (high/low or then/else) $\widehat{=}$ assign variable to true/false

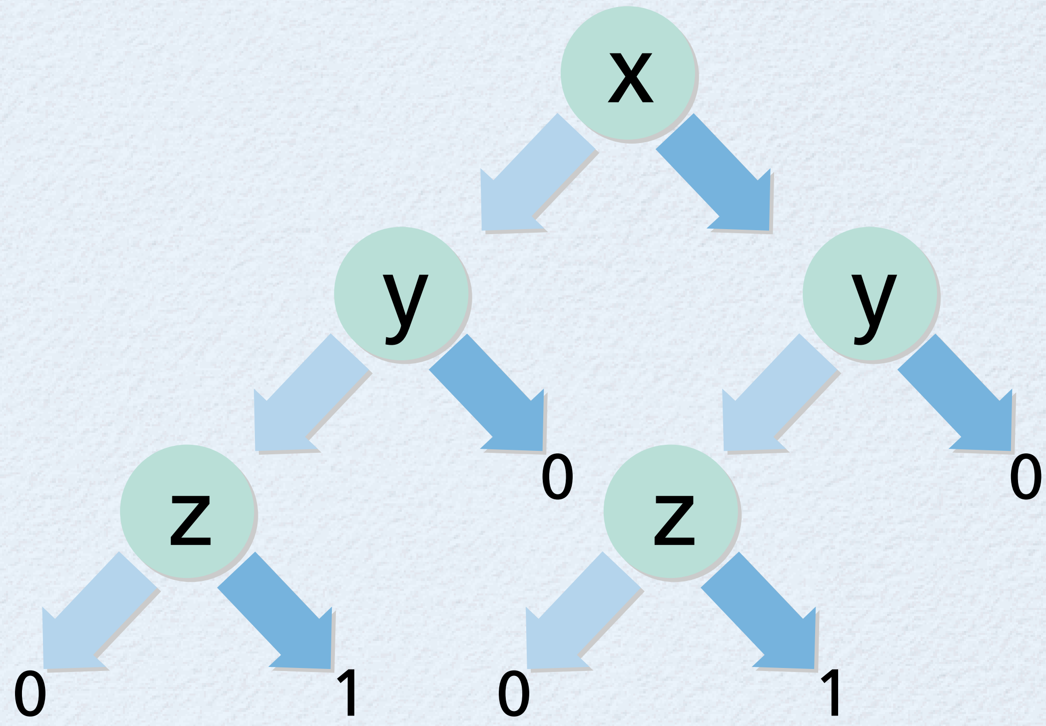
»Ordered« if the variable order is constant over all paths



Zero-Suppresed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\widehat{=}$ Boolean variables)



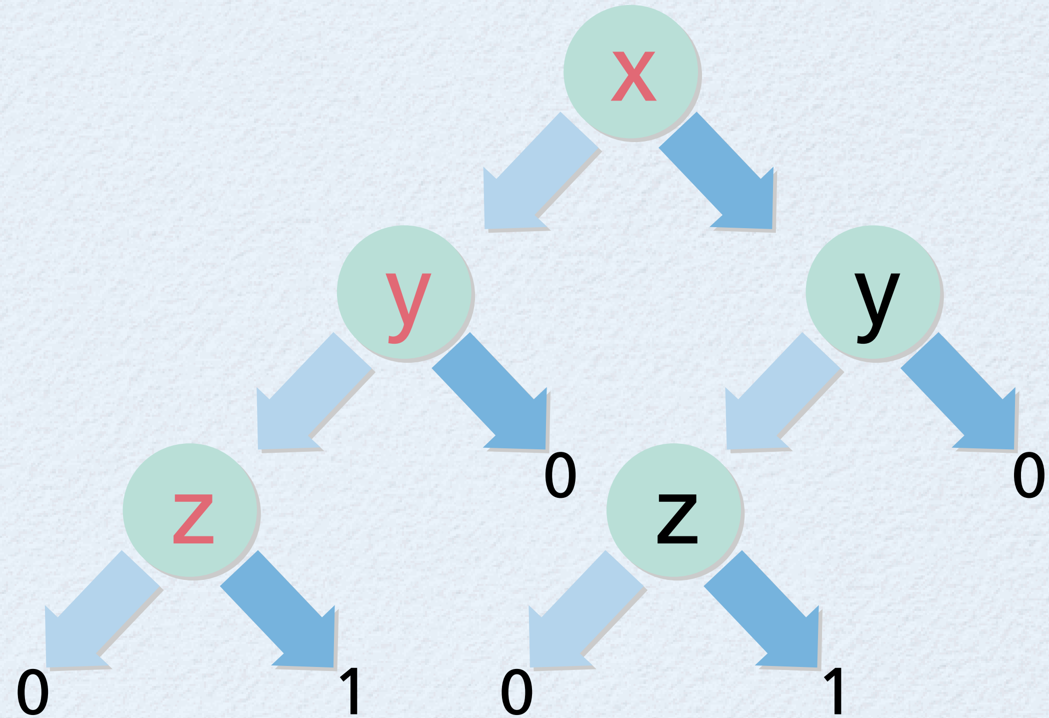
Zero -Suppresed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\triangleq$ Boolean variables)

