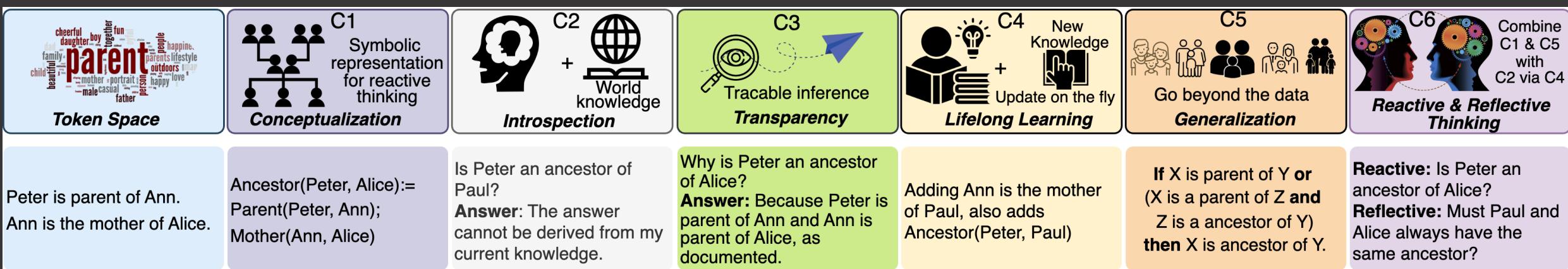


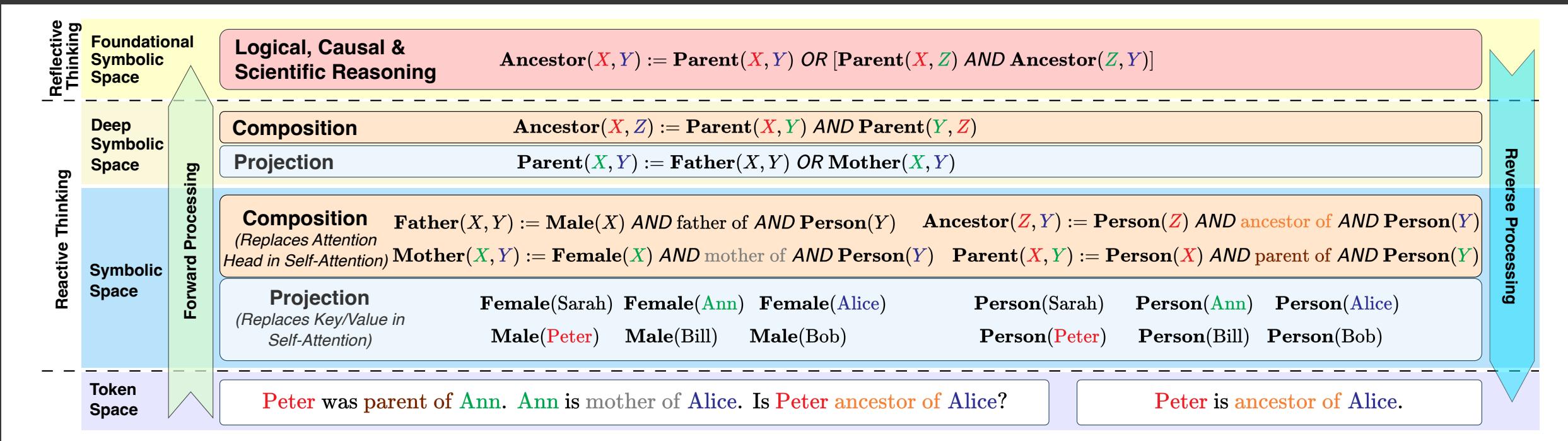
# Learning AND-OR hierarchies with Tsetlin machines

