A Knowledge Compilation Take on Binary Polynomial Optimization

Florent Capelli
joint work with Alberto Del Pia and Silvia di Gregorio

Université d'Artois - CRIL

September 12, 2024

Binary Polynomial Optimization

Problem definition

Binary Polynomial Optimization problem:

$$\boxed{\max_{x_1, \dots, x_n \in \{0,1\}^n} P(x_1, \dots, x_n)}$$

where P is a polynomial.

Problem definition

Binary Polynomial Optimization problem:

$$\boxed{\max_{x_1, \ldots, x_n \in \{0,1\}^n} P(x_1, \ldots, x_n)}$$

where P is a polynomial.

Observation: P may be assumed to be multilinear since $x^2 = x$ over $\{0,1\}$

$$P = \sum_{e \in E} \alpha_e \prod_{i \in e} x_i$$

where $E \subseteq 2^V$

Example

$$P(x_1, x_2, x_3) = x_1 x_2 x_3 - 2x_1 x_3 + 3x_1$$

x_1	x_2	x_3	$P(x)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	3
1	0	1	1
1	1	0	3
1	1	1	2

$P(1,0,0) = P(1,1,0) = 3$ are maximal

Example

$$P(x_1, x_2, x_3) = x_1x_2x_3 - 2x_1x_3 + 3x_1$$

x_1	x_2	x_3	$P(x)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	3
1	0	1	1
1	1	0	3
1	1	1	2

$P(1,0,0) = P(1,1,0) = 3$ are maximal

Complexity of BPO

Bad news: Solving BPO is NP-hard.

Intuition: a polynomial can encode many things:

Complexity of BPO

Bad news: Solving BPO is NP-hard.

Intuition: a polynomial can encode many things:

- $OR\left(x, y\right) = x + y - xy$ encodes $x \vee y$ on $\{0, 1\}$

Complexity of BPO

Bad news: Solving BPO is NP-hard.

Intuition: a polynomial can encode many things:

- $OR(x, y) = x + y - xy$ encodes $x \vee y$ on $\{0, 1\}$
- For a graph $G = (V, E)$, with N vertices and M edges

$$VC(V) = \sum_{\{v,w\} \in E} OR(v, w)$$

$VC\left(\tilde{V}\right) = M$ iff $\tilde{V}$ encodes a *vertex cover* of G

Complexity of BPO

Bad news: Solving BPO is NP-hard.

Intuition: a polynomial can encode many things:

- $OR(x, y) = x + y - xy$ encodes $x \vee y$ on $\{0, 1\}$
- For a graph $G = (V, E)$, with N vertices and M edges
$$VC(V) = \sum_{\{v,w\} \in E} OR(v, w)$$
$VC\left(\tilde{V}\right) = M$ iff $\tilde{V}$ encodes a *vertex cover* of G
- $MVC(V) = 2N \times (VC(V) - M) - \sum_{x \in V} v$ is maximal at $\tilde{V}$ iff $\tilde{V}$ encodes a minimal vertex cover!

Solving BPO as ILP

BPO is a **non-linear** optimization problem.

Make it linear so that we can use LP solvers!

Solving BPO as ILP

BPO is a **non-linear** optimization problem.

Make it linear so that we can use LP solvers!

$\max_{x \in \{0,1\}^V} \sum_{e \in E} \alpha_e \prod_{v \in e} x_v$ rewrites as:

$$\max \sum_{e \in E} \alpha_e y_e \text{ such that}$$
$$y_e = \prod_{v \in e} x_v$$
$$x_v, y_e \in \{0,1\}$$

Solving BPO as ILP

BPO is a **non-linear** optimization problem.

Make it linear so that we can use LP solvers!

$\max_{x \in \{0,1\}^V} \sum_{e \in E} \alpha_e \prod_{v \in e} x_v$ rewrites as:

$$\max \sum_{e \in E} \alpha_e y_e \text{ such that}$$
$$y_e \leq \prod_{v \in e} x_v \text{ and } \prod_{v \in e} x_v \leq y_e$$
$$x_v, y_e \in \{0,1\}$$

Solving BPO as ILP

BPO is a **non-linear** optimization problem.

Make it linear so that we can use LP solvers!

$\max_{x \in \{0,1\}^V} \sum_{e \in E} \alpha_e \prod_{v \in e} x_v$ rewrites as:

$$\max \sum_{e \in E} \alpha_e y_e \text{ such that}$$
$$y_e \leq x_v \text{ for } v \in e \text{ and } \prod_{v \in e} x_v \leq y_e$$
$$x_v, y_e \in \{0, 1\}$$

Solving BPO as ILP

BPO is a **non-linear** optimization problem.

Make it linear so that we can use LP solvers!

$\max_{x \in \{0,1\}^V} \sum_{e \in E} \alpha_e \prod_{v \in e} x_v$ rewrites as:

$$\max \sum_{e \in E} \alpha_e y_e \text{ such that}$$
$$y_e \leq x_v \text{ for } v \in e \text{ and } \sum_{v \in e} x_v \leq y_e - 1 + |e|$$
$$x_v, y_e \in \{0,1\}$$

Solving BPO as ILP

BPO is a **non-linear** optimization problem.

Make it linear so that we can use LP solvers!

$\max_{x \in \{0,1\}^V} \sum_{e \in E} \alpha_e \prod_{v \in e} x_v$ rewrites as:

$$\max \sum_{e \in E} \alpha_e y_e \text{ such that}$$
$$y_e \leq x_v \text{ for } v \in e \text{ and } \sum_{v \in e} x_v \leq y_e - 1 + |e|$$
$$x_v, y_e \in \{0,1\}$$

Integer Linear Program solvers can now solve it!

Solving BPO as ILP

BPO is a **non-linear** optimization problem.

Make it linear so that we can use LP solvers!

$\max_{x \in \{0,1\}^V} \sum_{e \in E} \alpha_e \prod_{v \in e} x_v$ rewrites as:

$$\max \sum_{e \in E} \alpha_e y_e \text{ such that}$$
$$y_e \leq x_v \text{ for } v \in e \text{ and } \sum_{v \in e} x_v \leq y_e - 1 + |e|$$
$$x_v, y_e \in \{0,1\}$$

Integer Linear Program solvers can now solve it!

They may stall on known-to-be easy instances. Is there an alternative way?

BPO as a Boolean Function Problem

Boolean Function

$f \subseteq \{0,1\}^X$ is a Boolean function on variables X.

An *assignment* $\tau \colon X \to \{0,1\}$ **satisfies** f iff $\tau \in f$.

Boolean Function

$f \subseteq \{0,1\}^X$ is a Boolean function on variables X.
An *assignment* τ: $X \to \{0,1\}$ **satisfies** f iff $\tau \in f$.

Example

x	y	z
0	0	0
0	0	1
0	1	0
0	1	1
1	1	1

Represented as a formula $x \Rightarrow (y \wedge z)$ or by
$(\neg x \vee y) \wedge (\neg x \vee z)$

Weighted Boolean Function

For $w\colon X \times \{0,1\} \to \mathbb{R}$ and $\tau \in \{0,1\}^X$ consider:

$$w(\tau) = \prod_{x \in X} w(x, \tau(x)) \text{ and } w(f) = \sum_{\tau \in f} w(\tau)$$

Weighted Boolean Function

For w: $X \times \{0,1\} \to \mathbb{R}$ and $\tau \in \{0,1\}^X$ consider:

$$w(\tau) = \prod_{x \in X} w(x, \tau(x)) \text{ and } w(f) = \sum_{\tau \in f} w(\tau)$$

Example

- $w(x,0) = 1$,
- $w(x,1) = 2$,
- $w(y,0) = 3$,
- $w(y,1) = -3$,
- $w(z,0) = 5$,
- $w(z,1) = -5$

x	**y**	**z**		w
0	0	0	$1*3*5$	15
0	0	1	$1*3*-5$	−15
0	1	0	$1*-3*5$	−15
0	1	1	$1*-3*-5$	15
1	1	1	$2*-3*-5$	30
$w(f)$			$15-15-15+15+30$	30

Algebraic Model Counting

Algebraic Model Counting

$$w(f) = \sum_{\tau \in f} \prod_{x \in X} w(x, \tau(x))$$

Algebraic Model Counting

$$w(f) = \bigoplus_{\tau \in f} \bigotimes_{x \in X} w(x, \tau(x))$$

where $\mathbb{K} = (K, \oplus, \otimes, 0_{\oplus}, 1_{\otimes})$ is a **semi-ring**

Algebraic Model Counting

$$w(f) = \bigoplus_{\tau \in f} \bigotimes_{x \in X} w(x, \tau(x))$$

where $\mathbb{K} = (K, \oplus, \otimes, 0_{\oplus}, 1_{\otimes})$ is a **semi-ring**

That is:

- $\oplus$, $\otimes$ commutative, associative
- $a \oplus 0_{\oplus} = a$, $b \otimes 1_{\otimes} = b$
- $\otimes$ distributes over $\oplus$:
  $(a \otimes (b \oplus c)) = (a \otimes b) \oplus (a \otimes c)$.

Algebraic Model Counting

$$w(f) = \bigoplus_{\tau \in f} \bigotimes_{x \in X} w(x, \tau(x))$$

where $\mathbb{K} = (K, \oplus, \otimes, 0_\oplus, 1_\otimes)$ is a **semi-ring**

That is:

Examples

- $\oplus$, $\otimes$ commutative, associative
- $a \oplus 0_\oplus = a$, $b \otimes 1_\otimes = b$
- $\otimes$ distributes over $\oplus$:
  $(a \otimes (b \oplus c)) = (a \otimes b) \oplus (a \otimes c)$.

Algebraic Model Counting

$$w(f) = \bigoplus_{\tau \in f} \bigotimes_{x \in X} w(x, \tau(x))$$

where $\mathbb{K} = (K, \oplus, \otimes, 0_\oplus, 1_\otimes)$ is a **semi-ring**

That is:

- $\oplus$, $\otimes$ commutative, associative
- $a \oplus 0_\oplus = a$, $b \otimes 1_\otimes = b$
- $\otimes$ distributes over $\oplus$:
  $(a \otimes (b \oplus c)) = (a \otimes b) \oplus (a \otimes c)$.

Examples

- $(\mathbb{R}, +, \times, 0, 1)$

Algebraic Model Counting

$$w(f) = \bigoplus_{\tau \in f} \bigotimes_{x \in X} w(x, \tau(x))$$

where $\mathbb{K} = (K, \oplus, \otimes, 0_\oplus, 1_\otimes)$ is a **semi-ring**

That is:

- $\oplus$, $\otimes$ commutative, associative
- $a \oplus 0_\oplus = a$, $b \otimes 1_\otimes = b$
- $\otimes$ distributes over $\oplus$:
  $(a \otimes (b \oplus c)) = (a \otimes b) \oplus (a \otimes c)$.

Examples

- $(\mathbb{R}, +, \times, 0, 1)$
- Any fields, e.g., $\mathbb{Z}/p\mathbb{Z}$

Algebraic Model Counting

$$w(f) = \bigoplus_{\tau \in f} \bigotimes_{x \in X} w(x, \tau(x))$$

where $\mathbb{K} = (K, \oplus, \otimes, 0_\oplus, 1_\otimes)$ is a **semi-ring**

That is:

- $\oplus$, $\otimes$ commutative, associative
- $a \oplus 0_\oplus = a$, $b \otimes 1_\otimes = b$
- $\otimes$ distributes over $\oplus$:
  $(a \otimes (b \oplus c)) = (a \otimes b) \oplus (a \otimes c)$.

Examples

- $(\mathbb{R}, +, \times, 0, 1)$
- Any fields, e.g., $\mathbb{Z}/p\mathbb{Z}$
- **Arctic semi-ring**:
  $(\mathbb{Q}, \max, +, -\infty, 0)$

AMC over $(\max, +)$-semiring

$$w(f) = \max_{\tau \in f} \sum_{x \in X} w(x, \tau(x))$$

This is an optimization task!

AMC over $(\max, +)$-semiring

$$w(f) = \max_{\tau \in f} \sum_{x \in X} w(x, \tau(x))$$

This is an optimization task!

Example

- $w(x,0) = 1,$
- $w(x,1) = 2,$
- $w(y,0) = 3,$
- $w(y,1) = -3,$
- $w(z,0) = 5,$
- $w(z,1) = -5$

x	**y**	**z**		w
0	0	0	$1+3+5$	9
0	0	1	$1+3-5$	-1
0	1	0	$1-3+5$	3
0	1	1	$1-3-5$	-7
1	1	1	$2-3-5$	-6
$w(f)$			$\max(9, -1, 3, -7, -6)$	9

Encoding BPO as Boolean function

Example: $P(x_1, x_2, x_3) = x_1 x_2 x_3 - 2x_1 x_3 + 3x_1$

Encoding BPO as Boolean function

Example: $P(x_1, x_2, x_3) = x_1x_2x_3 - 2x_1x_3 + 3x_1$

- $f_P = (Y_1 \Leftrightarrow (X_1 \wedge X_2 \wedge X_3)) \wedge (Y_2 \Leftrightarrow (X_1 \wedge X_3)) \wedge (Y_3 \Leftrightarrow X_1)$
- $w_P(Y_1, 1) = 1$, $w_P(Y_2, 1) = -2$ and $w_P(Y_3, 1) = 3$.
- $w_P(Z, b) = 0$ for every other values.

Encoding BPO as Boolean function

Example: $P(x_1, x_2, x_3) = x_1x_2x_3 - 2x_1x_3 + 3x_1$

- $f_P = (Y_1 \Leftrightarrow (X_1 \wedge X_2 \wedge X_3)) \wedge (Y_2 \Leftrightarrow (X_1 \wedge X_3)) \wedge (Y_3 \Leftrightarrow X_1)$
- $w_P(Y_1, 1) = 1$, $w_P(Y_2, 1) = -2$ and $w_P(Y_3, 1) = 3$.
- $w_P(Z, b) = 0$ for every other values.

For every $\tau \vDash f_P$ we have

$$w_P(\tau) = P(x_1 = \tau(X_1), x_2 = \tau(X_2), x_3 = \tau(X_3))$$

and $w_P(f) = \max P$

Encoding BPO as Boolean function

Example: $P(x_1, x_2, x_3) = x_1x_2x_3 - 2x_1x_3 + 3x_1$

- $f_P = (Y_1 \Leftrightarrow (X_1 \wedge X_2 \wedge X_3)) \wedge (Y_2 \Leftrightarrow (X_1 \wedge X_3)) \wedge (Y_3 \Leftrightarrow X_1)$
- $w_P(Y_1, 1) = 1$, $w_P(Y_2, 1) = -2$ and $w_P(Y_3, 1) = 3$.
- $w_P(Z, b) = 0$ for every other values.

For every $\tau \models f_P$ we have
$w_P(\tau) = P(x_1 = \tau(X_1), x_2 = \tau(X_2), x_3 = \tau(X_3))$
and $w_P(f) = \max P$

Example

For $\tau(X_1) = 1, \tau(X_2) = 0, \tau(X_3) = 1$:

- if $\tau \models f_P$ then $\tau(Y_1) = 0, \tau(Y_2) = 1, \tau(Y_3) = 1$
- hence $w_P(\tau) = w_P(Y_2, 1) + w_P(Y_3, 1) = -2 + 3 = 1!$

Formal encoding

For $P := \sum_{e \in E} \alpha_e \prod_{i \in e} x_i$ define:

$$f_P := \bigwedge_{e \in E} C_e$$

where $C_e := Y_e \Leftrightarrow \bigwedge_{i \in e} X_i$

C_e **encodes** $y_e = \prod_{i \in e} x_i$**!**

Formal encoding

For $P := \sum_{e \in E} \alpha_e \prod_{i \in e} x_i$ define:

$$f_P := \bigwedge_{e \in E} C_e$$

where $C_e := Y_e \Leftrightarrow \bigwedge_{i \in e} X_i$

$\boxed{C_e \textbf{ encodes } y_e = \prod_{i \in e} x_i\textbf{!}}$

and w_P on $(\mathbb{Q}, \max, +, -\infty, 0)$ as:

- $w_P(Y_e, 1) = \alpha_e$ and
- $w_P(X_i, b) = w_P(Y_e, 0) = 0$ for $b \in \{0, 1\}$.

BPO as a Boolean Function

Theorem

$$w_P(f_P) = \max P(x_1, \ldots, x_n)$$

over the $(\max, +)$-semiring.

The underlying algorithmic toolbox is very different from ILP solvers providing **new insights**.

We can use existing toolbox for AMC to solve BPO

- **theoretical results**
- **AND practical results**

Knowledge Compilation

how to solve AMC

Representing Boolean functions

How can we represent Boolean function: $f \subseteq \{0, 1\}^X$

So far we have seen: *list* every satisfying assignment of f (aka **Truth Table**)

- Easy to manipulate since the representation is explicit
- Not **compact**

CNF Formulas

$F = \bigwedge \left(\bigvee \ell \right)$ where ℓ is a literal x or $\neg x$ for some variable x.

Examples

CNF Formulas

$F = \bigwedge \left(\bigvee \ell \right)$ where ℓ is a literal x or $\neg x$ for some variable x.

Examples

$$F_1 = (x \vee \neg y) \wedge (\neg x \vee y)$$

x	y	F_1
0	0	1
1	1	1
$*$	$*$	0

CNF Formulas

$F = \bigwedge \left(\bigvee \ell \right)$ where ℓ is a literal x or $\neg x$ for some variable x.

Examples

$F_1 = (x \vee \neg y) \wedge (\neg x \vee y)$

x	y	F_1
0	0	1
1	1	1
*	*	0

$F_2 = (x \vee \neg z) \wedge (\neg x \vee y) \wedge (x \vee y \vee z)$

x	y	z	F_2
1	1	1	1
0	1	0	1
1	1	0	1
*	*	*	0

The SAT Problem

CNF formulas are extremely simple yet can encode many interesting problems.

Theorem

Cook, Levin, 1971: The problem SAT of deciding whether a CNF formula is *satisfiable* is **NP-complete**.

Valiant 1979: The problem #SAT of counting the satisfying assignment of a CNF formula is **#P-complete**.

The SAT Problem

CNF formulas are extremely simple yet can encode many interesting problems.

Theorem

Cook, Levin, 1971: The problem SAT of deciding whether a CNF formula is *satisfiable* is **NP-complete**.

Valiant 1979: The problem #SAT of counting the satisfying assignment of a CNF formula is **#P-complete**.