Two decending edges per node (high/low or then/else) $\triangleq$ assign variable to true/false

»Ordered« if the variable order is constant over all paths



Zero-Suppresed Binary Decision Diagrams

Binary Decision Diagram

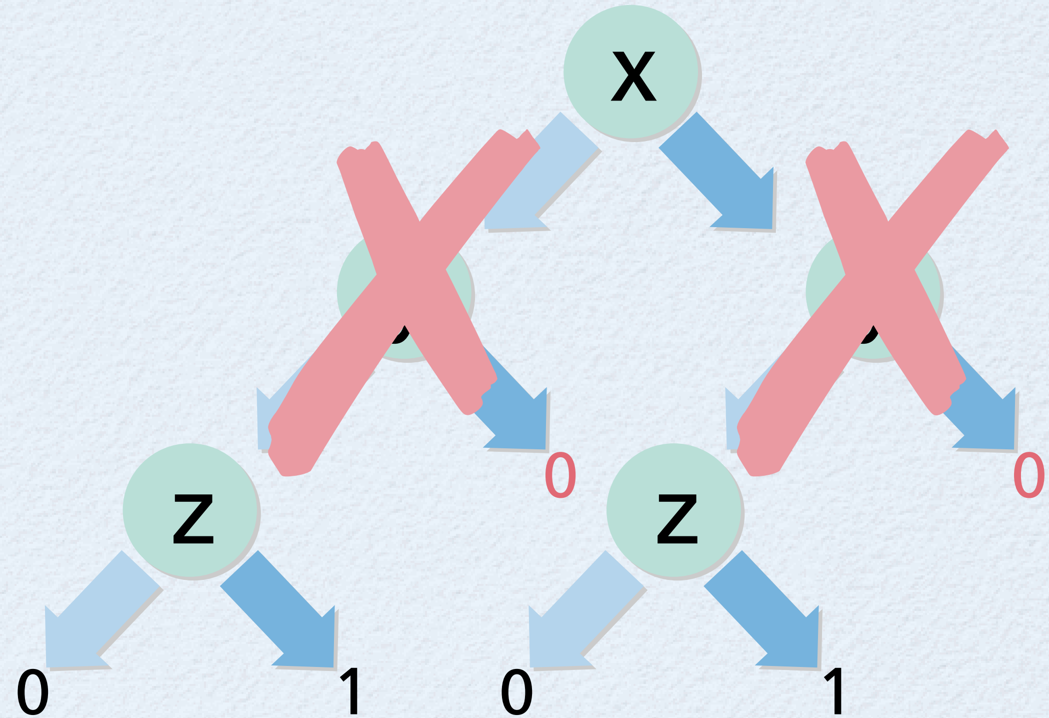
Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\widehat{=}$ Boolean variables)

Two decending edges per node (high/low or then/else) $\widehat{=}$ assign variable to true/false

»Ordered« if the variable order is constant over all paths

Zero-suppressed BDD (ZDD)

Node eliminated $\iff$ then $\rightarrow 0$



Zero -Suppresed Binary Decision Diagrams

Binary Decision Diagram

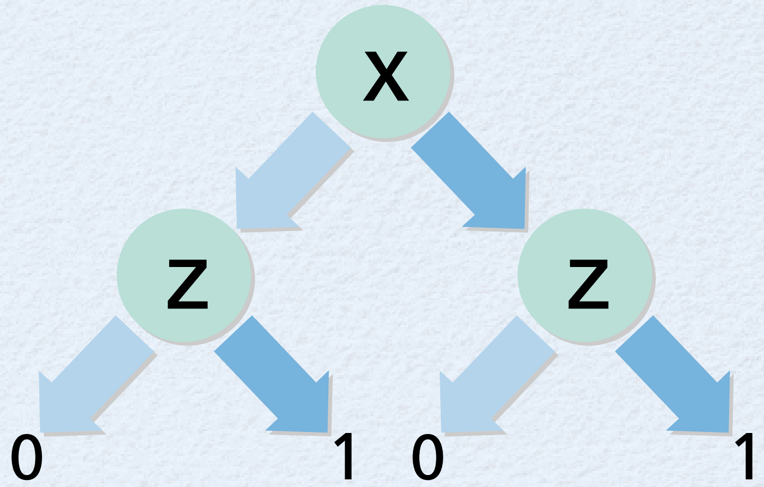
Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\widehat{=}$ Boolean variables)

Two decending edges per node (high/low or then/else) $\widehat{=}$ assign variable to true/false

»Ordered« if the variable order is constant over all paths

Zero-suppressed BDD (ZDD)

Node eliminated $\iff$ then $\rightarrow 0$



Zero-Suppresed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\widehat{=}$ Boolean variables)