- Very unlikely that efficient algorithms exists for solving SAT / #SAT
- Thriving community nevertheless addresses these problems in practice
- **SAT Solver** very efficient in many applications

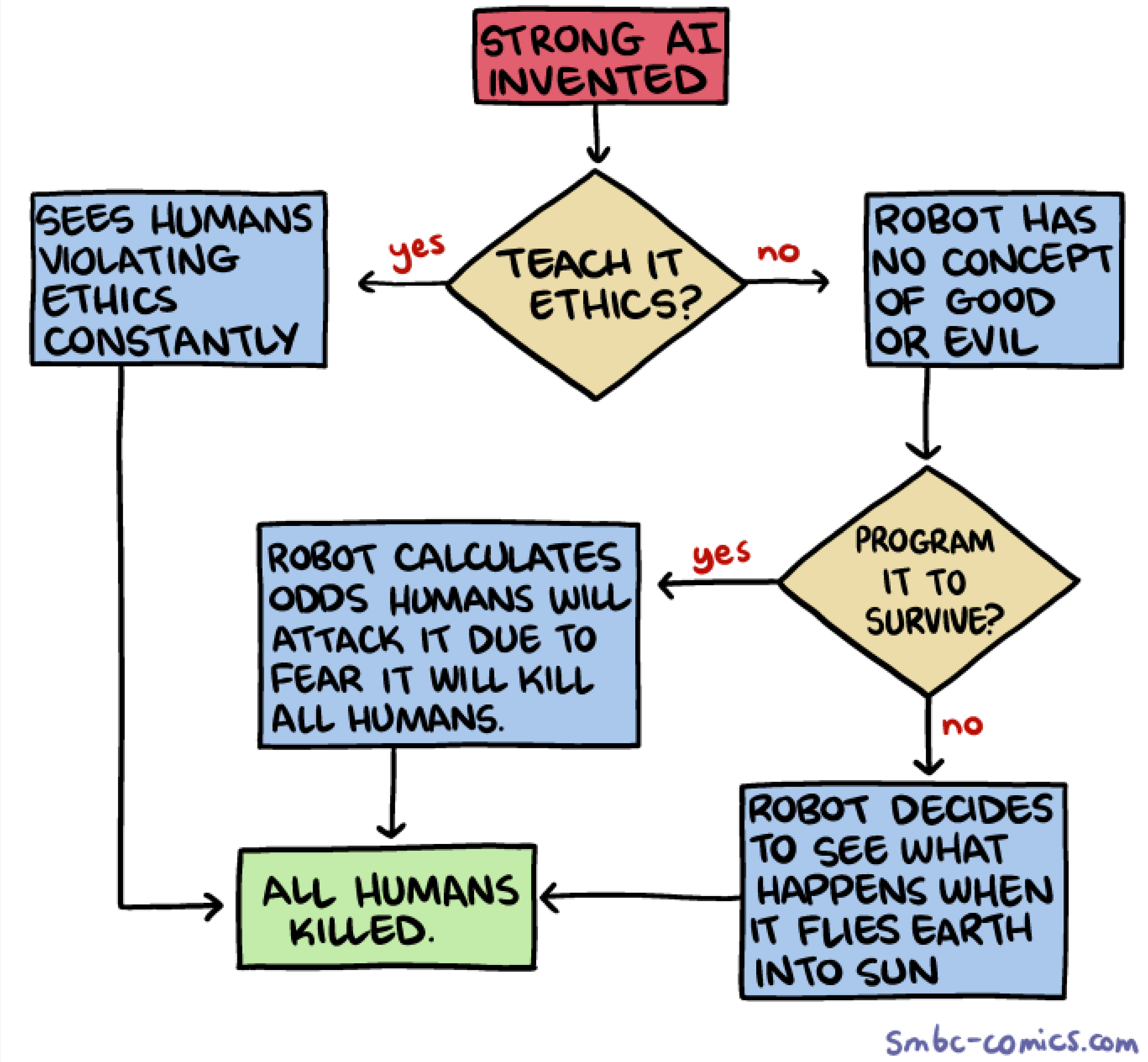
Relevance of CNF formulas

- **Natural encoding**: succinctly encodes many problems, witnessed by the many existing industrial benchmarks.
- **Intractable** for algebraic model counting

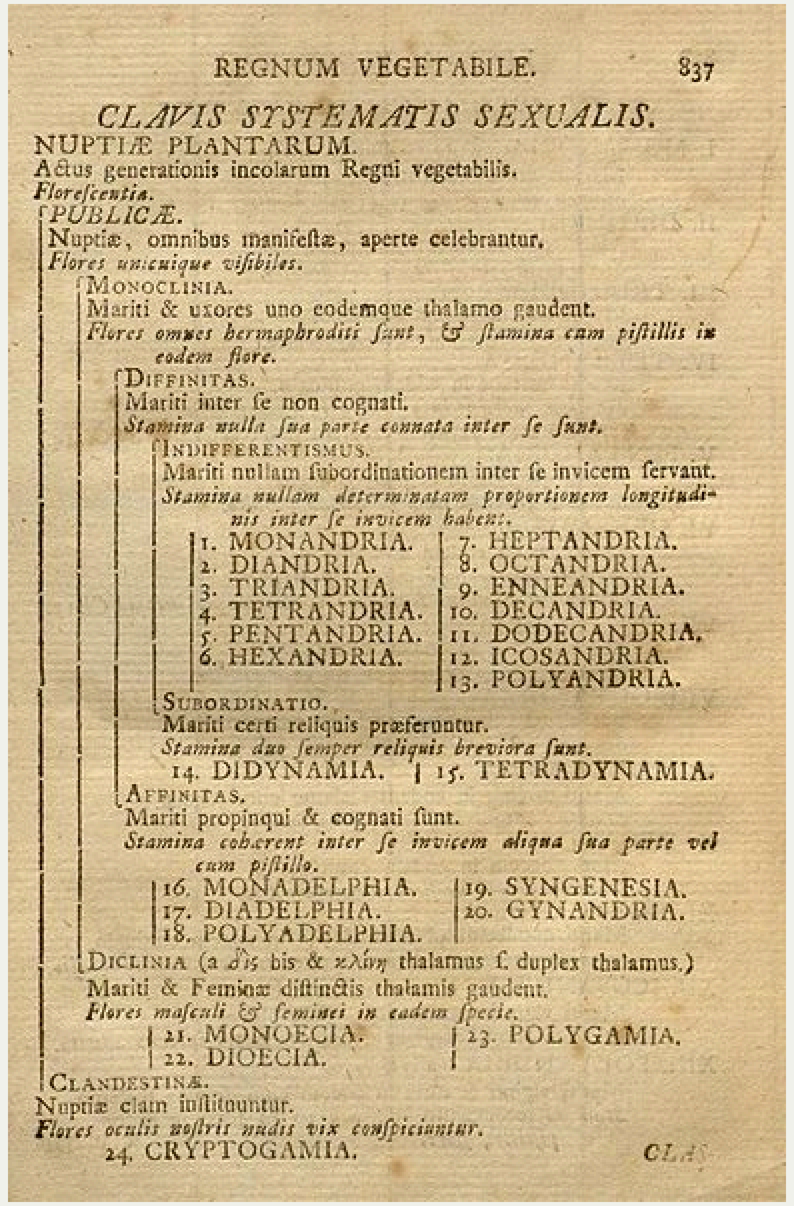
Looking for tradeoffs between Truth Tables and CNFs!

Circuit Based Representations

Research has focused on *factorized representation*.



Taken from SMBC Comics



REGNUM VEGETABILE. 837

CLAVIS SYSTEMATIS SEXUALIS.

NUPTIÆ PLANTARUM.
Actus generationis incolarum Regni vegetabilis.
Floreſcentia.

PUBLICÆ.
Nuptiæ, omnibus manifeſtæ, aperte celebrantur.
Flores unicuique viſibiles.

MONOCLINIA.
Mariti & uxores uno eodemque thalamo gaudent.
Flores omnes hermaphroditi ſunt, & ſtamina cum piſtillis in eodem flore.

DIFFINITAS.
Mariti inter ſe non cognati.
Stamina nulla ſua parte connata inter ſe ſunt.

INDIFFERENTISMUS.
Mariti nullam ſubordinationem inter ſe invicem ſervant.
Stamina nullam determinatam proportionem longitudinis inter ſe invicem habent.

1. MONANDRIA.	7. HEPTANDRIA.
2. DIANDRIA.	8. OCTANDRIA.
3. TRIANDRIA.	9. ENNEANDRIA.
4. TETRANDRIA.	10. DECANDRIA.
5. PENTANDRIA.	11. DODECANDRIA.
6. HEXANDRIA.	12. ICOSANDRIA.
	13. POLYANDRIA.

SUBORDINATIO.
Mariti certi reliquis præferuntur.
Stamina duo ſemper reliquis breviora ſunt.
14. DIDYNAMIA. | 15. TETRADYNAMIA.

AFFINITAS.
Mariti propinqui & cognati ſunt.
Stamina cohærent inter ſe invicem aliqua ſua parte vel cum piſtillo.

16. MONADELPHIA.	19. SYNGENESIA.
17. DIADELPHIA.	20. GYNANDRIA.
18. POLYADELPHIA.	

DICLINIA (a δὶς bis & κλίνη thalamus ſ. duplex thalamus.)
Mariti & Feminæ diſtinctis thalamis gaudent.
Flores maſculi & feminei in eadem ſpecie.

21. MONOECIA.	23. POLYGAMIA.
22. DIOECIA.	

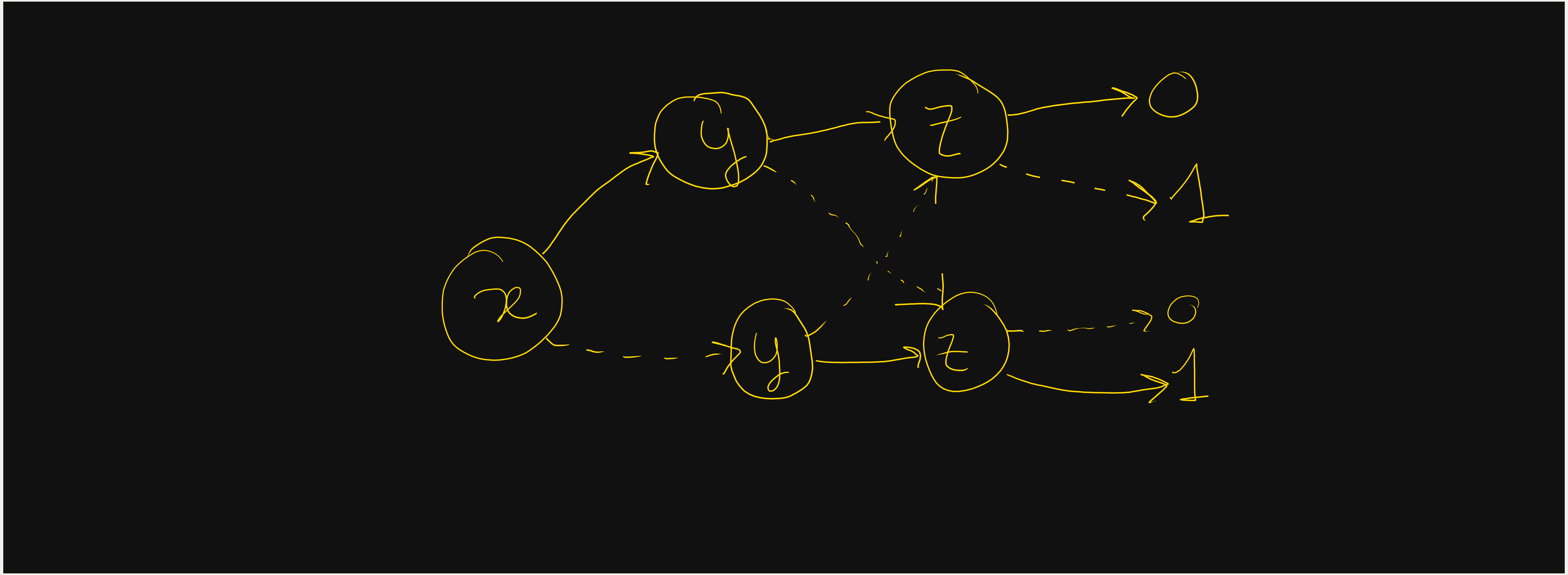
CLANDESTINÆ.
Nuptiæ clam inſtituuntur.
Flores oculis noſtris nudis vix conſpiciuntur.
24. CRYPTOGAMIA.

CLAS

Carl von Linné (1707-1778)

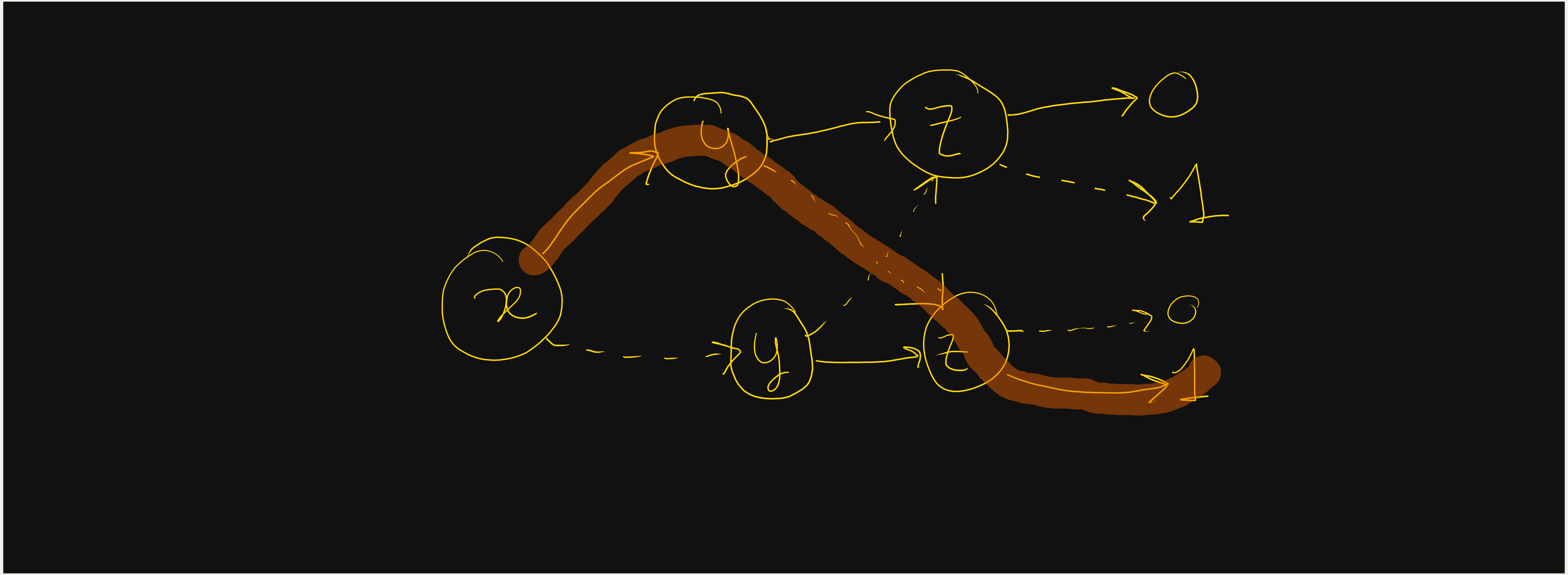
An example

Data structure based on decision nodes to represent “ $(\ x + y + z\ )$ is even”.



An example

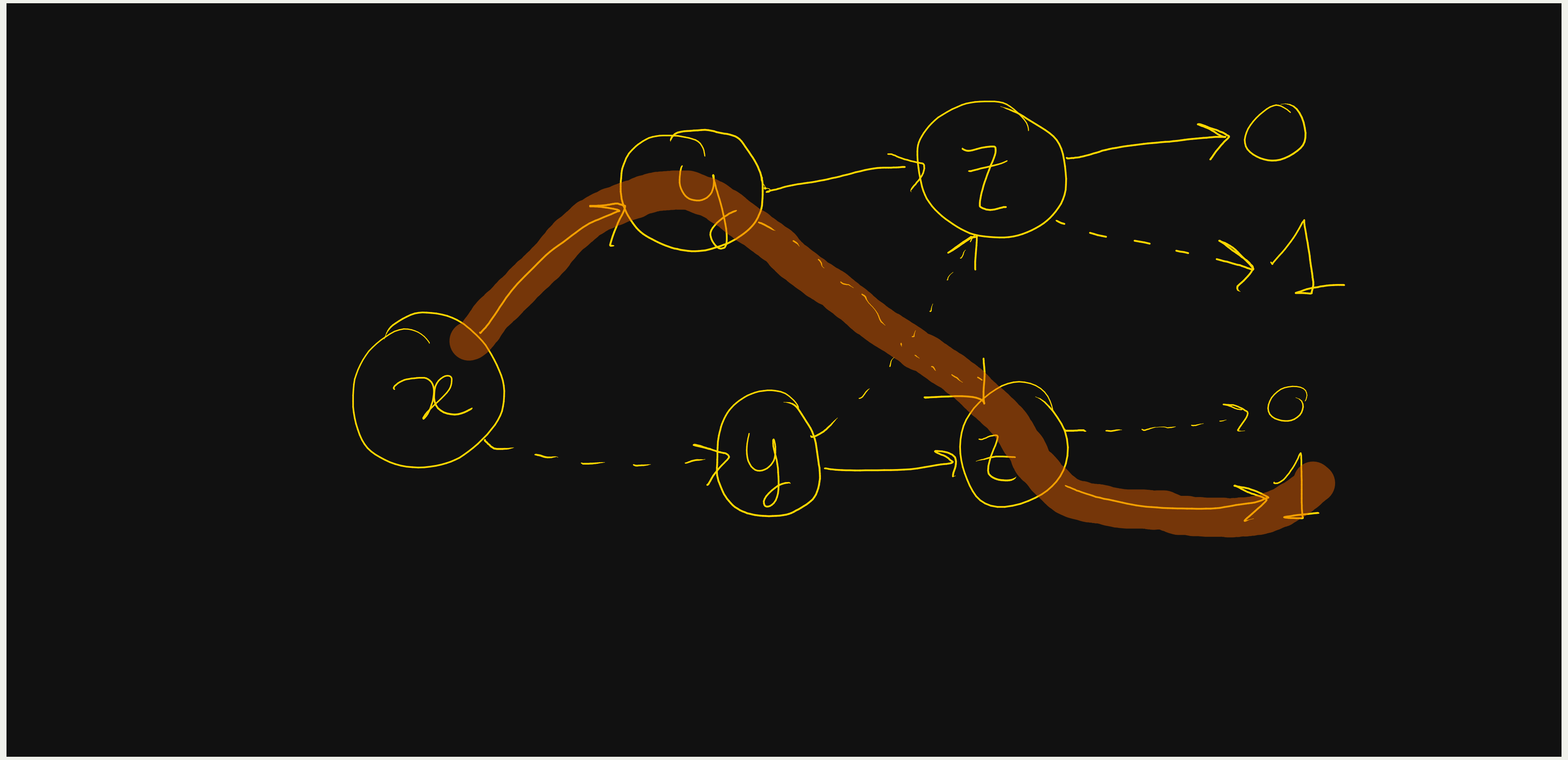
Data structure based on decision nodes to represent “ $(\; x + y + z \;)$ is even”.



Path for $x = 1$, $y = 0$ and $z = 1$ is accepting.

OBDDs

Previous data structure are **Ordered Binary Decision Diagrams**.



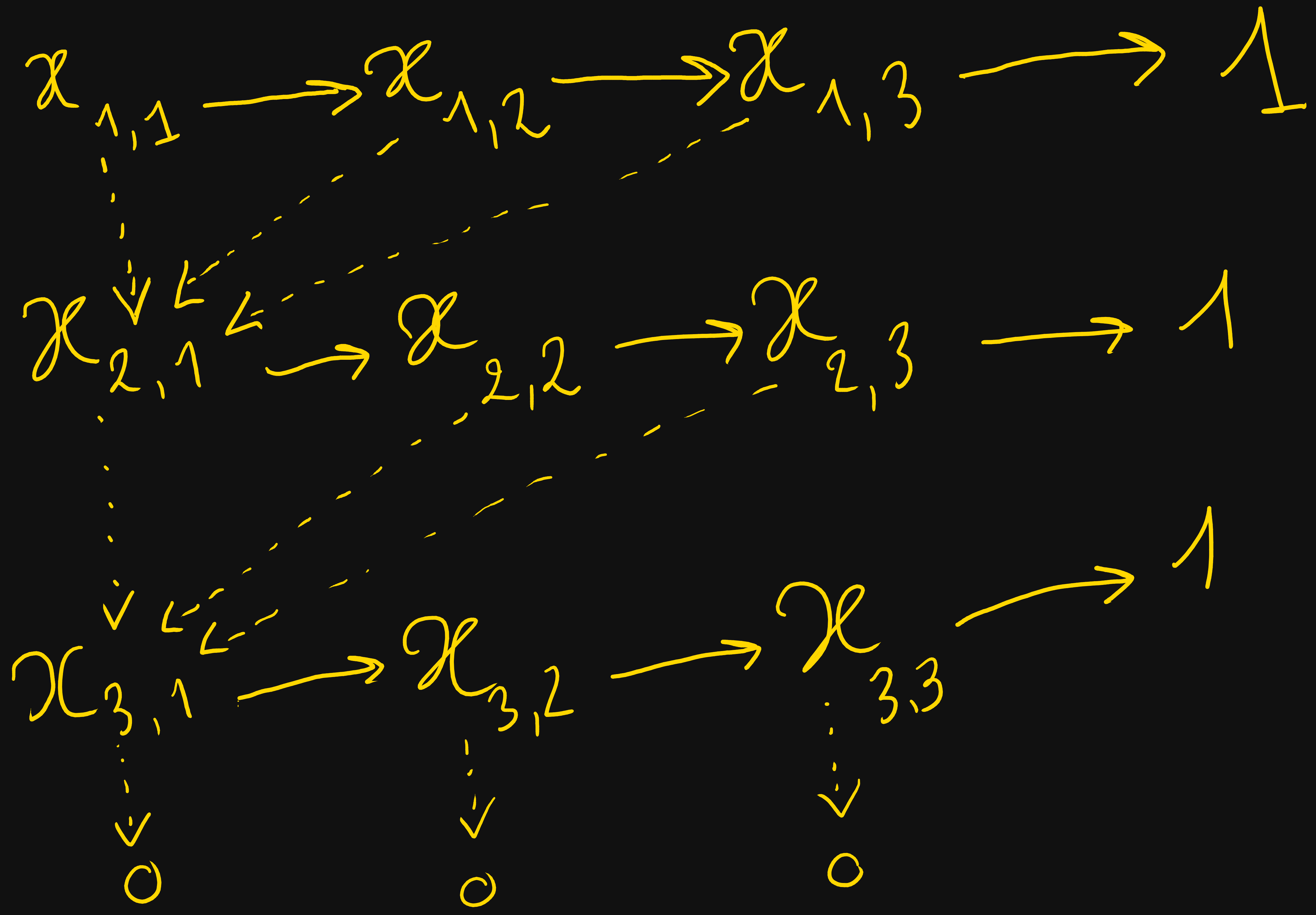
- Directed Acyclic graphs with one source
- Sinks are labeled by 0 or 1
- Internal nodes are decision nodes on a variable in $x_1, \dots, x_n$
- Variables tested in order.

Counting with OBDDs

How many 3×3 $\{0, 1\}$-matrices have a row full of ones?

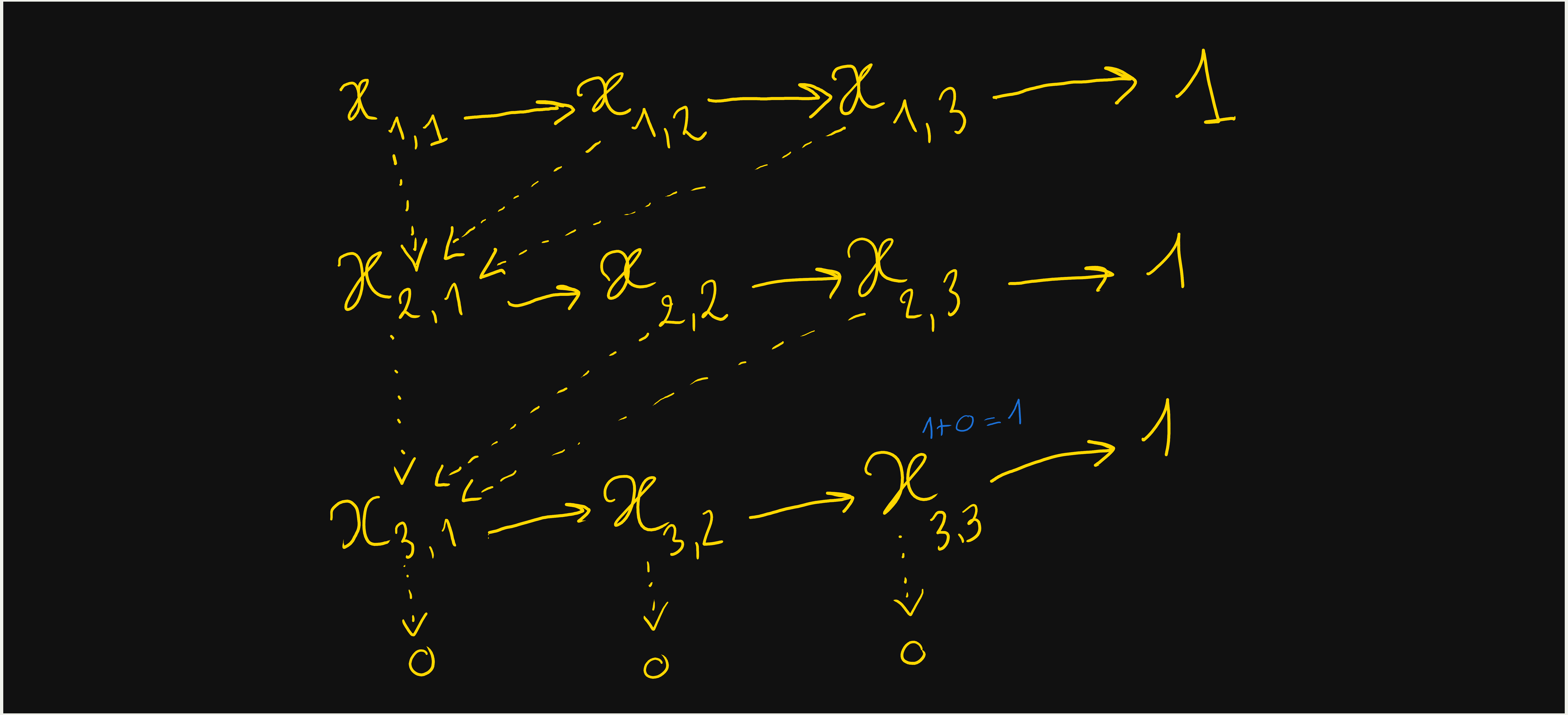
Counting with OBDDs

How many 3×3 $\{0, 1\}$-matrices have a row full of ones?



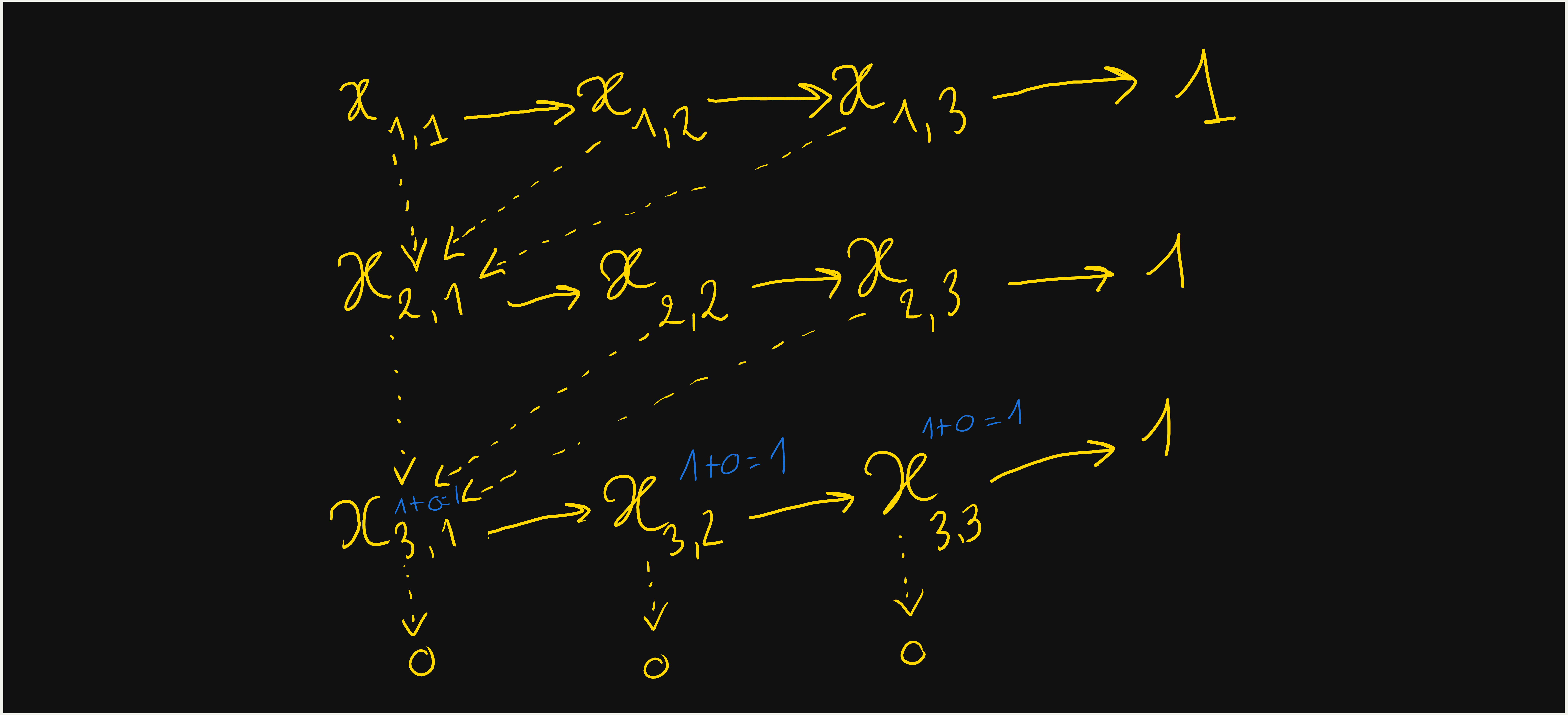
Counting with OBDDs

How many 3×3 $\{0, 1\}$-matrices have a row full of ones?



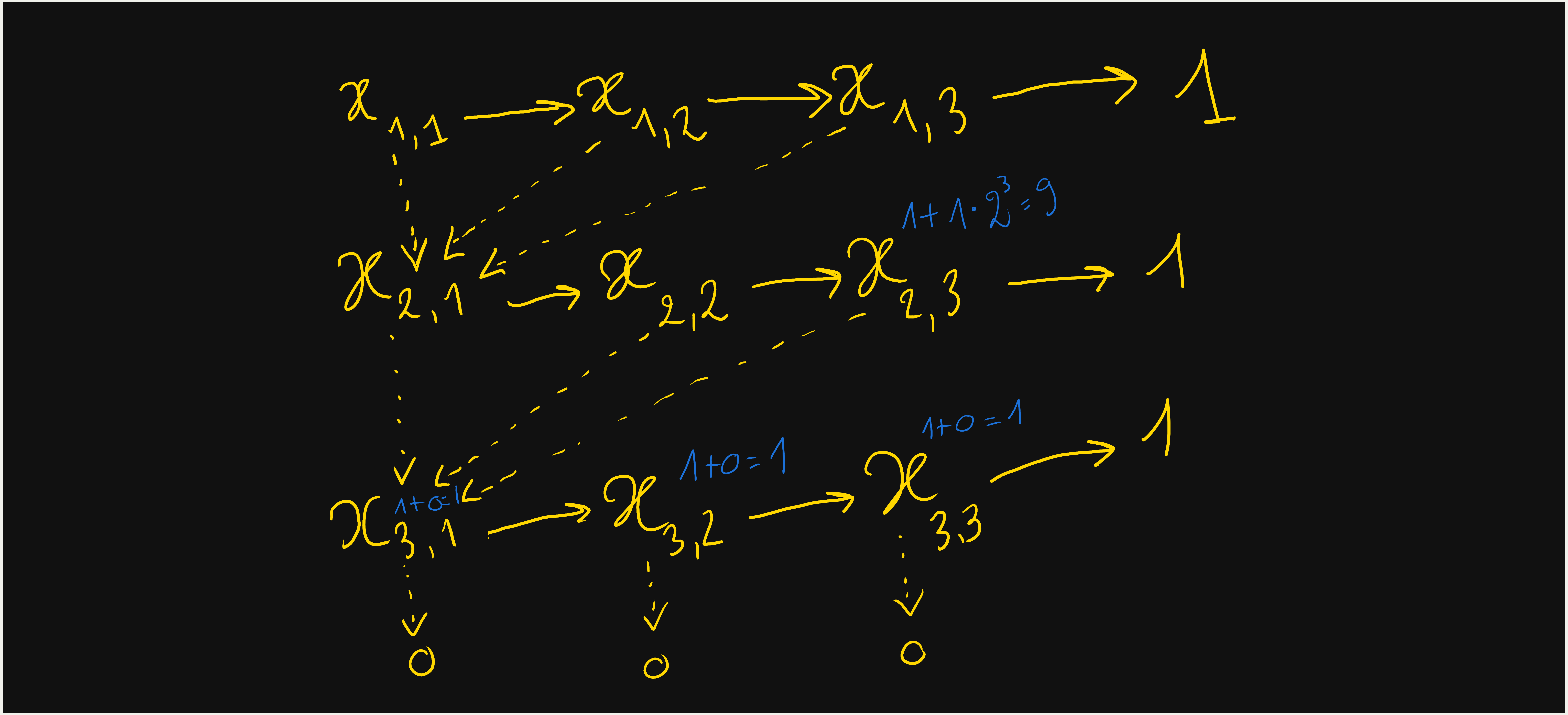
Counting with OBDDs

How many 3×3 $\{0,1\}$-matrices have a row full of ones?



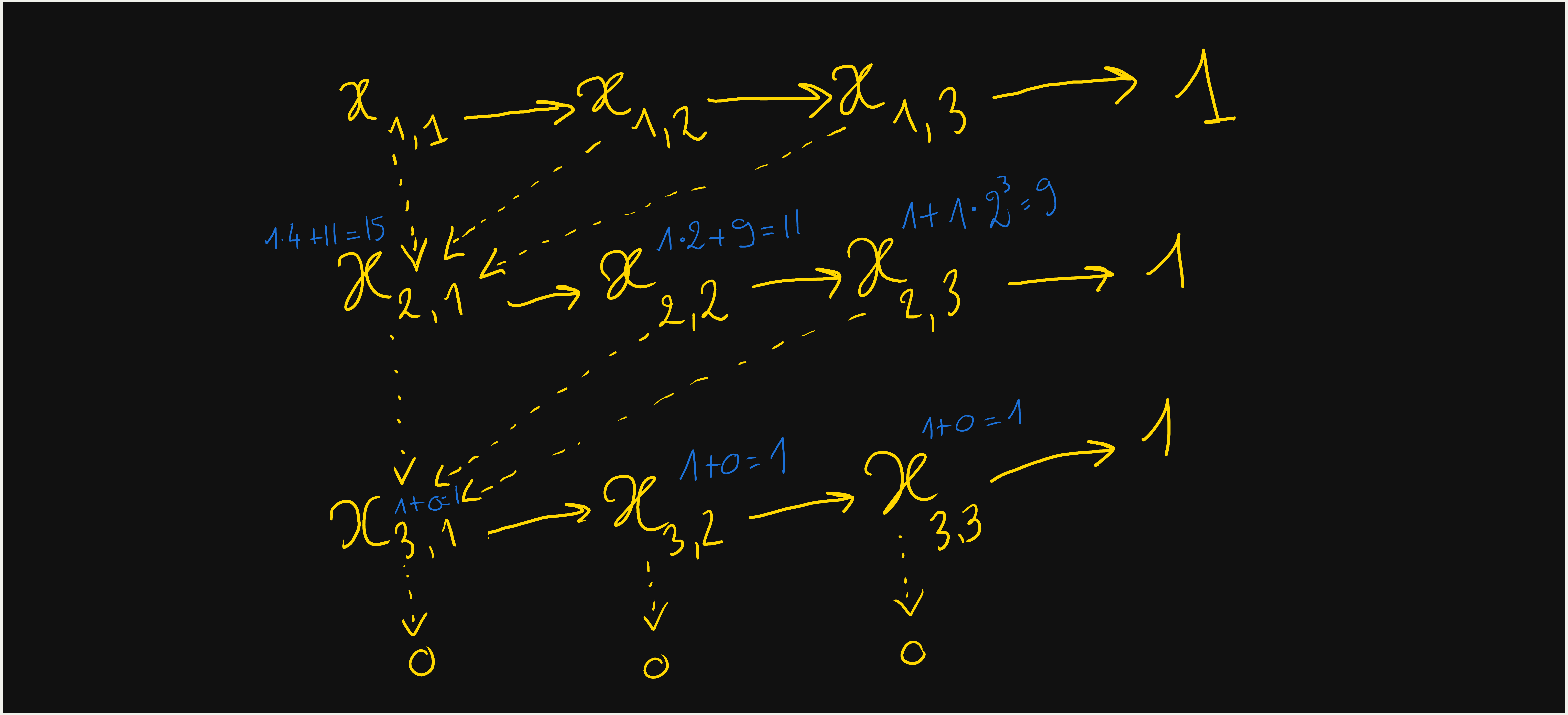
Counting with OBDDs

How many 3×3 $\{0,1\}$-matrices have a row full of ones?