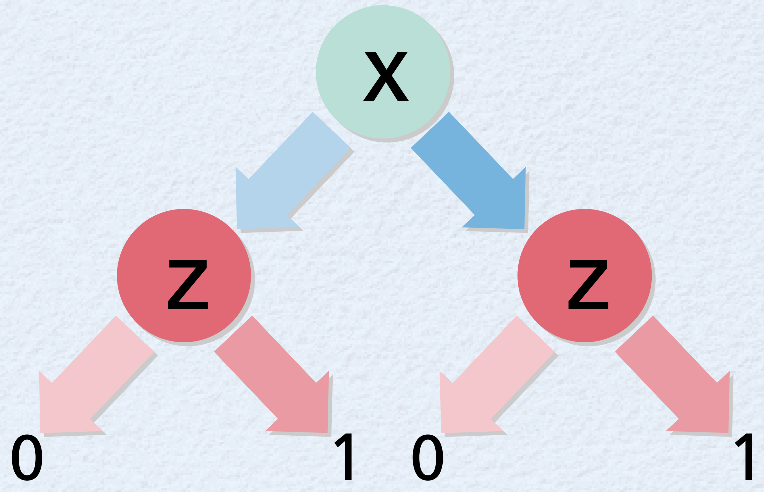
Two decending edges per node (high/low or then/else) $\widehat{=}$ assign variable to true/false

»Ordered« if the variable order is constant over all paths

Zero-suppressed BDD (ZDD)

Node eliminated $\iff$ then $\rightarrow 0$

Equal subgraphs merged



Zero-Suppresed Binary Decision Diagrams

Binary Decision Diagram

Rooted, directed, acyclic graph, terminal nodes $\{0, 1\}$, decision nodes ($\widehat{=}$ Boolean variables)

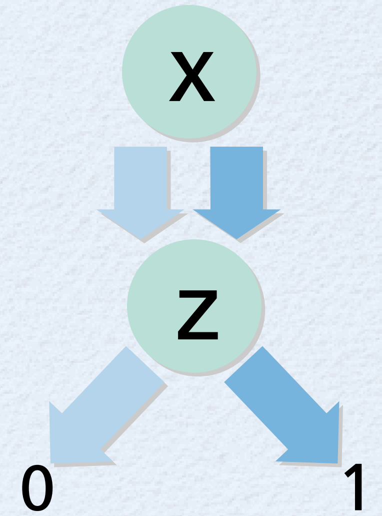
Two decending edges per node (high/low or then/else) $\widehat{=}$ assign variable to true/false

»Ordered« if the variable order is constant over all paths

Zero-suppressed BDD (ZDD)

Node eliminated $\iff$ then $\rightarrow 0$

Equal subgraphs merged



Advantages of ZDDs

Idea

ZDDs store term structure (**not** the Boolean function behind)

Reasons

- Compact data structure
- Suitable for sparse sets of subsets
- Polynomial structure recognizable
  - Valid paths $\widehat{=}$ polynomial terms
  - Natural path sequence $\widehat{=}$ lexicographical term ordering

Polynomial Arithmetic

Boolean polynomial operations	Correspond to set operations
Example	$(x + x\,y) + (x\,y + z) = x + 2\,x\,y + z \equiv x + z \iff$
	$(S_1 \cup S_2) \setminus (S_1 \cap S_2) = \{\{x\}, \{z\}\}$ (with $S_1 = \{\{x\}, \{x, y\}\}, S_2 = \{\{x, y\}, \{z\}\}$)
	$x \cdot (y + z) = x\,y + y\,z \iff$
	$\{\{x\} \cup \{y\}, \{x\} \cup \{z\}\} = \{\{x, y\}, \{x, z\}\}$
Likewise (but more complicated)	Factors/multiples of monomials, degree of a polynomial. . .
ZDD Implementation	Free C/C++ Library Cudd: Fabio Somenzi (University of Colorado)

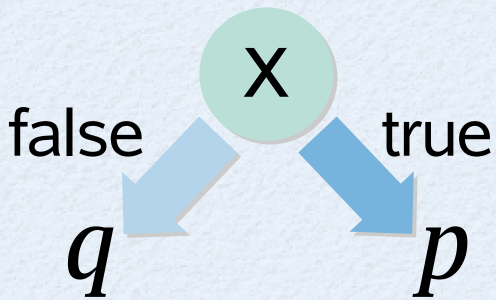
Caching and Recursion

ZDD normalform	Unique diagram root nodes $a = b \iff \texttt{rootnode}(a) \texttt{ is } \texttt{rootnode}(b)$
Reference counting	Lower memory usage (no deep copies)
Caching of operations	$a \diamond b$ never evaluated twice (also for sub-diagrams)
Advantage	Speed-up recursive procedures

Caching and Recursion

Caching and Recursion

Example: lead(f)



$$f = q + x \cdot p$$

First (lexicographical) term t in f with $\deg t = \deg f$.

Input: f Boolean polynomial **Output:** $t = \text{lead}(f)$ (deg-lex)

if $f \in \{0, 1\}$ **then**
    set $t := 1$
else if isCached(lead, f) **then**
    set $t :=$ cache(lead, f)
else
    set $x :=$ root variable of f
    if $\deg(f) = \deg(\text{thenBranch}(f)) + 1$ **then**
        set $t := x \cdot \text{lead}(\text{thenBranch}(f))$
    else
        set $t := \text{lead}(\text{elseBranch}(f))$
    end if
    insert cache(lead, f) $:= t$
end if

From ZDDs to PolyBoRi

C++-Library	High-level data types for Boolean polynomials, monomials, exponent vectors, and underlying rings Implements polynomial operations and basic functionality
ZDDs	Internal handling of polynomial structure (utilizing cache and uniqueness)
Ordering-dependend functions	Leading term computation, monomial comparisons, . . . added
Non-trivial Monomial Orderings (run- & compile-time selectable)	Lexicographic, degree-lexicographic, and degree-reverse-lexicographic (ascending variable order) orderings Orderings consisting of blocks of degree-orderings
SINGULAR Interface	Prototype

Python Interface

PolyBoRi's Python Interface

Python interface allows for
- Parsing of complex polynomial systems (Inner-domain specific language)
- Interactive use (via ipython)

Extensive testsuite
- Mainly satisfiability examples; some from cryptography

Rapid Prototyping
- Many algorithms existed in python first
- Sophisticated and easy extendable strategies for Gröbner base computation

Find a one of a Boolean polynomial

Input: Boolean polynomial $p \neq 0$

Out: $\{x_i \to v_i\}_i : p(v_1, \ldots, v_k) = 1$

```
if p = 1 then
    return ∅                                         // empty tuple
end if
Set r := rootnode(p)
Set x := Variable corresponding to r
if elseBranch(r) is 0 then
    Set v = 1
    Set q = thenBranch(r)                            // plug in 1
else
    Set v = 0
    Set q = elseBranch(r)                            // plug in 0
end if
return {x → v} ∪ find_one(q)
```

Some approach to cryptoanalysis

- many crypto system AES/CTC consists mainly of auxiliary variables and a few key variables
- Eliminate the auxiliary variables and use some other method on the remaining equations in the key variables

EXAMPLE

k3_0 + s3_0 + s3_1 + s3_2 + s2_0 + s2_1 + s2_2 + s1_0 + s1_1 + s1_2 + k0_0 + 1
k3_1 + s3_1 + s3_2 + s3_3 + s2_1 + s2_2 + s2_3 + s1_1 + s1_2 + s1_3 + k0_1
k3_2 + s3_0 + s3_2 + s3_3 + s2_0 + s2_2 + s2_3 + s1_0 + s1_2 + s1_3 + k0_2
k3_3 + s3_0 + s3_1 + s3_3 + s2_0 + s2_1 + s2_3 + s1_0 + s1_1 + s1_3 + k0_3
k3_4 + s3_4 + s3_5 + s3_6 + s2_4 + s2_5 + s2_6 + s1_4 + s1_5 + s1_6 + k0_4
k3_5 + s3_5 + s3_6 + s3_7 + s2_5 + s2_6 + s2_7 + s1_5 + s1_6 + s1_7 + k0_5 + 1
k3_6 + s3_4 + s3_6 + s3_7 + s2_4 + s2_6 + s2_7 + s1_4 + s1_6 + s1_7 + k0_6 + 1
k3_7 + s3_4 + s3_5 + s3_7 + s2_4 + s2_5 + s2_7 + s1_4 + s1_5 + s1_7 + k0_7
k3_8 + s3_0 + s3_1 + s3_2 + s1_0 + s1_1 + s1_2 + k0_0 + k0_8 + 1
k3_9 + s3_1 + s3_2 + s3_3 + s1_1 + s1_2 + s1_3 + k0_1 + k0_9
k3_10 + s3_0 + s3_2 + s3_3 + s1_0 + s1_2 + s1_3 + k0_2 + k0_10 + 1
k3_11 + s3_0 + s3_1 + s3_3 + s1_0 + s1_1 + s1_3 + k0_3 + k0_11
k3_12 + s3_4 + s3_5 + s3_6 + s1_4 + s1_5 + s1_6 + k0_4 + k0_12
k3_13 + s3_5 + s3_6 + s3_7 + s1_5 + s1_6 + s1_7 + k0_5 + k0_13
k3_14 + s3_4 + s3_6 + s3_7 + s1_4 + s1_6 + s1_7 + k0_6 + k0_14
k3_15 + s3_4 + s3_5 + s3_7 + s1_4 + s1_5 + s1_7 + k0_7 + k0_15
k2_0 + s2_0 + s2_1 + s2_2 + s1_0 + s1_1 + s1_2 + k0_0 + 1
k2_1 + s2_1 + s2_2 + s2_3 + s1_1 + s1_2 + s1_3 + k0_1 + 1
k2_2 + s2_0 + s2_2 + s2_3 + s1_0 + s1_2 + s1_3 + k0_2
k2_3 + s2_0 + s2_1 + s2_3 + s1_0 + s1_1 + s1_3 + k0_3
k2_4 + s2_4 + s2_5 + s2_6 + s1_4 + s1_5 + s1_6 + k0_4
k2_5 + s2_5 + s2_6 + s2_7 + s1_5 + s1_6 + s1_7 + k0_5
k2_6 + s2_4 + s2_6 + s2_7 + s1_4 + s1_6 + s1_7 + k0_6
k2_7 + s2_4 + s2_5 + s2_7 + s1_4 + s1_5 + s1_7 + k0_7
k2_8 + s2_0 + s2_1 + s2_2 + k0_8
k2_9 + s2_1 + s2_2 + s2_3 + k0_9
k2_10 + s2_0 + s2_2 + s2_3 + k0_10 + 1
k2_11 + s2_0 + s2_1 + s2_3 + k0_11
k2_12 + s2_4 + s2_5 + s2_6 + k0_12
k2_13 + s2_5 + s2_6 + s2_7 + k0_13 + 1
k2_14 + s2_4 + s2_6 + s2_7 + k0_14 + 1
k2_15 + s2_4 + s2_5 + s2_7 + k0_15
k1_0 + s1_0 + s1_1 + s1_2 + k0_0 + 1
k1_1 + s1_1 + s1_2 + s1_3 + k0_1 + 1
k1_2 + s1_0 + s1_2 + s1_3 + k0_2 + 1
k1_3 + s1_0 + s1_1 + s1_3 + k0_3
k1_4 + s1_4 + s1_5 + s1_6 + k0_4
k1_5 + s1_5 + s1_6 + s1_7 + k0_5 + 1
k1_6 + s1_4 + s1_6 + s1_7 + k0_6 + 1
k1_7 + s1_4 + s1_5 + s1_7 + k0_7
k1_8 + s1_0 + s1_1 + s1_2 + k0_0 + k0_8 + 1
k1_9 + s1_1 + s1_2 + s1_3 + k0_1 + k0_9 + 1
k1_10 + s1_0 + s1_2 + s1_3 + k0_2 + k0_10 + 1
k1_11 + s1_0 + s1_1 + s1_3 + k0_3 + k0_11
k1_12 + s1_4 + s1_5 + s1_6 + k0_4 + k0_12
k1_13 + s1_5 + s1_6 + s1_7 + k0_5 + k0_13 + 1