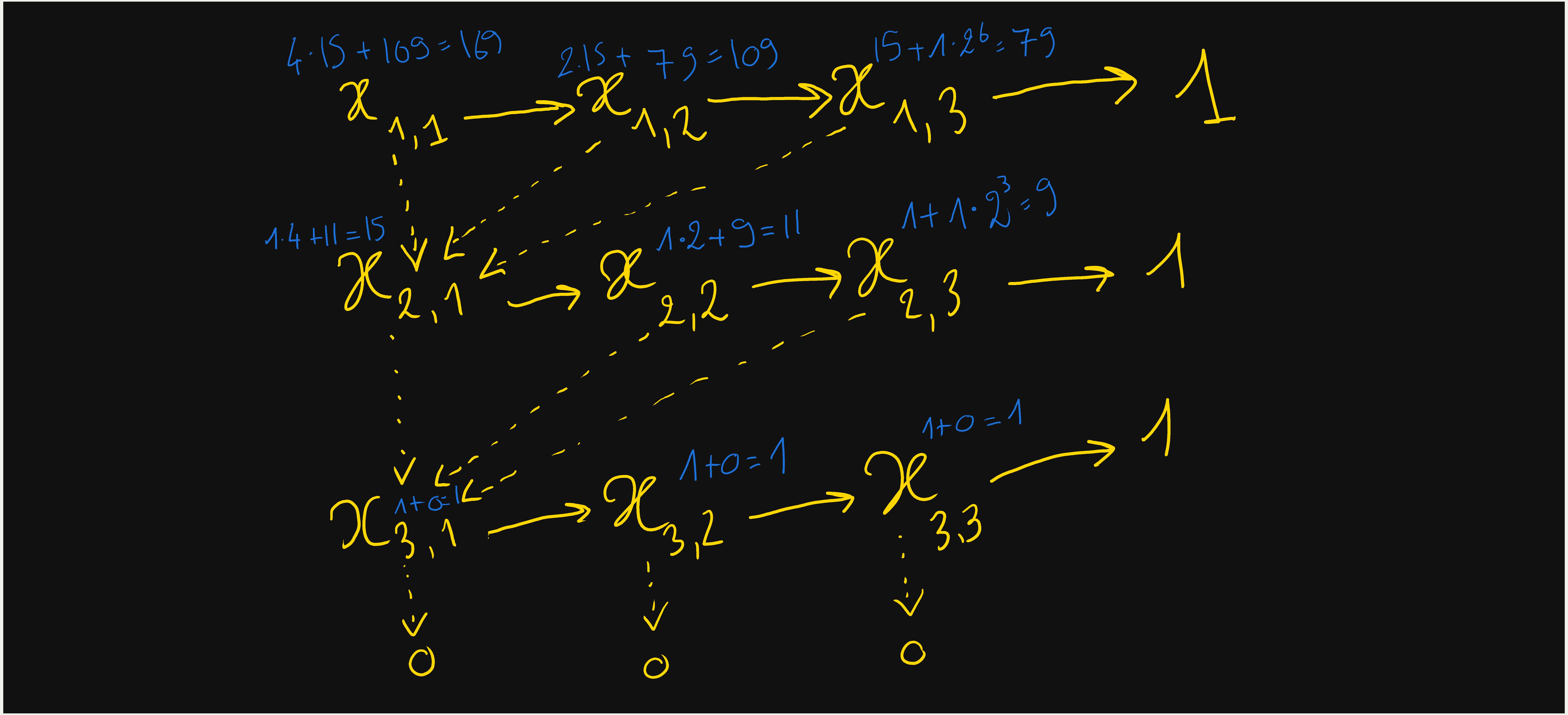
Counting with OBDDs

How many 3×3 $\{0,1\}$-matrices have a row full of ones?



Counting with OBDDs

How many 3×3 $\{0, 1\}$-matrices have a row full of ones?



Tractability of OBDDs

This idea can be generalized to any OBDDs:

Theorem

Let $f \subseteq \{0,1\}^X$ be a function computed by an OBDD having E edges. We can compute $\#f$ with $O(E)$ arithmetic operations.

Tractability of OBDDs

This idea can be generalized to any OBDDs:

Theorem

Let $f \subseteq \{0,1\}^X$ be a function computed by an OBDD having E edges. We can compute $\#f$ with $O(E)$ arithmetic operations.

Generalises to many tasks:

- Evaluate $Pr(f)$ if probabilities $Pr(x=1)$ are given for each $x \in X$
- Enumerate f
- **Algebraic Model Counting** on any semi-ring.

Back to BPO

Theorem

$w_P(f_P) = \max P(x_1, ..., x_n)$ where

- $f_P = \bigwedge_{e \in E} Y_e \Leftrightarrow \bigwedge_{i \in e} X_i$
- $w_P(Y_e, 1) = \alpha_e$ and 0 otherwise

Back to BPO

Theorem

$w_P(f_P) = \max P(x_1, \ldots, x_n)$ where

- $f_P = \bigwedge_{e \in E} Y_e \Leftrightarrow \bigwedge_{i \in e} X_i$
- $w_P(Y_e, 1) = \alpha_e$ and 0 otherwise

Solving BPO:

- transform f_P into an OBDD
- compute $w_P(f_P)$ via dynamic programming on the **OBDD itself**

Back to BPO

Theorem

$$w_P\,(\,f_P\,)\ =\ \max\,P(\,x_1,\ldots,x_n\,)\ \text{ where}$$

- $f_P = \bigwedge_{e\,\in\,E} Y_e \Leftrightarrow \bigwedge_{i\,\in\,e} X_i$
- $w_P\,(\,Y_e, 1\,)\ = \alpha_e$ and 0 otherwise

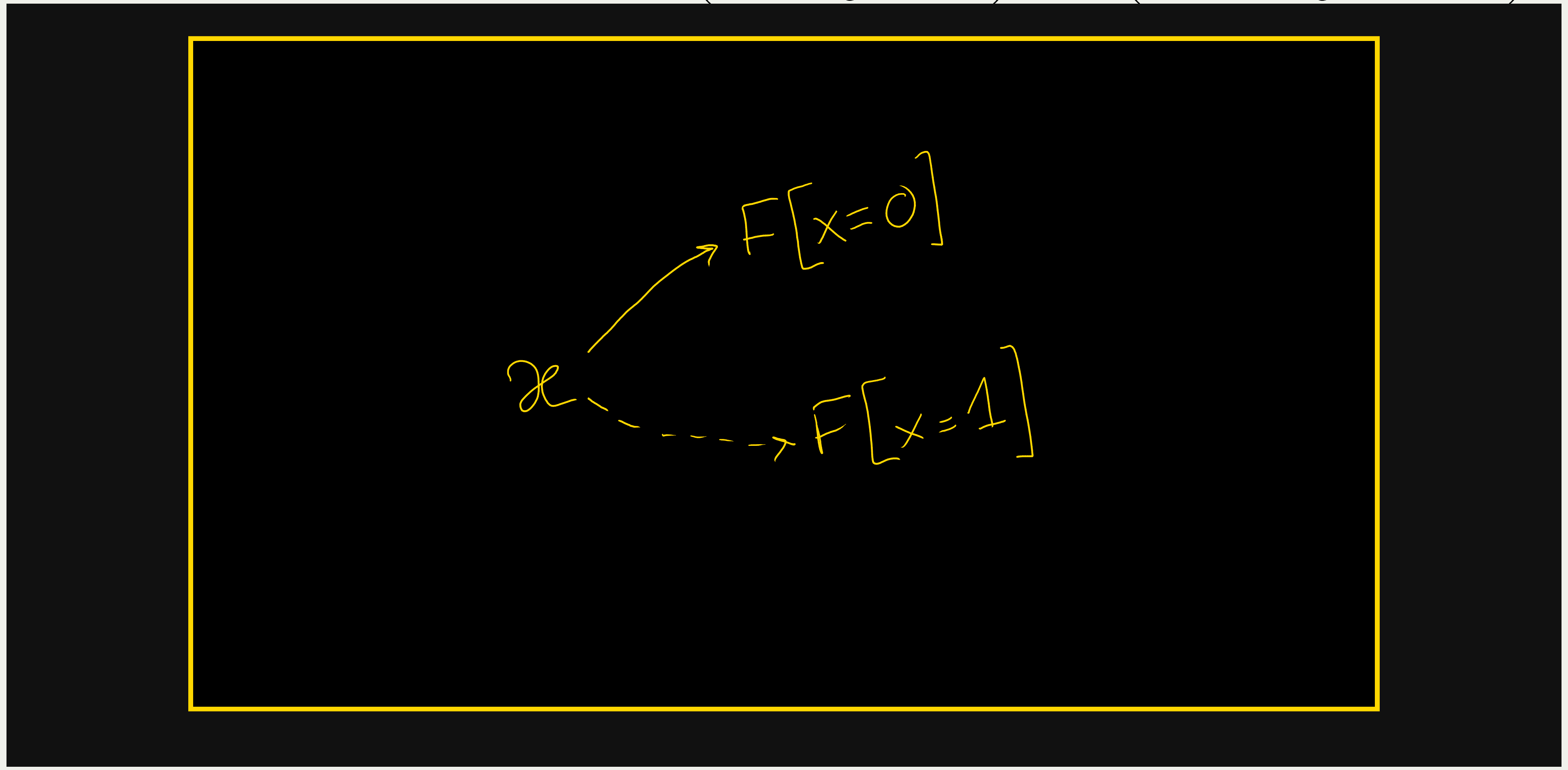
Solving BPO:

- transform f_P into an OBDD
- compute $w_P\,(\,f_P\,)$ via dynamic programming on the **OBDD itself**

A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

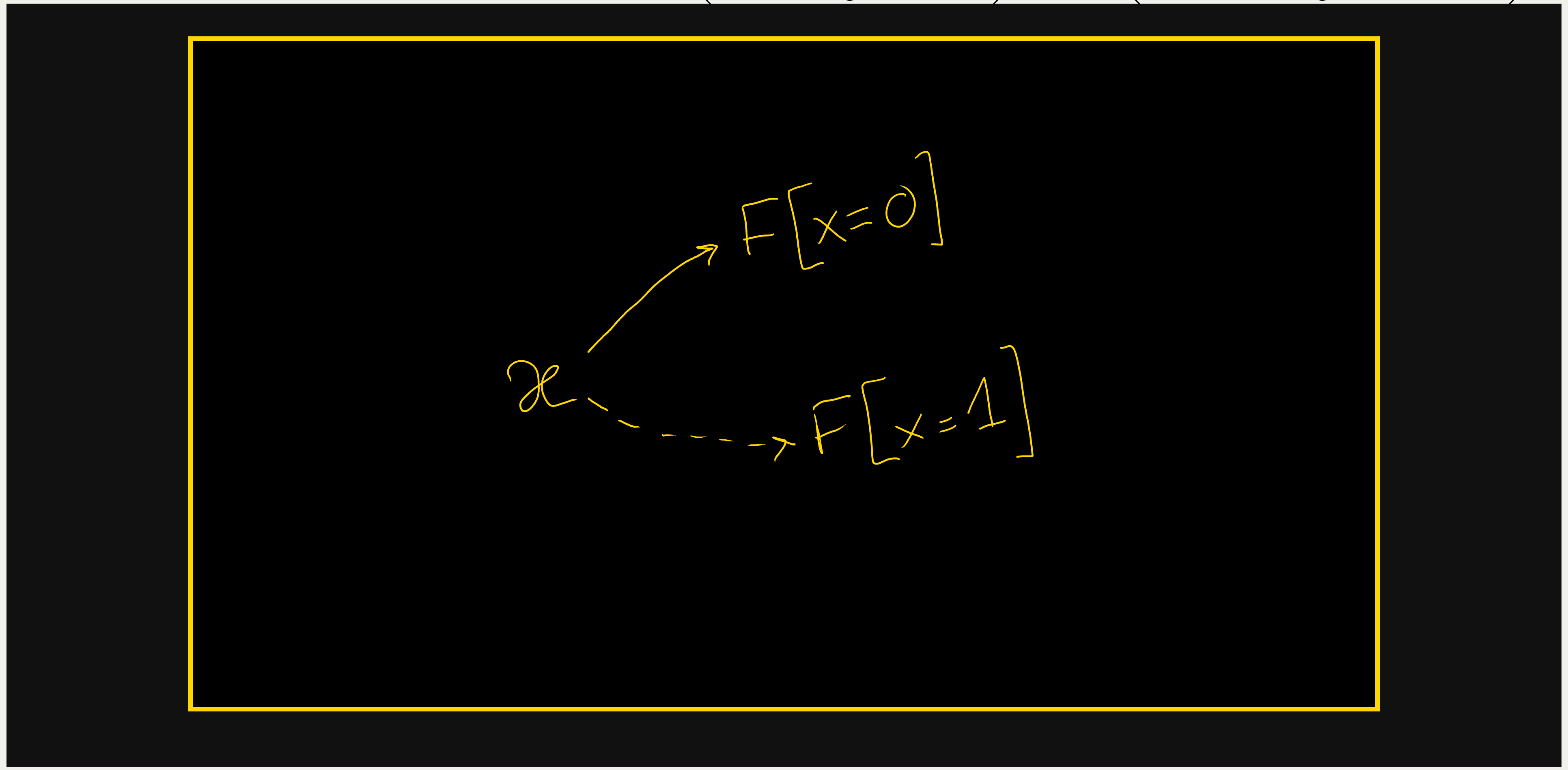
$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$



A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$

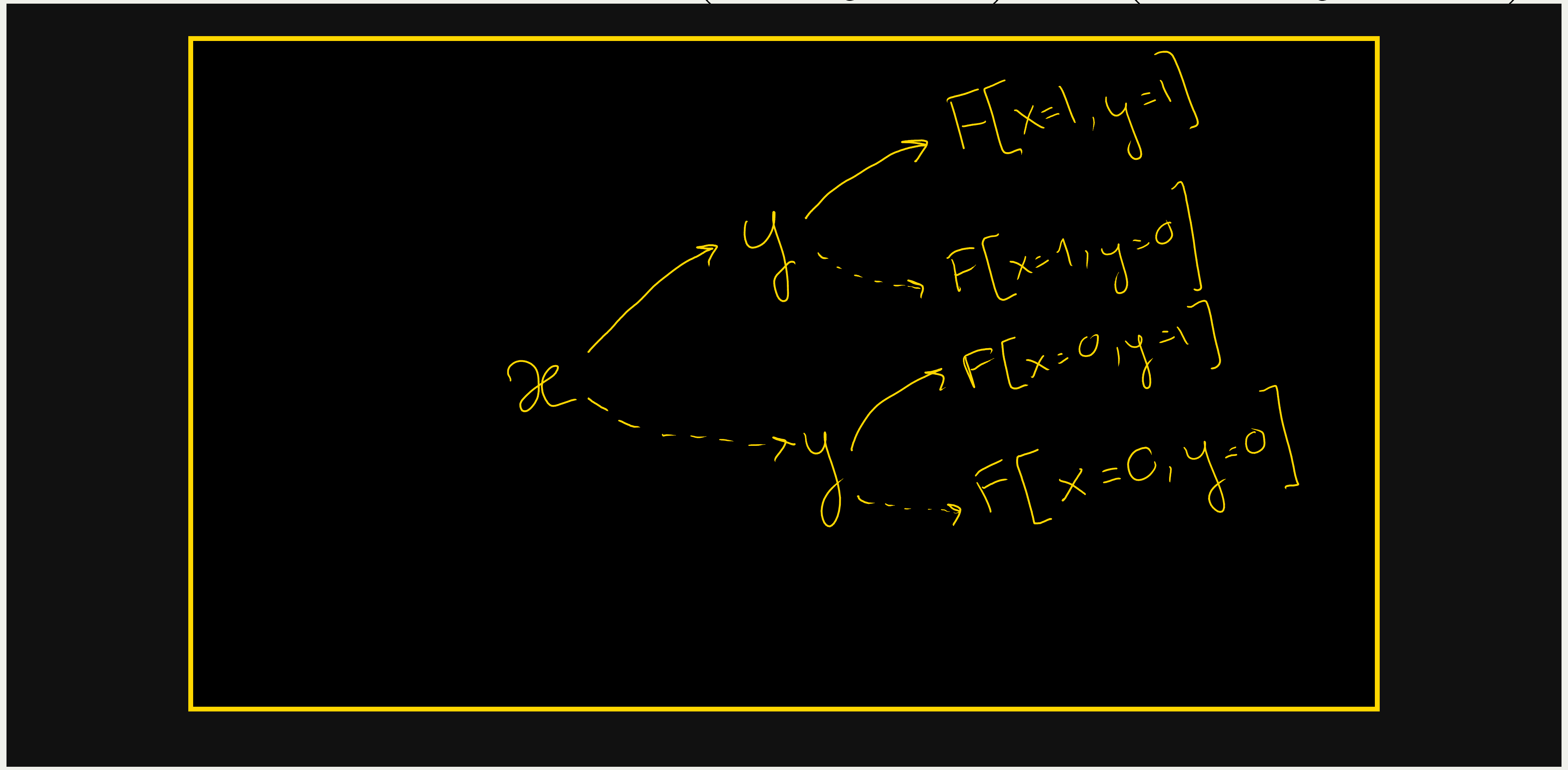


- $F[x=0] = (y \vee z) \wedge (\neg y \vee \neg z)$
- $F[x=1] = (\neg y \vee \neg z) \wedge (y \vee z)$

A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$

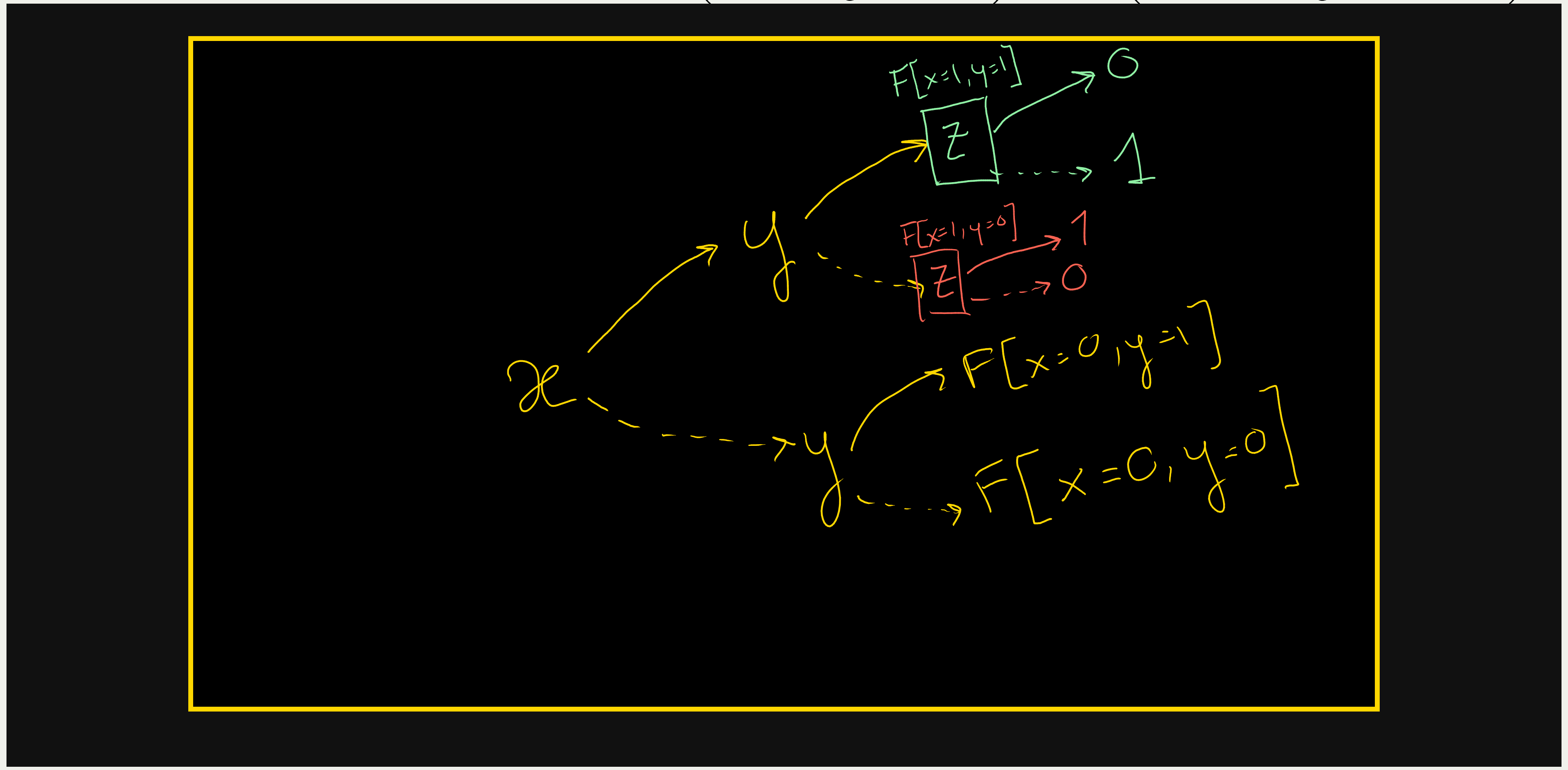


- $F[x=0] = (y \vee z) \wedge (\neg y \vee \neg z)$
- $F[x=1] = (\neg y \vee \neg z) \wedge (y \vee z)$
- $F[x=1, y=1] = \neg z$
- $F[x=1, y=0] = z$
-
-