k1_9 + s1_1 + s1_2 + s1_3 + k0_1 + k0_9 + 1
k1_10 + s1_0 + s1_2 + s1_3 + k0_2 + k0_10 + 1
k1_11 + s1_0 + s1_1 + s1_3 + k0_3 + k0_11
k1_12 + s1_4 + s1_5 + s1_6 + k0_4 + k0_12
k1_13 + s1_5 + s1_6 + s1_7 + k0_5 + k0_13 + 1
k1_14 + s1_4 + s1_6 + s1_7 + k0_6 + k0_14 + 1
k1_15 + s1_4 + s1_5 + s1_7 + k0_7 + k0_15
x1_0 + w0_0*w0_1*w0_2 + w0_0*w0_2 + w0_0 + w0_1*w0_2*w0_3 + w0_1*w0
x1_1 + w0_0*w0_1*w0_3 + w0_0*w0_1 + w0_0*w0_2 + w0_1*w0_2 +
x1_2 + w0_0*w0_1 + w0_0*w0_2*w0_3 + w0_0*w0_2 + w0_0*w0_3
x1_3 + w0_0*w0_3 + w0_1*w0_2*w0_3 + w0_1*w0_3 + w0_1 + w0_2*w
x1_4 + w0_4*w0_5*w0_6 + w0_4*w0_6 + w0_4 + w0_5*w0_6*w0_7 + w0_5*w0
x1_5 + w0_4*w0_5*w0_7 + w0_4*w0_5 + w0_4*w0_6 + w0_5*w0_6 + w
x1_6 + w0_4*w0_5 + w0_4*w0_6*w0_7 + w0_4*w0_6 + w0_4*w0_7
x1_7 + w0_4*w0_7 + w0_5*w0_6*w0_7 + w0_5*w0_7 + w0_5 + w0_6*w
x1_8 + w0_8*w0_9*w0_10 + w0_8*w0_10 + w0_8 + w0_9*w0_10*w0_11 + w0_9*w
x1_9 + w0_8*w0_9*w0_11 + w0_8*w0_9 + w0_8*w0_10 + w0_9*w0_10 +
x1_10 + w0_8*w0_9 + w0_8*w0_10*w0_11 + w0_8*w0_10 + w0_8*w0_1
x1_11 + w0_8*w0_11 + w0_9*w0_10*w0_11 + w0_9*w0_11 + w0_9 + w0_10*
x1_12 + w0_12*w0_13*w0_14 + w0_12*w0_14 + w0_12 + w0_13*w0_14*w0_15 + w0_1
x1_13 + w0_12*w0_13*w0_15 + w0_12*w0_13 + w0_12*w0_14 + w0_13*w0_1
x1_14 + w0_12*w0_13 + w0_12*w0_14*w0_15 + w0_12*w0_14 + w0_12*w
x1_15 + w0_12*w0_15 + w0_13*w0_14*w0_15 + w0_13*w0_15 + w0_13 + w0_1
x2_0 + w1_0*w1_1*w1_2 + w1_0*w1_2 + w1_0 + w1_1*w1_2*w1_3 + w1_1*w1
x2_1 + w1_0*w1_1*w1_3 + w1_0*w1_1 + w1_0*w1_2 + w1_1*w1_2 +
x2_2 + w1_0*w1_1 + w1_0*w1_2*w1_3 + w1_0*w1_2 + w1_0*w1_3
x2_3 + w1_0*w1_3 + w1_1*w1_2*w1_3 + w1_1*w1_3 + w1_1 + w1_2*w
x2_4 + w1_4*w1_5*w1_6 + w1_4*w1_6 + w1_4 + w1_5*w1_6*w1_7 + w1_5*w1
x2_5 + w1_4*w1_5*w1_7 + w1_4*w1_5 + w1_4*w1_6 + w1_5*w1_6 +
x2_6 + w1_4*w1_5 + w1_4*w1_6*w1_7 + w1_4*w1_6 + w1_4*w1_7
x2_7 + w1_4*w1_7 + w1_5*w1_6*w1_7 + w1_5*w1_7 + w1_5 + w1_6*w
x2_8 + w1_8*w1_9*w1_10 + w1_8*w1_10 + w1_8 + w1_9*w1_10*w1_11 + w1_9*w
x2_9 + w1_8*w1_9*w1_11 + w1_8*w1_9 + w1_8*w1_10 + w1_9*w1_10 +
x2_10 + w1_8*w1_9 + w1_8*w1_10*w1_11 + w1_8*w1_10 + w1_8*w1_1
x2_11 + w1_8*w1_11 + w1_9*w1_10*w1_11 + w1_9*w1_11 + w1_9 + w1_10*
x2_12 + w1_12*w1_13*w1_14 + w1_12*w1_14 + w1_12 + w1_13*w1_14*w1_15 + w1_1
x2_13 + w1_12*w1_13*w1_15 + w1_12*w1_13 + w1_12*w1_14 + w1_13*w1_1
x2_14 + w1_12*w1_13 + w1_12*w1_14*w1_15 + w1_12*w1_14 + w1_12*w
x2_15 + w1_12*w1_15 + w1_13*w1_14*w1_15 + w1_13*w1_15 + w1_13 + w1_1
x3_0 + w2_0*w2_1*w2_2 + w2_0*w2_2 + w2_0 + w2_1*w2_2*w2_3 + w2_1*w2
x3_1 + w2_0*w2_1*w2_3 + w2_0*w2_1 + w2_0*w2_2 + w2_1*w2_2 +
x3_2 + w2_0*w2_1 + w2_0*w2_2*w2_3 + w2_0*w2_2 + w2_0*w2_3
x3_3 + w2_0*w2_3 + w2_1*w2_2*w2_3 + w2_1*w2_3 + w2_1 + w2_2*w
x3_4 + w2_4*w2_5*w2_6 + w2_4*w2_6 + w2_4 + w2_5*w2_6*w2_7 + w2_5*w2
x3_5 + w2_4*w2_5*w2_7 + w2_4*w2_5 + w2_4*w2_6 + w2_5*w2_6 +
x3_6 + w2_4*w2_5 + w2_4*w2_6*w2_7 + w2_4*w2_6 + w2_4*w2_7