A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$

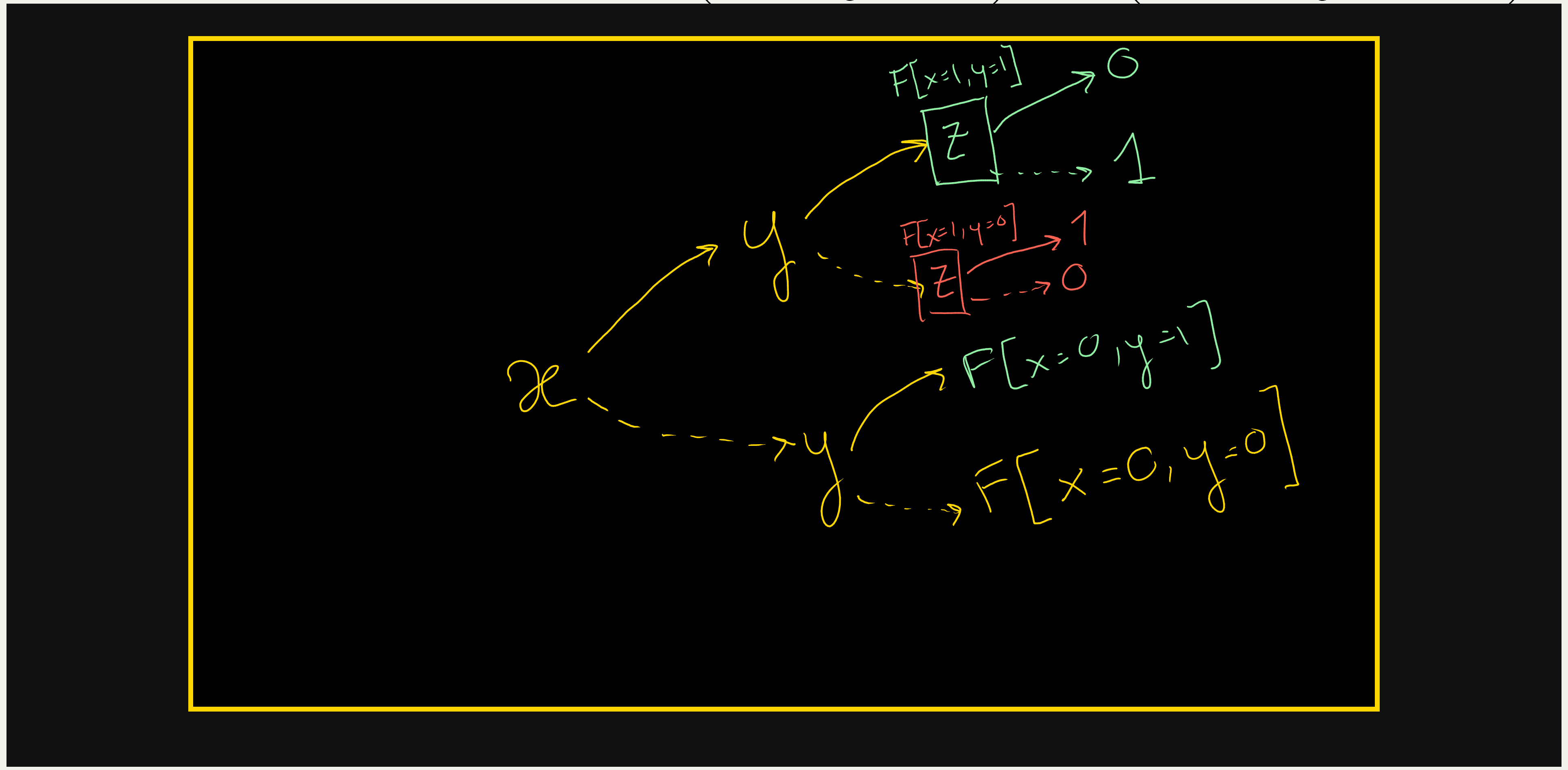


- $F[x=0] = (y \vee z) \wedge (\neg y \vee \neg z)$
- $F[x=1] = (\neg y \vee \neg z) \wedge (y \vee z)$
- $F[x=1, y=1] = \neg z$
- $F[x=1, y=0] = z$
-
-

A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$

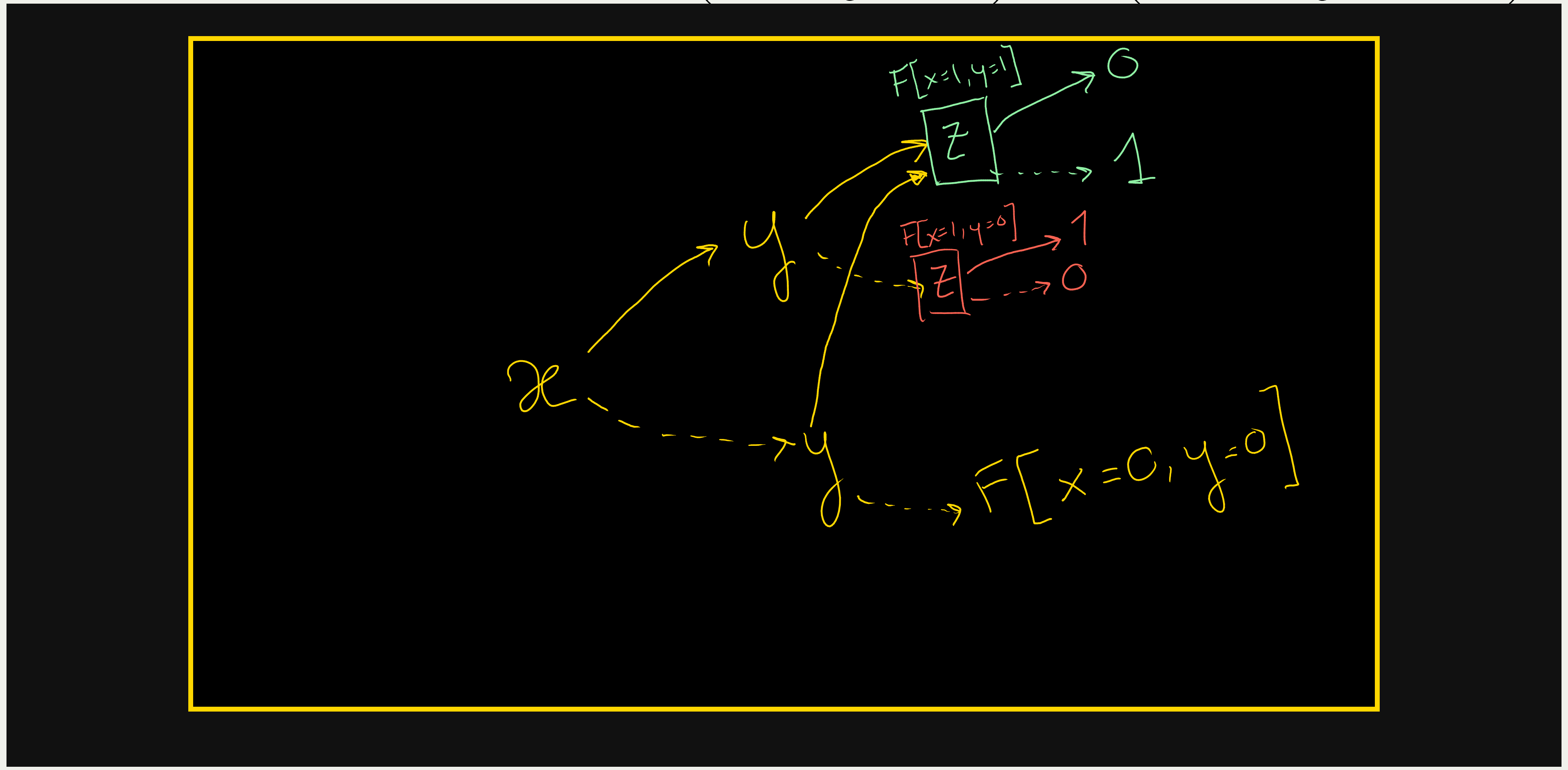


- $F[\,x = 0\,] = (\,y \vee z\,) \wedge (\,\neg y \vee \neg z\,)$
- $F[\,x = 1\,] = (\,\neg y \vee \neg z\,) \wedge (\,y \vee z\,)$
- $F[\,x = 1, y = 1\,] = \neg z$
- $F[\,x = 1, y = 0\,] = z$
- $F[\,x = 0, y = 1\,] = \neg z$
  $= F[\,x = 1, y = 1\,]$
-

A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$

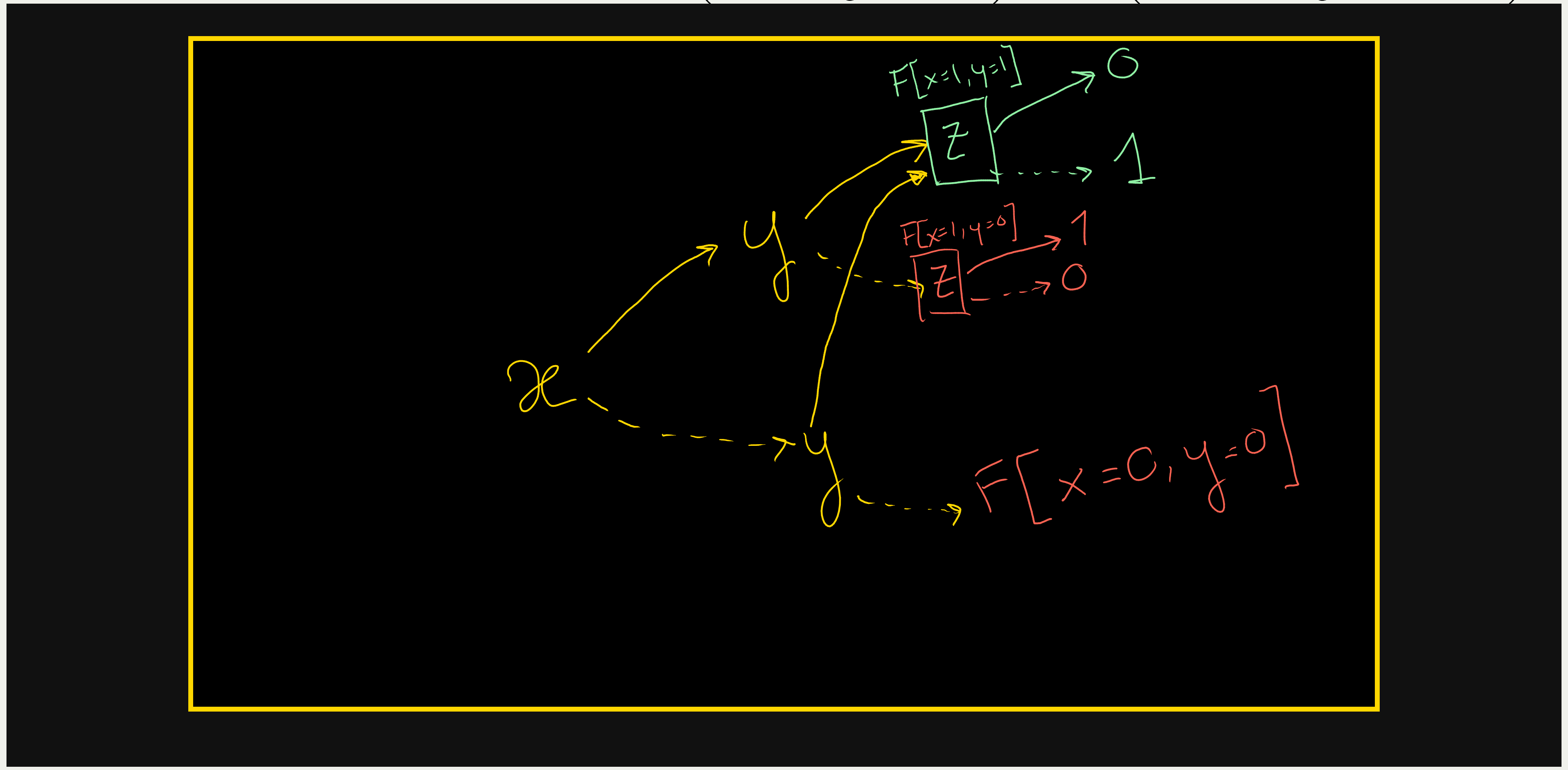


- $F[x=0] = (y \vee z) \wedge (\neg y \vee \neg z)$
- $F[x=1] = (\neg y \vee \neg z) \wedge (y \vee z)$
- $F[x=1, y=1] = \neg z$
- $F[x=1, y=0] = z$
- $F[x=0, y=1] = \neg z$
  $= F[x=1, y=1]$
-

A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$

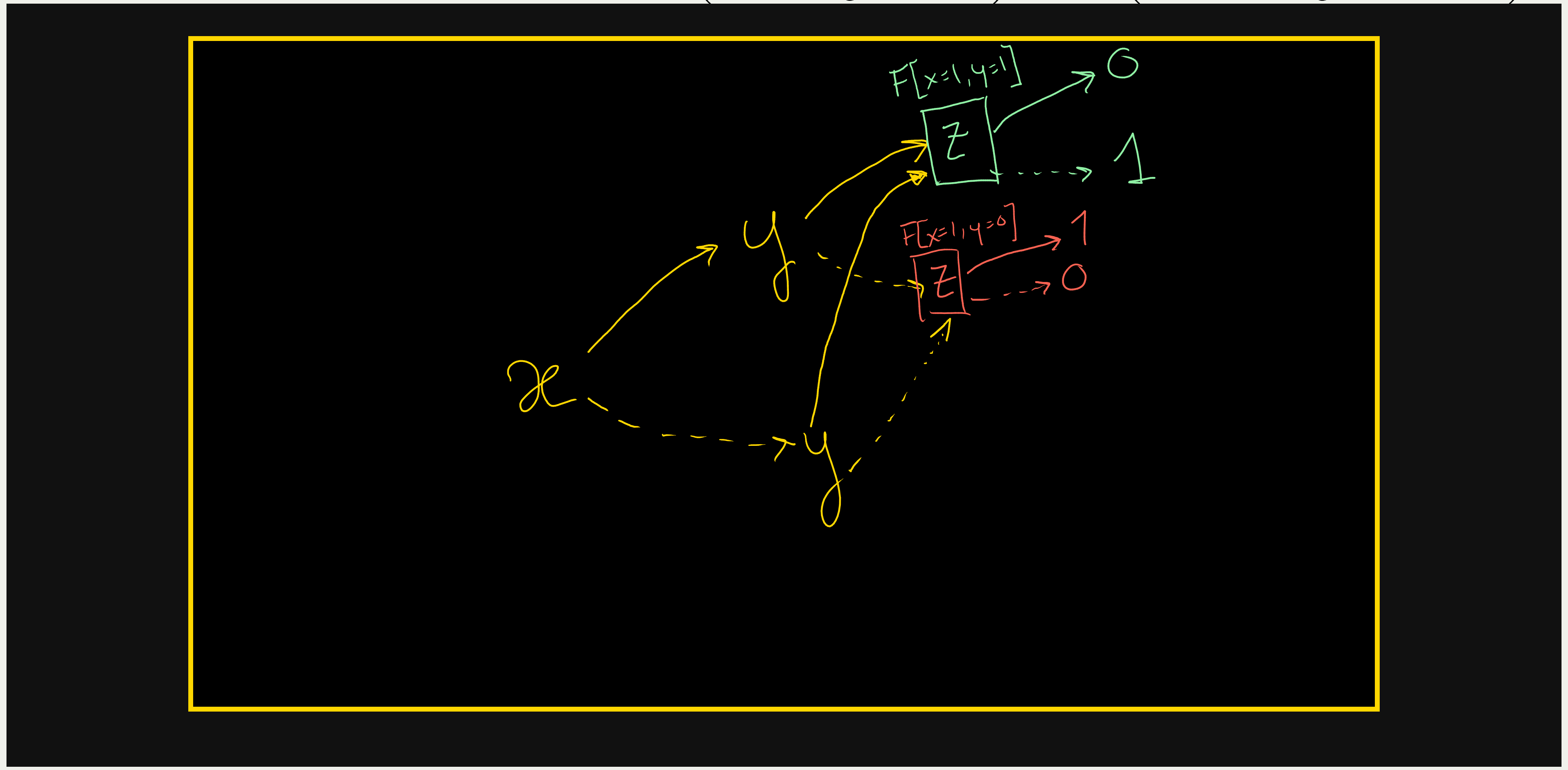


- $F[\,x = 0\,] = (\,y \vee z\,) \wedge (\,\neg y \vee \neg z\,)$
- $F[\,x = 1\,] = (\,\neg y \vee \neg z\,) \wedge (\,y \vee z\,)$
- $F[\,x = 1, y = 1\,] = \neg z$
- $F[\,x = 1, y = 0\,] = z$
- $F[\,x = 0, y = 1\,] = \neg z$
  $= F[\,x = 1, y = 1\,]$
- $F[\,x = 0, y = 0\,] = z = F[\,x = 1, y = 0\,]$

A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$

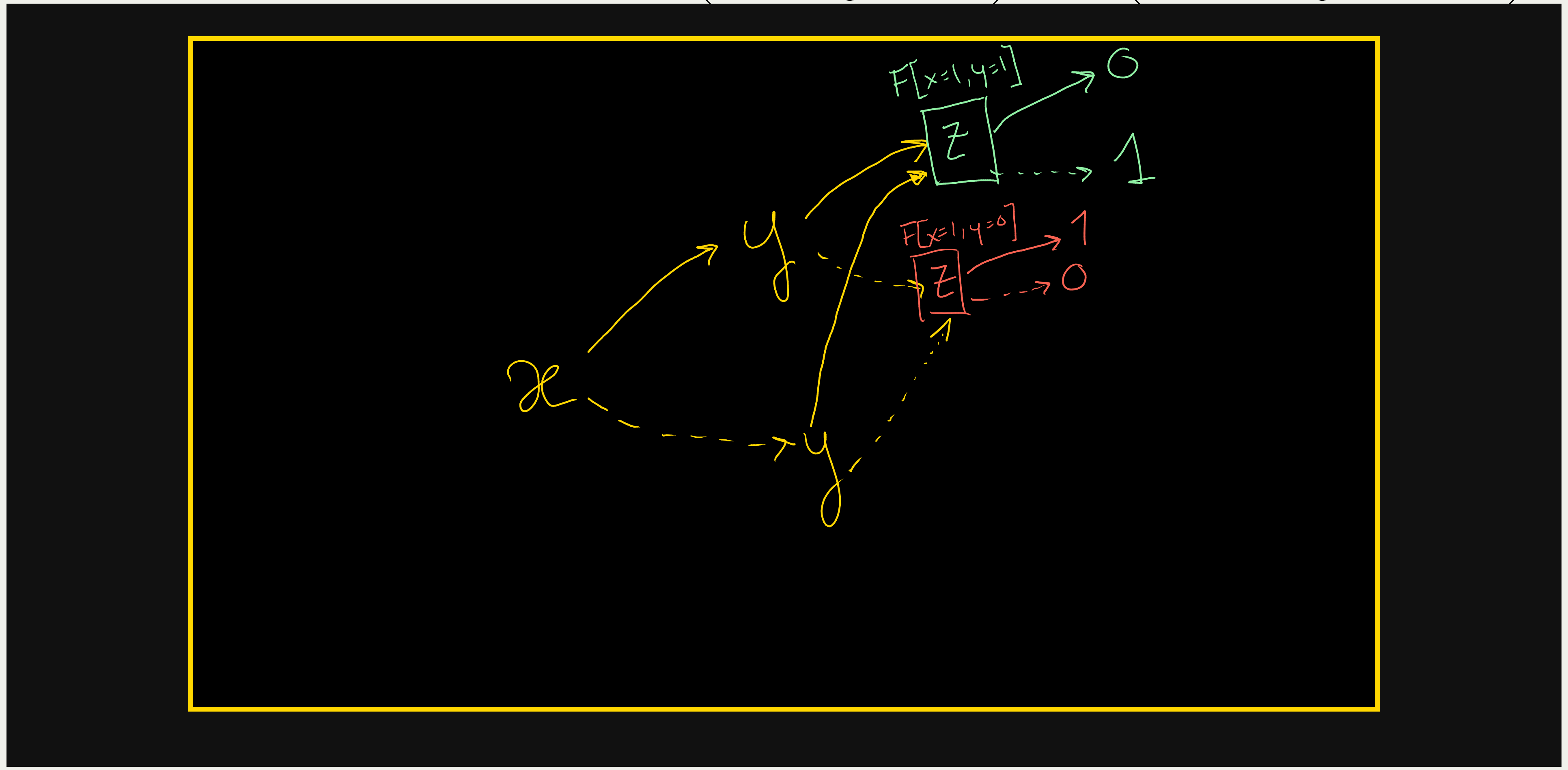


- $F[\,x = 0\,] = (\,y \vee z\,) \wedge (\,\neg y \vee \neg z\,)$
- $F[\,x = 1\,] = (\,\neg y \vee \neg z\,) \wedge (\,y \vee z\,)$
- $F[\,x = 1, y = 1\,] = \neg z$
- $F[\,x = 1, y = 0\,] = z$
- $F[\,x = 0, y = 1\,] = \neg z$
  $= F[\,x = 1, y = 1\,]$
- $F[\,x = 0, y = 0\,] = z = F[\,x = 1, y = 0\,]$

A Knowledge Compiler for OBDD

Exhaustive DPLL with Caching based on **Shannon Expansion**:

$$F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z)$$



- $F[x=0] = (y \vee z) \wedge (\neg y \vee \neg z)$
- $F[x=1] = (\neg y \vee \neg z) \wedge (y \vee z)$
- $F[x=1, y=1] = \neg z$
- $F[x=1, y=0] = z$
- $F[x=0, y=1] = \neg z$
  $= F[x=1, y=1]$
- $F[x=0, y=0] = z = F[x=1, y=0]$

This scheme is parameterized by:

- caching policy
- branching heuristics

Exploiting decomposition

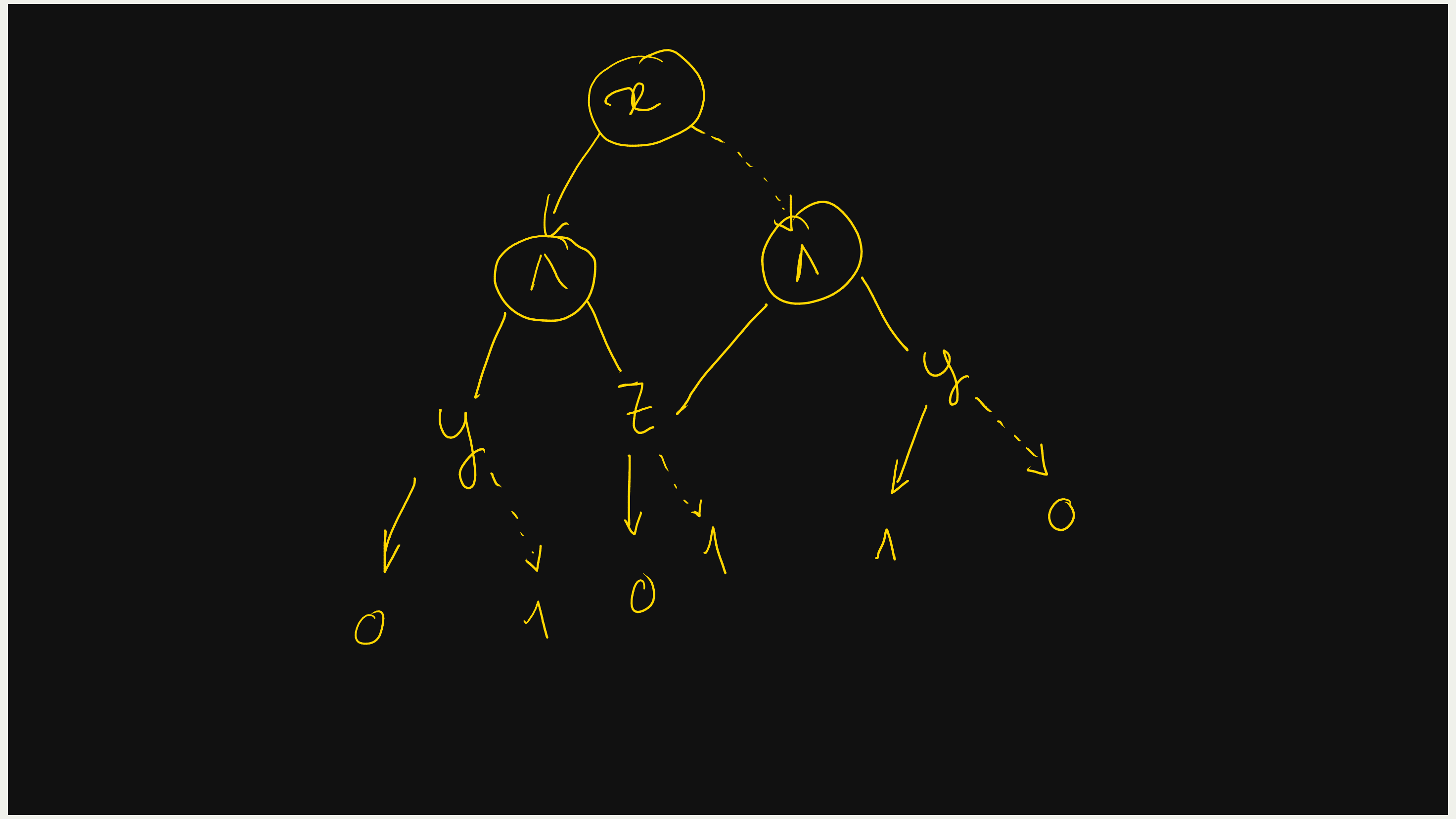
For many tasks, such as model counting, it is interesting to detect **syntactic decomposable** part of the formula, that is:

$$\boxed{F(X) = G(Y) \land H(Z) \ \textbf{and}\ Y \cap Z = \emptyset}$$

Exploiting decomposition

For many tasks, such as model counting, it is interesting to detect **syntactic decomposable** part of the formula, that is:

$$\boxed{F(X) = G(Y) \land H(Z) \textbf{ and } Y \cap Z = \emptyset}$$



- **decDNNF**: OBDD + $\land$-gates *decomposable*
- Still allows for algebraic model counting via the identity $w(F) = w(G) \times w(H)$
- Compilers can be adapted to detect this rule.

The D4 compiler

`D4` is a top-down compiler as shown earlier:

- Use oracle calls to a SAT solver **with clause learning** to cut branches and speed up later computation
- Use heuristics to decompose the formula so that it **breaks** into smaller connected components.

The D4 compiler

D4 is a top-down compiler as shown earlier:

- Use oracle calls to a SAT solver **with clause learning** to cut branches and speed up later computation
- Use heuristics to decompose the formula so that it **breaks** into smaller connected components.

instance	**d4 (s)**	**scip (s)**
bernasconi.20.3	0.002	0.01
bernasconi.20.5	0.04	8.91
bernasconi.20.10	1.21	119.20
bernasconi.20.15	14.92	479.15
bernasconi.25.3	0.00	0.01
bernasconi.25.6	0.19	151.65
bernasconi.25.13	12.59	1 698.18
bernasconi.25.19	442.26	TIMEOUT
bernasconi.25.25	TIMEOUT	TIMEOUT

This only illustrates that the underlying structure of Bernasconi instances is better addressed using heuristics from model counting than the ILP approach.

Tractability results

Tractable classes of BPO

$$\boxed{P(x_1, \ldots, x_n) = \sum_{e \in E} \alpha_e \prod_{i \in e} x_i \text{ where } E \subseteq 2^V}$$

$H = (V, E)$ is a **hypergraph**.

Tractable classes of BPO

$$P(x_1, \ldots, x_n) = \sum_{e \in E} \alpha_e \prod_{i \in e} x_i \text{ where } E \subseteq 2^V$$

$H = (V, E)$ is a **hypergraph**.

Exploit the structure of H to solve BPO more efficiently.

A toy example

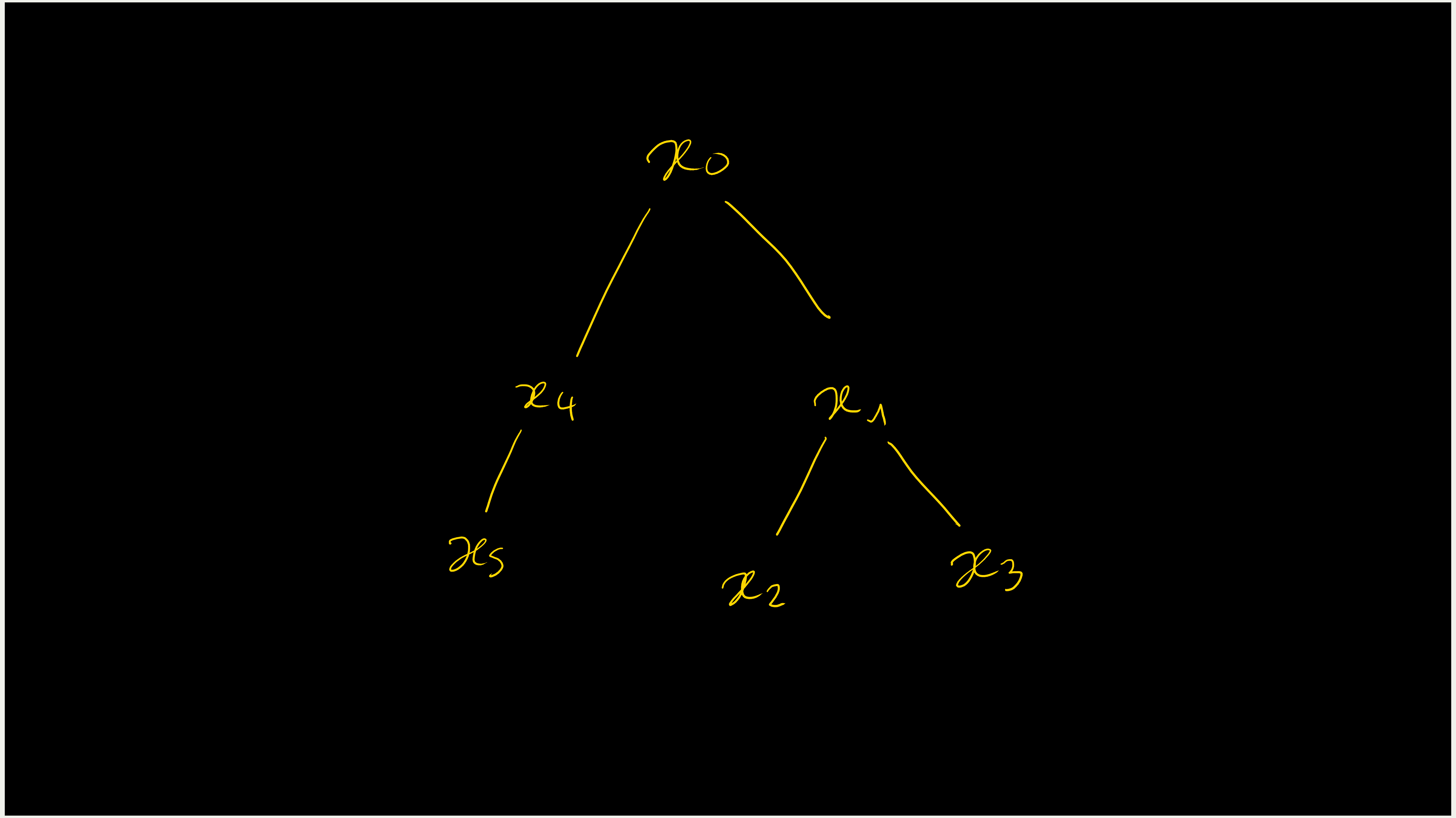
Tree BPO: BPO problem where H is a tree.

Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$

A toy example

Tree BPO: BPO problem where H is a tree.

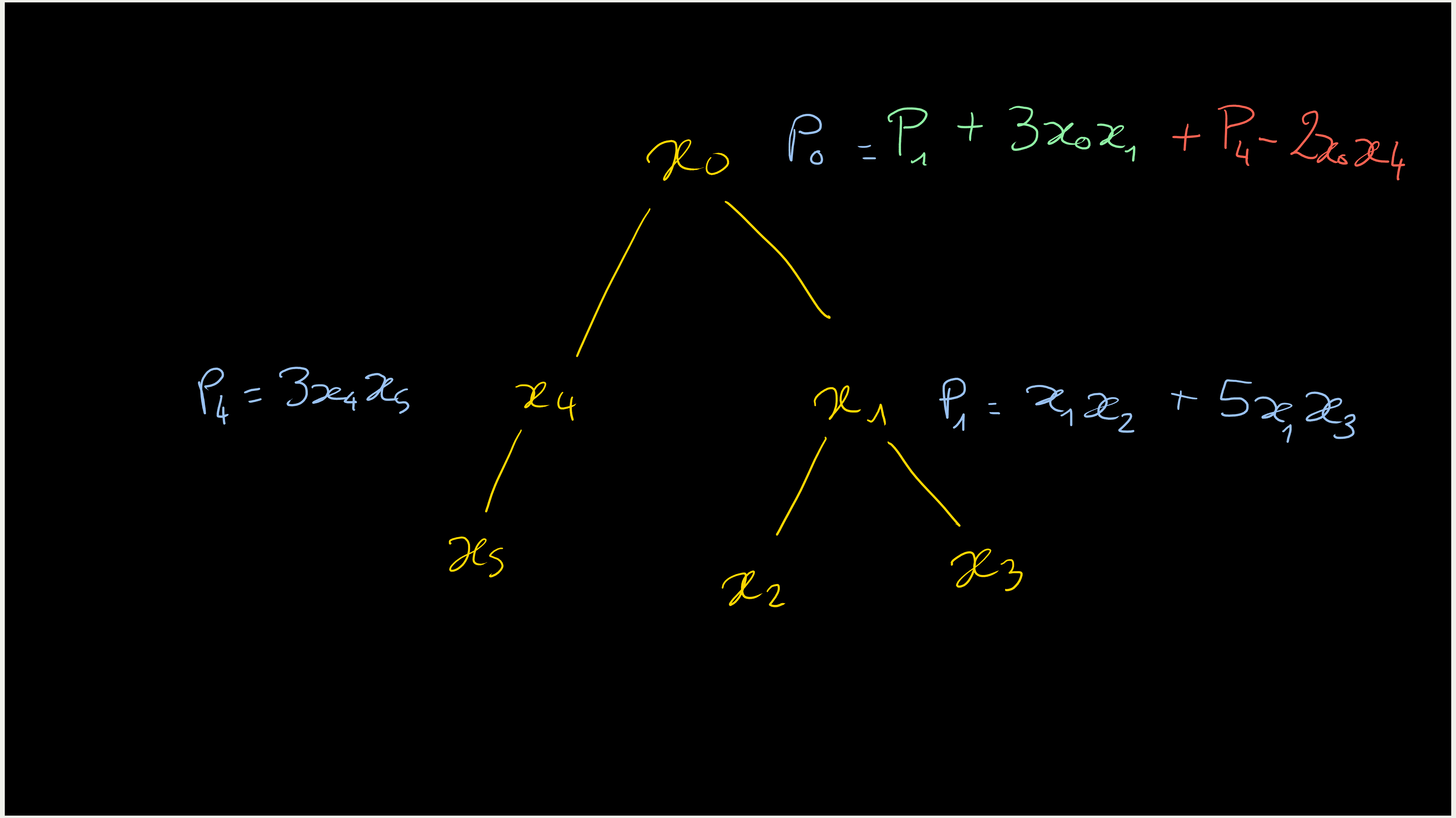
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

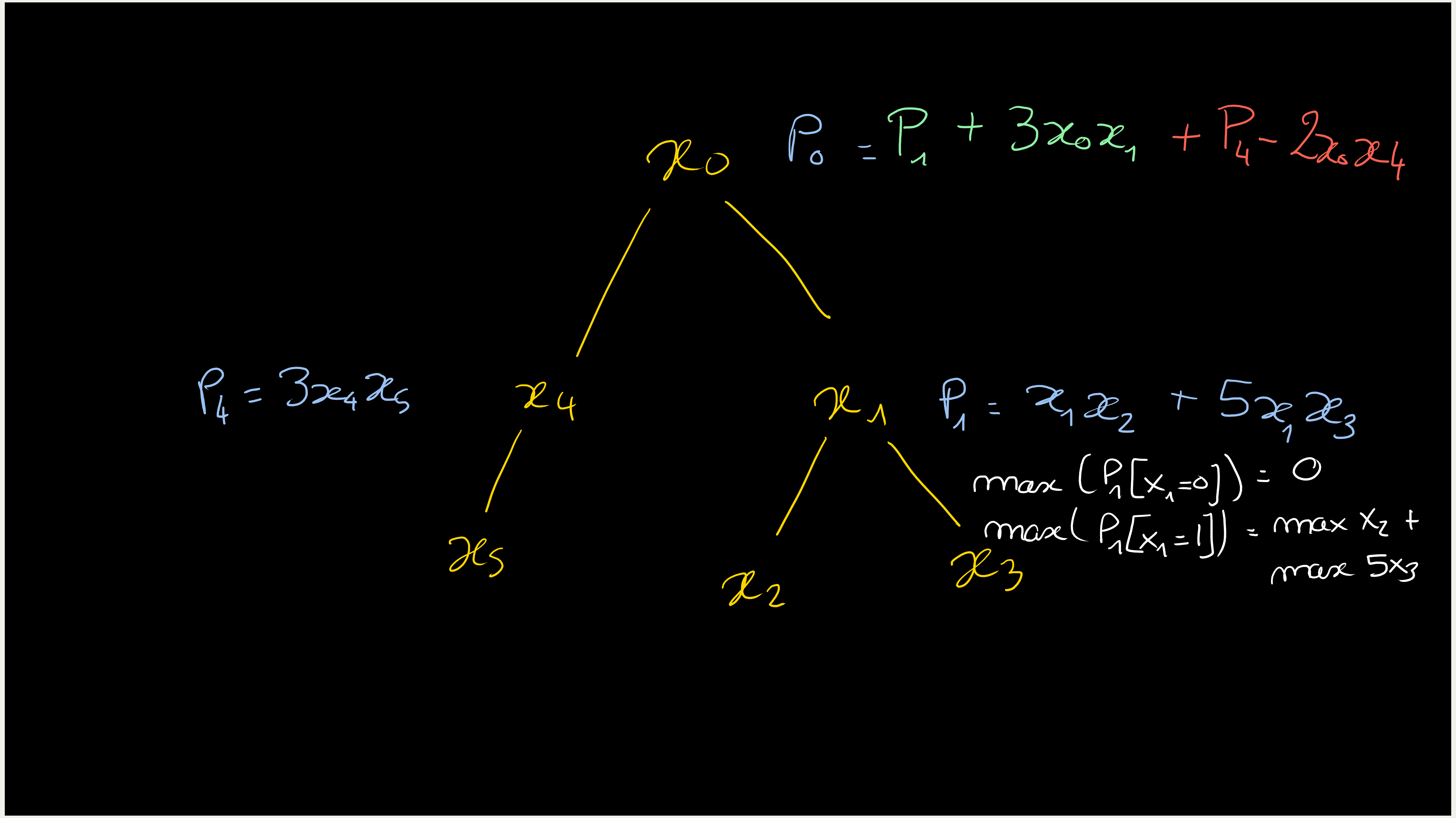
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