Why is it natural?

- It is easy to formulate equations and reorder variables s.t.
  - for each auxiliary variable x we have an equation x=f, where f is a polynomial in smaller terms than x, following the encryption algorithm
  - these equations form a Gröbner basis allowing reduction of the other polynomials
- only need to calculate a few normals forms

What is the problem

- the new equations, which are normal forms of original ones are incredibly large (measured in the number of terms, size of ZDDs instead depends on some „complexity“)
- need suitable normal form algorithms, that don‘t depend on the number of terms

Special Normal Form Algorithms

NORMAL FORM ENCODING

- Problem: „CUDD recursive“ functions on ZDDs have a fixed number of arguments (1,2,3)
- Function normal form NF(f,G) where f is a Boolean polynomial, G is a set of Boolean polynomials
- How to encode G as Polynomial/ZDD?
- Encoding has to lead to a good algorithm

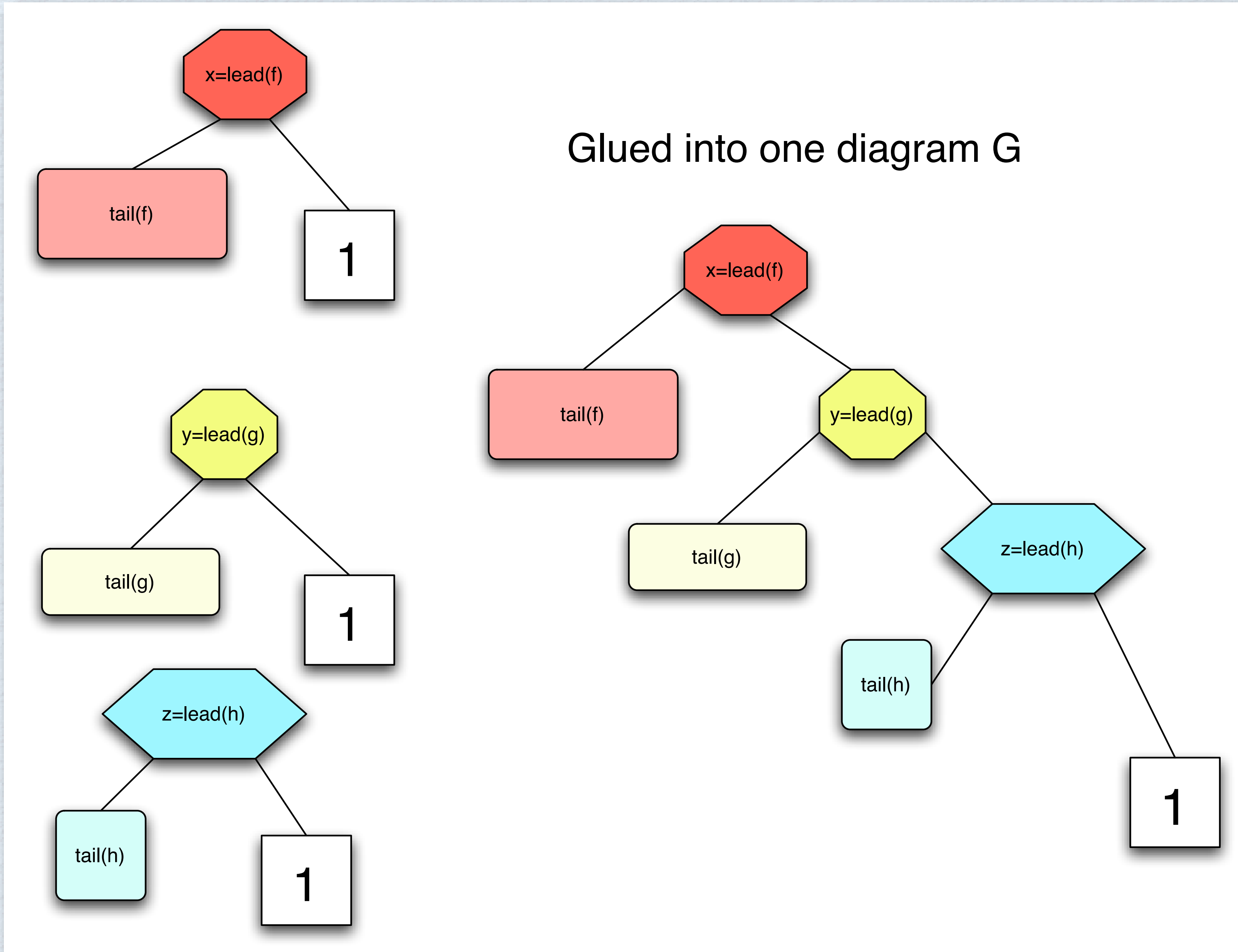
Easiest case: G is a set of monomials

- A set of monomials G is equivalent to a polynomial where the terms are the members of G
- Normal form calculation means: Cancel every term in f, which is divisible by a term in G
  - Example: $f=xy+x+y+xyz$, $G=\{x,zy\}$
    - $NF(f,G)=y$

NF(f, set of monomials G)

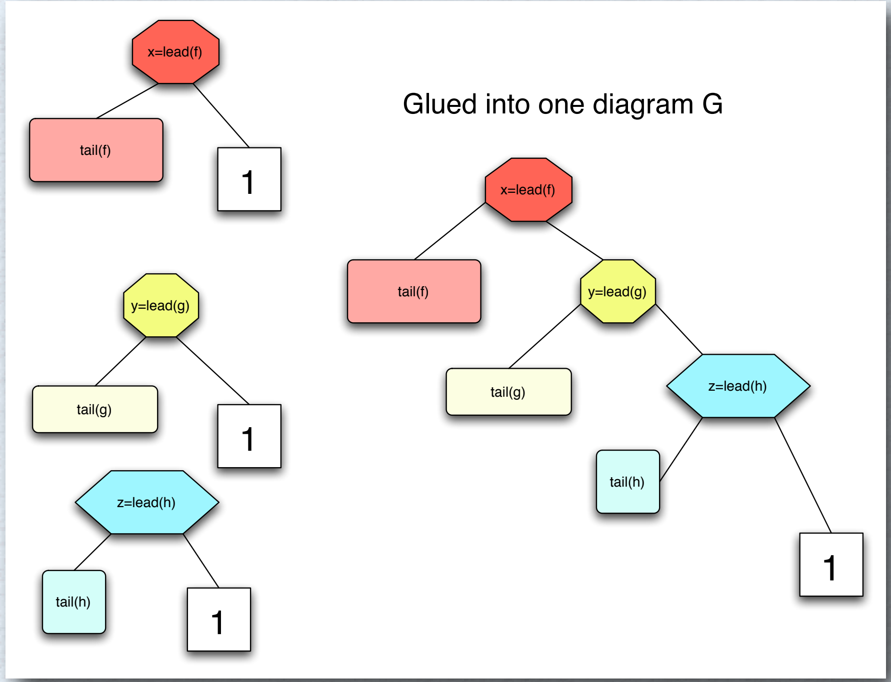
- if 1 in G: return 0
- if G is zero: return f
- while rootvarindex(f)>rootvarindex(G)
  - G=elseBranch(G)
- if f is zero or one: return f
- if rootvarindex(f)==rootvarindex(G):
  - return NF(NF(thenBranch(f),elseBranch(G),thenBranch(G)) *rootvar(f)+NF(elseBranch(G),elseBranch(G)
- else:
  - return NF(elseBranch(f),G)*NF(thenBranch(f),G)*rootvar(f)