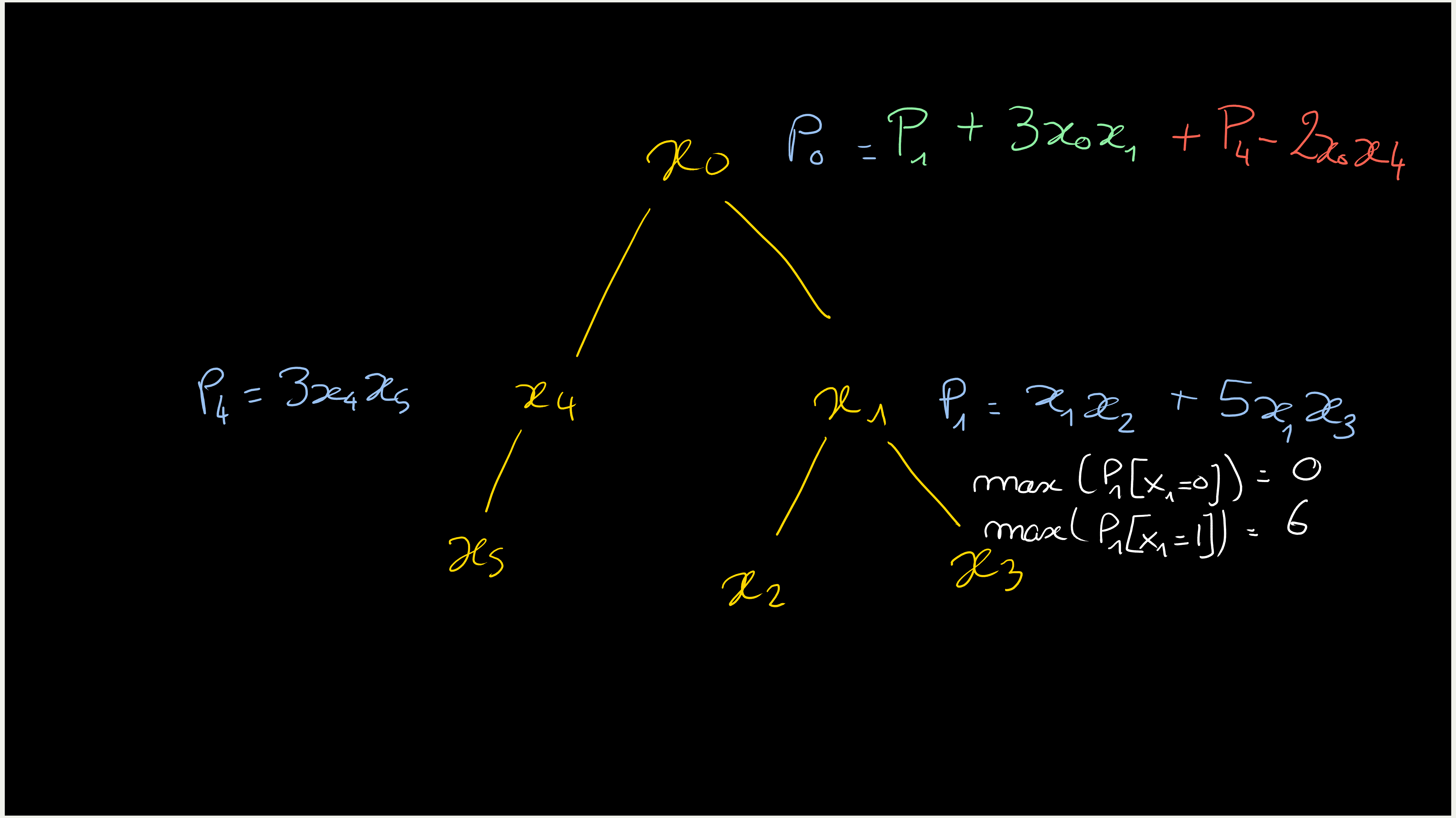
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

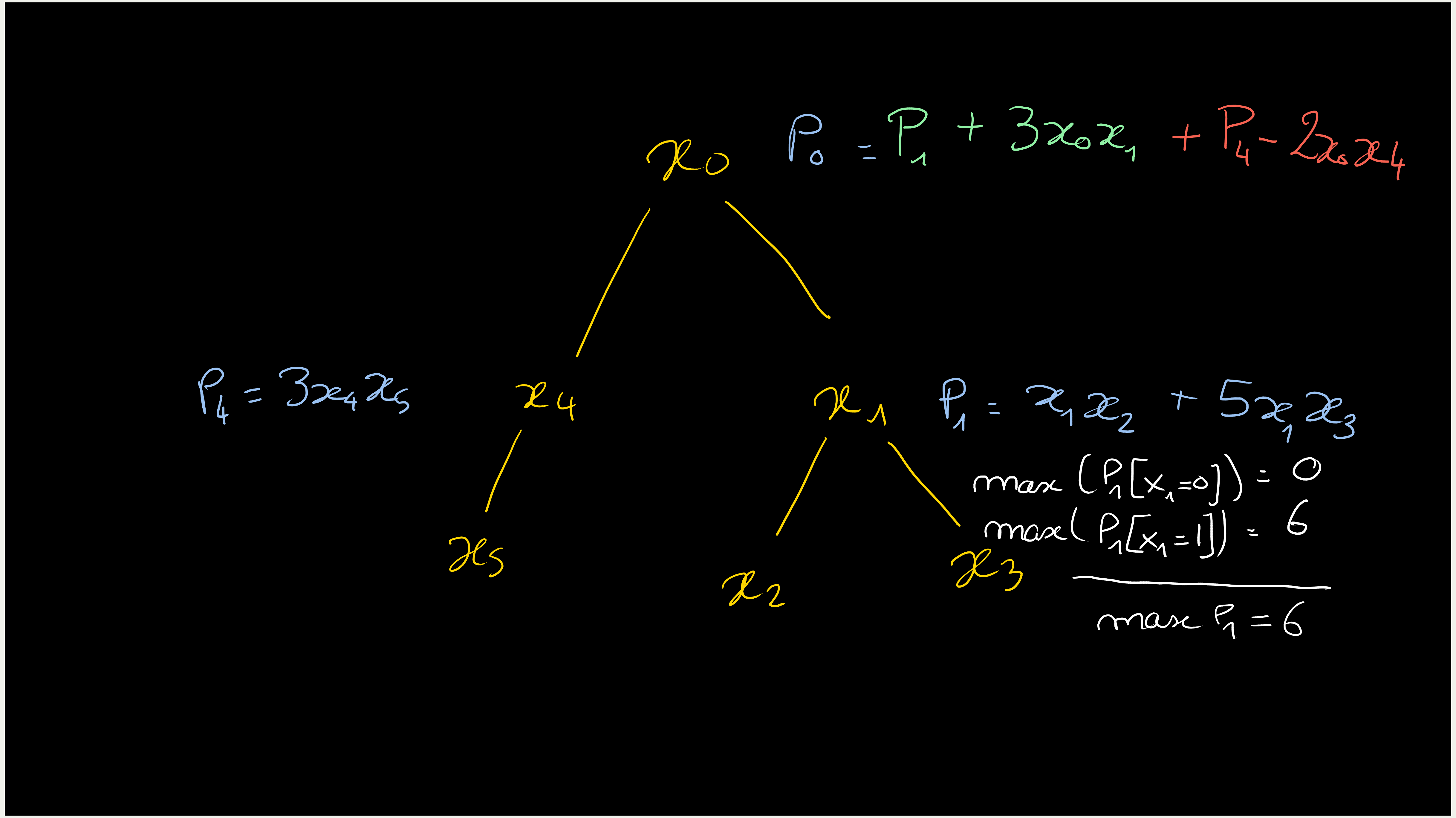
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

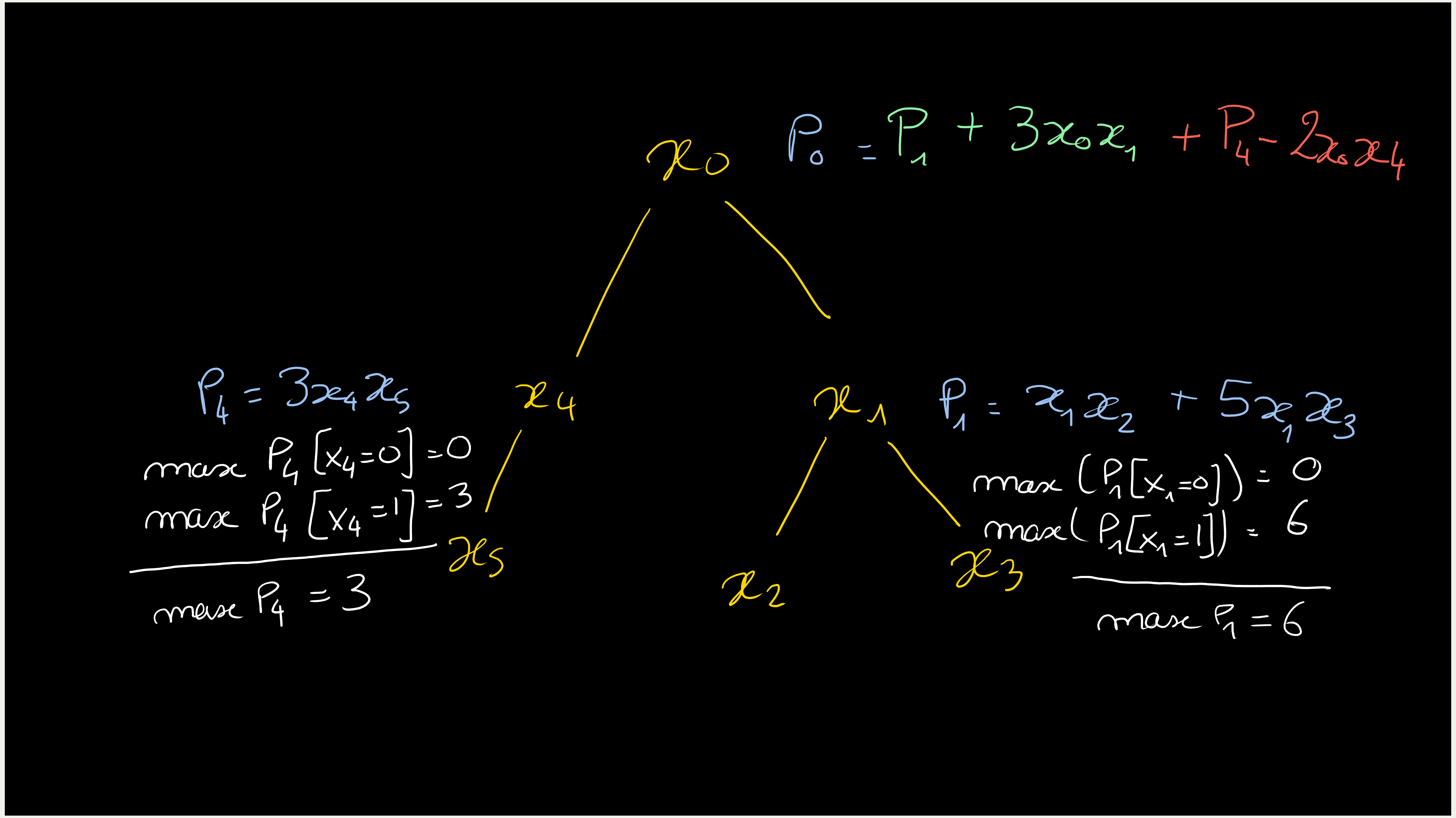
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

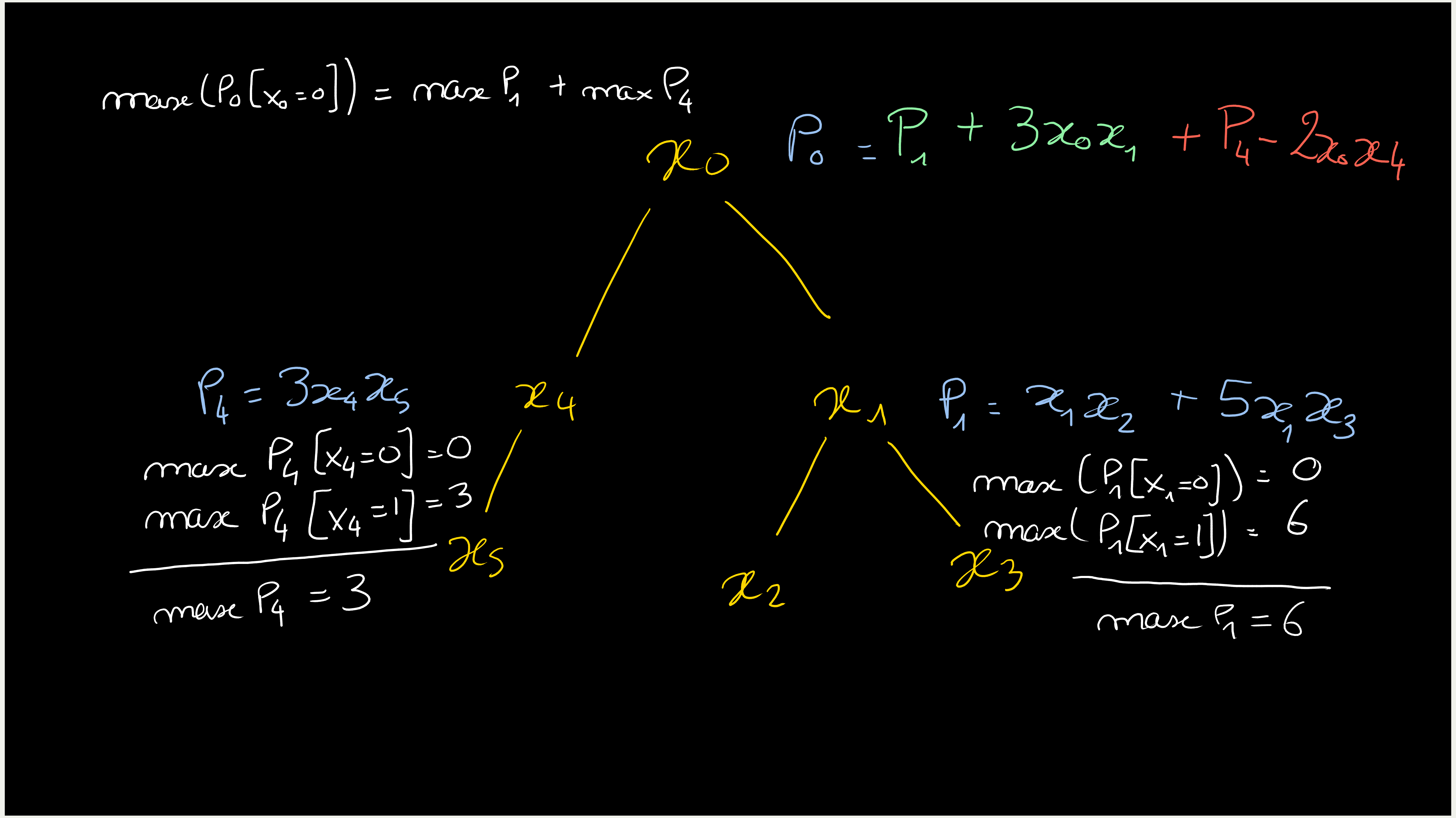
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

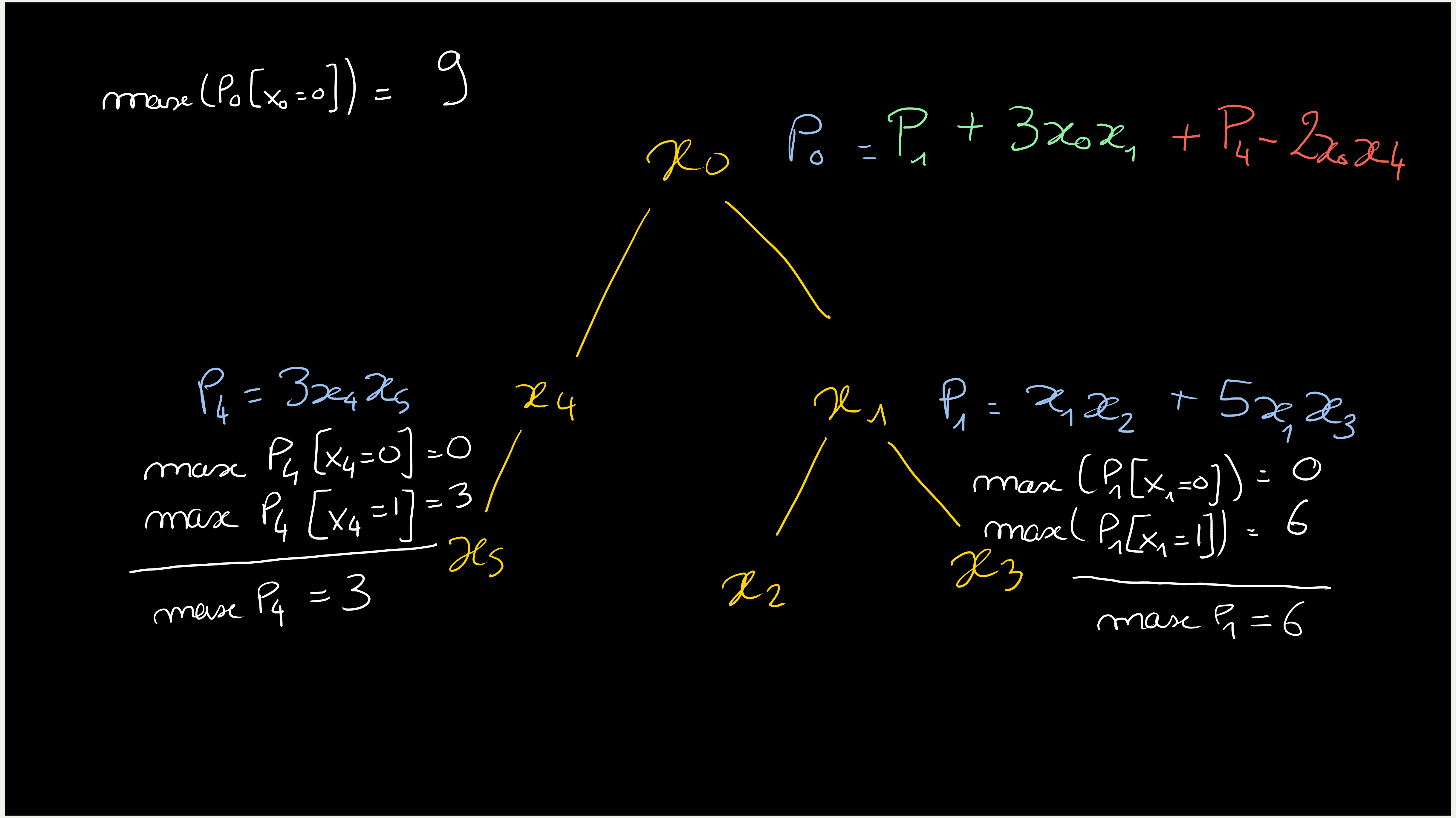
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