Linear Leads Encoding



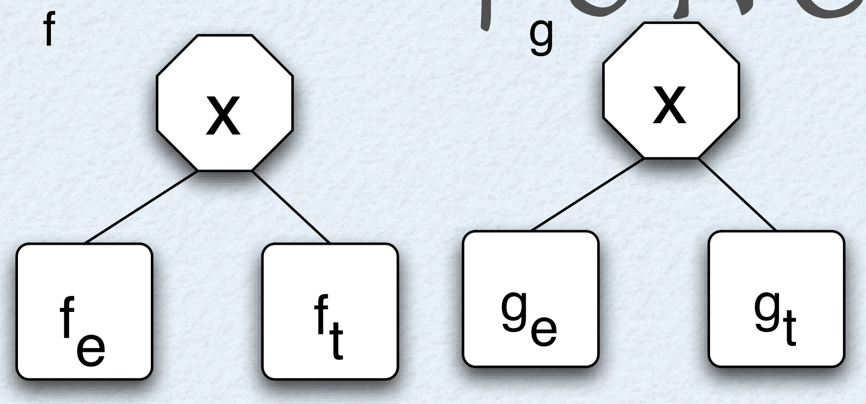
Linear Lex Lead NF(p,G)

- handle terminal cases
- ensure rootvarindex(G) =rootvarindex(p)
- next:=thenBranch(G)
- return NF(elseBranch(G),next) *NF(thenBranch(p),next)+NF (elseBranch(p),next)



In the example G= {f,g,h}

Tower of Recursive Functions



- $+$: if_then_else(x, f_t+g_t, f_e+g_e)
- $*$: if_then_else($x, f_t*g_t+f_e*g_t+f_t*g_e, f_e*g_e$)
- LLNF(f,g): g_e*LLNF(f_t,g_t) (linear leads)

Interpretation of reduced normal forms w.r.t. lex ordering in Boolean case

- Extend term ordering to ordering on Boolean polynomials by doing string like comparison with terms as literals
- Example: lex $x>y>z$
  - $x + y + z \quad >$
  - $x \quad + \quad z + 1$
- **redNF(f,G)=min{g | g in f+<G>}**

NF Problem Decomposition

- $f=x*g+h$ //x rootvar of f
- $NF(f,G)=x*g_n+h_n$
- Then
  - $g_n=\min\{l \mid \text{exists } k: x*l+k \text{ in } f+\langle G\rangle\}$ //more choice
  - $h_n=\min\{k \mid k+\mathbf{g_n} \text{ in } f+\langle G\rangle\}$ //depends on g_n
- These partial problems can be considered seperately (Cudd-recursively)

Normal form against an ideal given by a Boolean variety I(V)

- $g=\mathrm{redNF}(f,I(V))$ <=>
    - $g=$**$\min\{g \mid g$ in $f+I(V)\}$**
    - $g(x)=f(x)$ for all x in V
- So we have just to know the zeroes and ones of f in V and calculate a minimal interpolation polynomial
- Following the original interpretation of ZDDs as sets of sets, we encode the zeroes of f in I as well as the ones of f in I as ZDDs

Minimal Interpolation

- interpolate_minimal(f,zeroes,ones)
- V:=zeroes ∪ ones
    - recursive decomposition
        - result=x*g+h
        - calculate interpolation for g
            - only with prescribed values on thenBranch(V) intersect(elseBranch(g))
        - rest of function is adjusted by h

INTERPOLATION NF

number of points	time(interpolation)	len(interpolation)	#basis
100	0.01	52	287
500	0.08	253	1943
1000	0.33	485	3393
5000	5.52	2509	10319
10000	18.99	4992	17868
50000	250.95	25012	82929
100000	897.85	50093	162024

Gröbner basis size can be determined without explicit calculation

random data in 100 variables

PolyBoRi against MiniSat

	Vars./Eqs.		PolyBoRi		MiniSat	
hole8	72	297	1.88 s	56.59 MB	0.30 s	2.08 MB
hole9	90	415	8.01 s	84.04 MB	2.31 s	2.35 MB
hole10	110	561	44.40 s	97.68 MB	25.20 s	3.24 MB
hole11	132	738	643.14 s	130.83 MB	782.65 s	7.19 MB
hole12	156	949	10264.92 s	338.66 MB	22920.20 s	17.13 MB

Cracking the famous pigeon hole example by simple multiplications