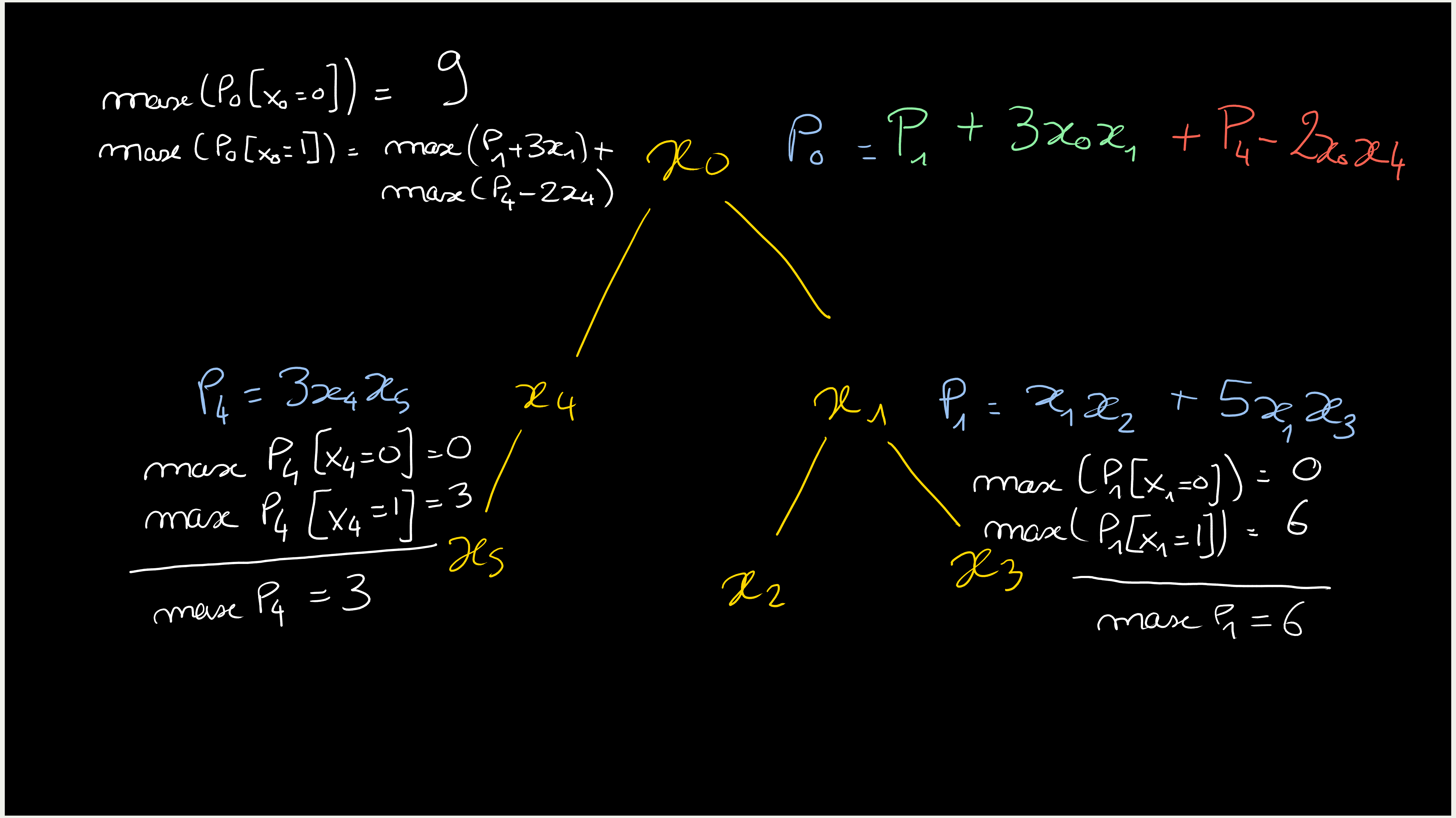
Example: $x_1 x_2 + 5x_1 x_3 + 3x_0 x_1 - 2x_0 x_4 + 3x_4 x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

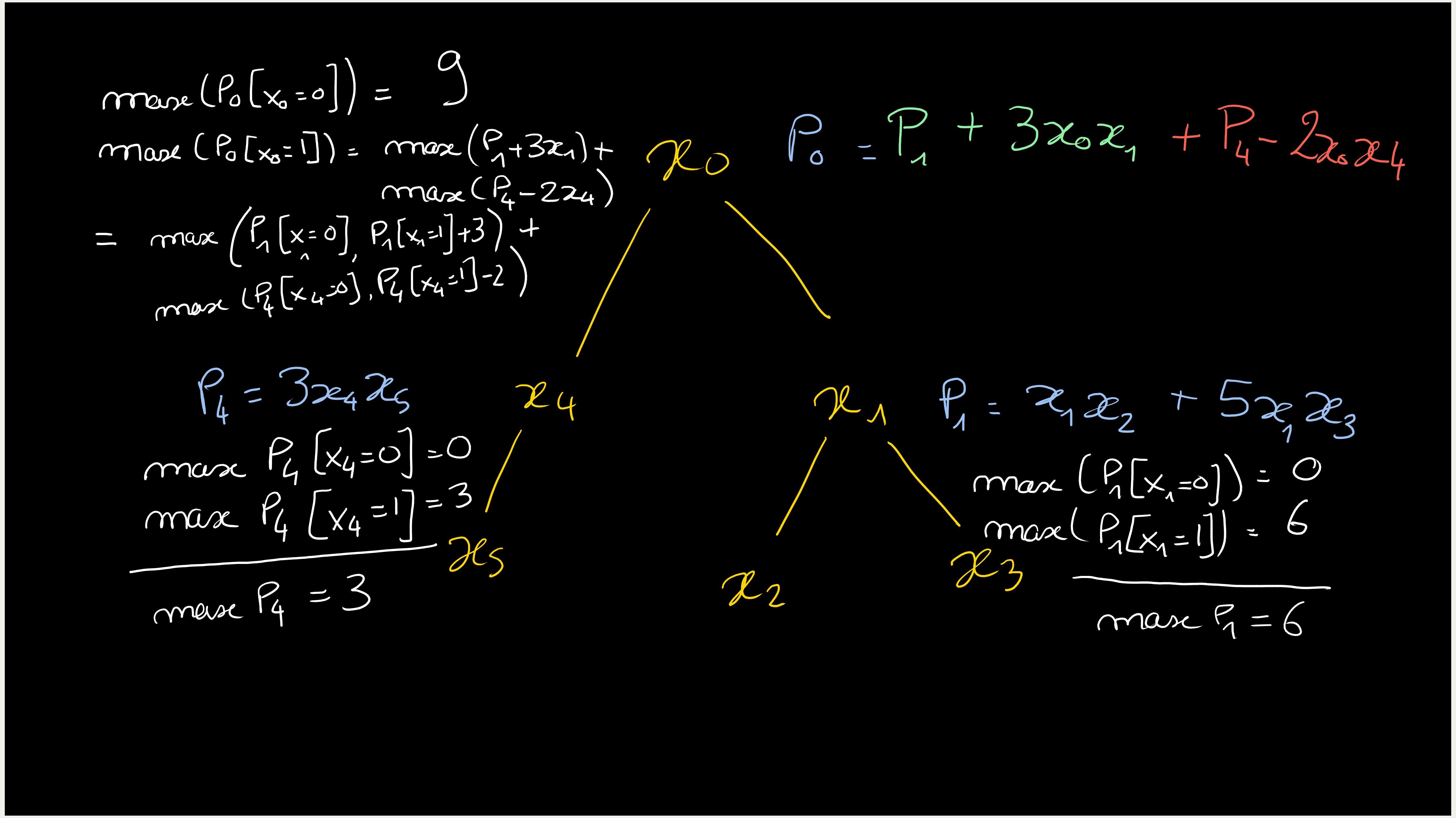
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

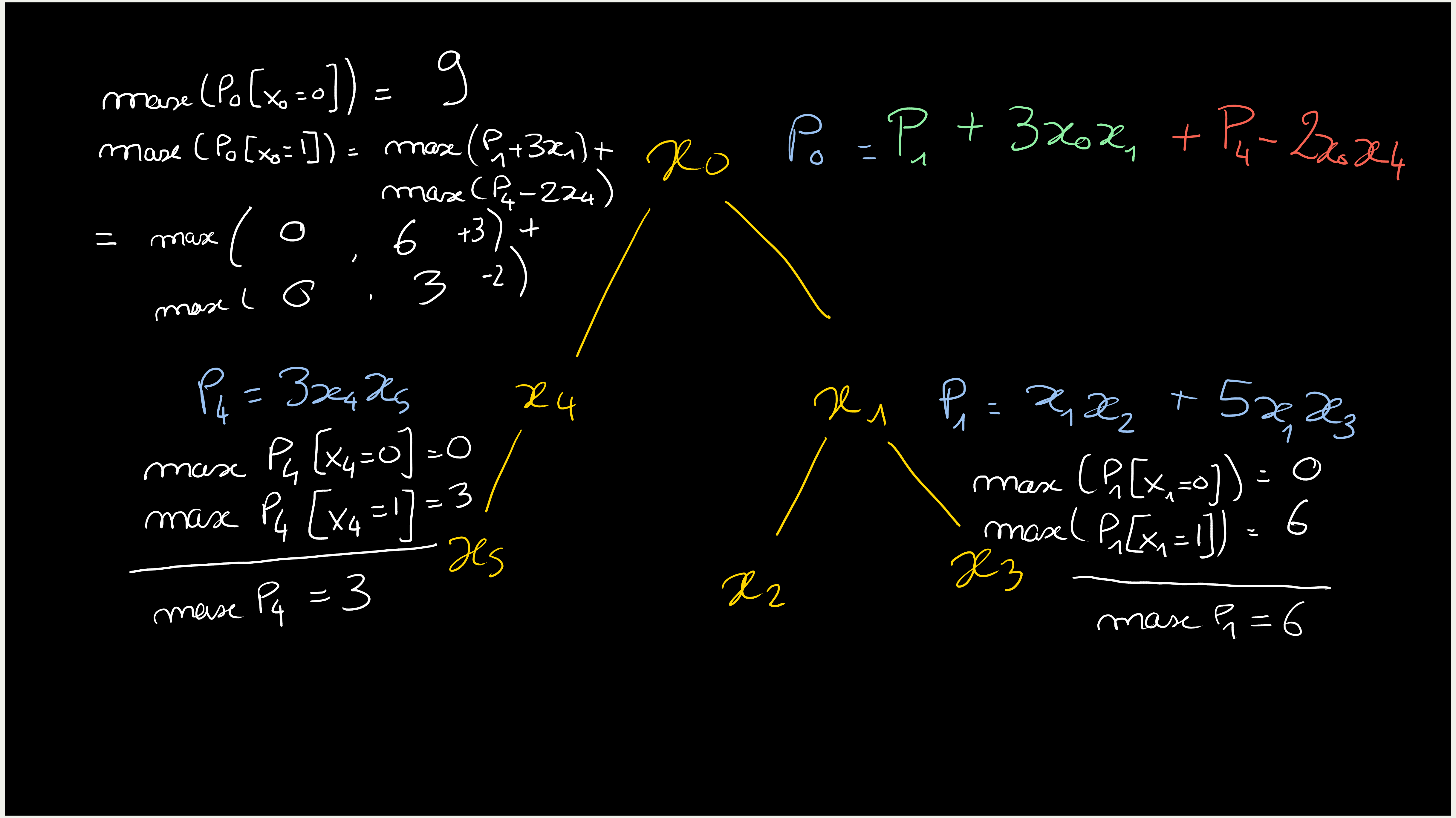
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

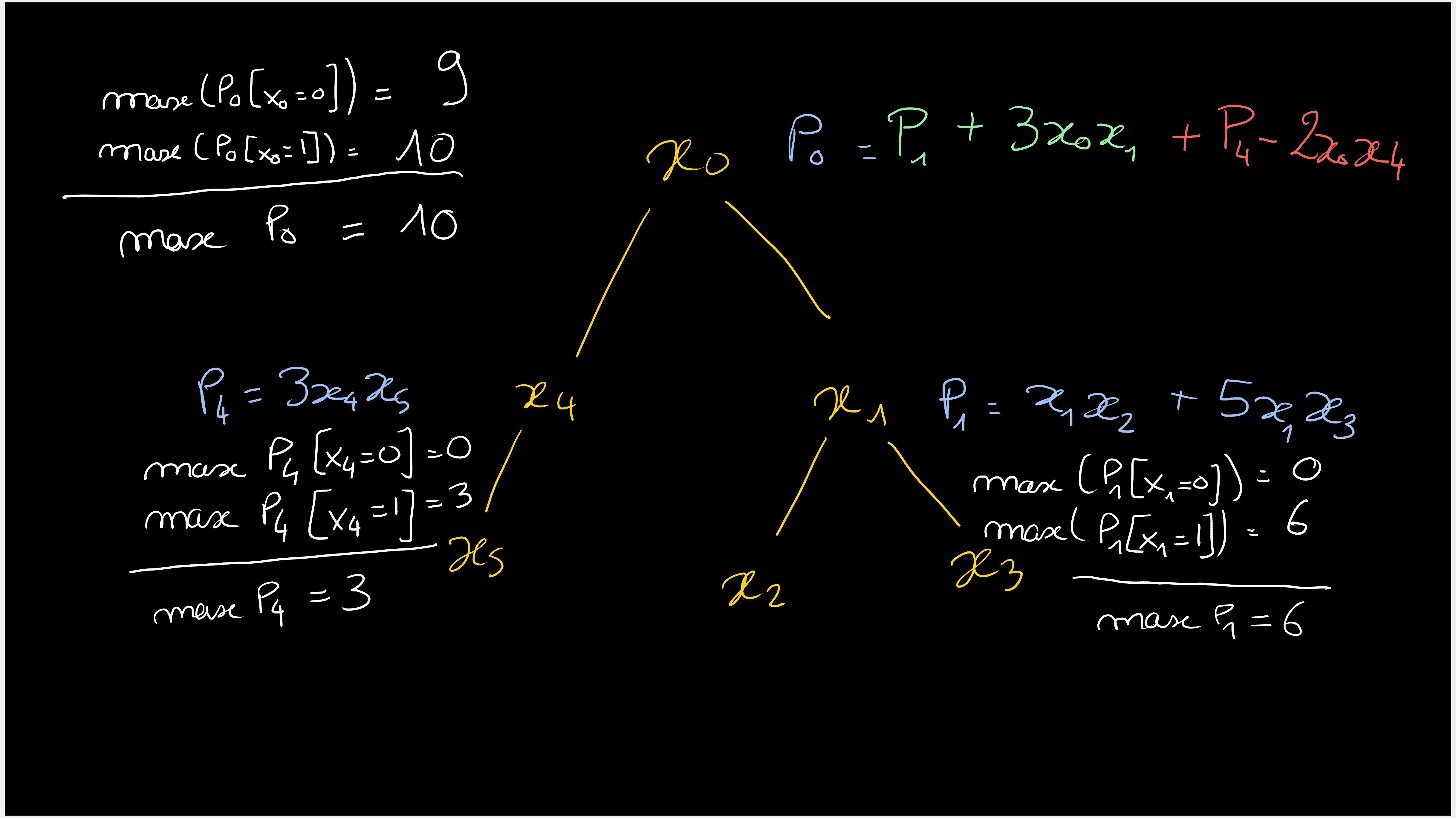
Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



A toy example

Tree BPO: BPO problem where H is a tree.

Example: $x_1x_2 + 5x_1x_3 + 3x_0x_1 - 2x_0x_4 + 3x_4x_5$



Many Known Tractable Classes

Theorem

- H has tree width k: BPO can be solved in time $2^{O(k)} poly(H)$.
- H is β-acyclic: BPO can be solved in time $poly(H)$.

Dedicated algorithm for each class.

A strange symmetry

Very similar results from Boolean function literature:

Theorem

- If a CNF F has tree width k then one can construct a DNNF for F of size $2^{O(k)} poly(F)$.
- If a CNF F is β-acyclic then one can construct a DNNF for F of size $poly(F)$.

Is there a connection?

Encoding BPO as a CNF

For $P := \sum_{e \in E} \alpha_e \prod_{i \in e} x_i$ define:

$$f_P := \bigwedge_{e \in E} C_e$$

where $C_e := Y_e \Leftrightarrow \bigwedge_{i \in e} X_i$

C_e can be encoded as the conjunction of:

- $\bigvee_{i \in e} \neg X_i \vee Y_e$
- $\neg Y_e \vee X_i$ for every $i \in e$

Encoding BPO as a CNF

For $P := \sum_{e \in E} \alpha_e \prod_{i \in e} x_i$ define:

$$f_P := \bigwedge_{e \in E} C_e$$

where $C_e := Y_e \Leftrightarrow \bigwedge_{i \in e} X_i$

C_e can be encoded as the conjunction of:

- $\bigvee_{i \in e} \neg X_i \vee Y_e$
- $\neg Y_e \vee X_i$ for every $i \in e$

f_P is naturally encoded as a CNF F_P that preserves tree width.

Tractability of BPO via KC

Every known tractability for BPO can be recovered in our framework as follows:

1. Encode P as a CNF formula F_P
2. Transform F_P into a polynomial size tractable representation C_P **using known results**
3. Solve AMC on C_P

And we get new tractability results for structure that where not known to make BPO tractable.

Beyond BPO

KC approach very versatile:

Beyond BPO

KC approach very versatile:

- Solve top-k BPO: find the k best solutions of P by **finding the k best in the circuit**

Beyond BPO

KC approach very versatile:

- Solve top-k BPO: find the k best solutions of P by **finding the k best in the circuit**
- Solve BPO + Cardinality constraints: $\max P(x_1, \ldots, x_n)$ *such that* $\sum_{i=1}^{n} x_i \in S$ where $S \subseteq [n]$ by **transforming the circuit**

Beyond BPO

KC approach very versatile:

- Solve top-k BPO: find the k best solutions of P by **finding the k best in the circuit**
- Solve BPO + Cardinality constraints: $\max P(x_1, ..., x_n)$ *such that* $\sum_{i=1}^{n} x_i \in S$ where $S \subseteq [n]$ by **transforming the circuit**
- Solve pseudo BPO: P can contains monomial of the form $\prod_{i \in A} x_i \prod_{i \in B} (1 - x_i)$

Conclusion

Connection between BPO and Boolean functions:

- Recover known results and generalize them using the existing rich literature
- Seems to have practical relevance

Perspective:

KC only exploits combinatorics of the underlying Boolean function. How could we mix existing more algebraic techniques?