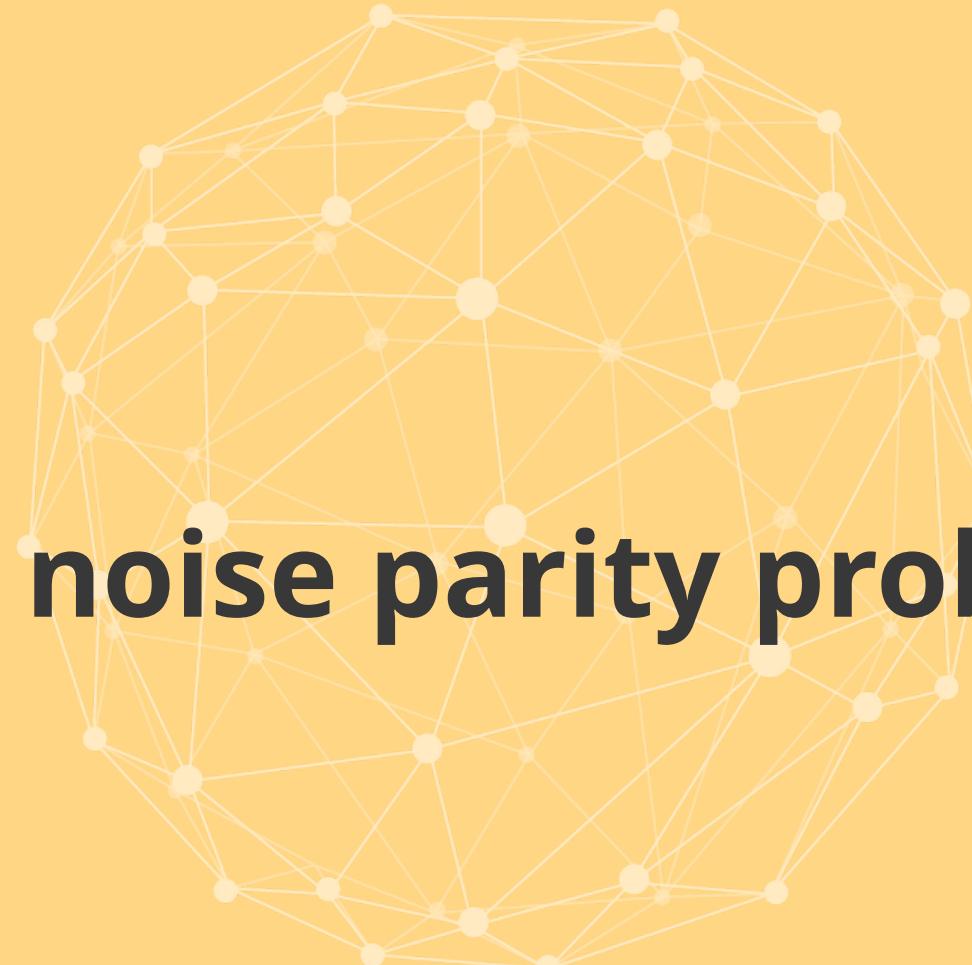
Ole-Christoffer Granmo  
February 6, 2026

# Bridging reactive and reflective thinking - An 8-year research agenda



# Bridging reactive and reflective thinking - An 8-year research agenda





# The noise parity problem

Odd number of bits set  $\rightarrow y = 1$

Even number of bits set  $\rightarrow y = 0$

**X**

**0 1 0 1 0 0 1 0 1 1 0 1**  
**1 1 0 0 1 0 1 0 1 0 1 1**

**Y**

0  
1



Erroneous label 20% of the time

8 informative bits (in bold)  $\rightarrow$  **256 unique patterns**

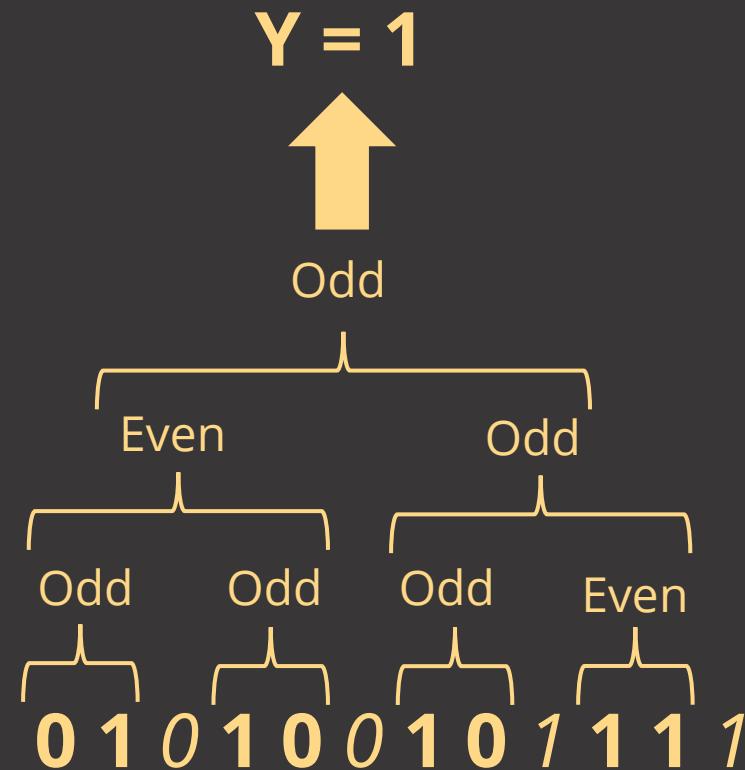
**X**

0 1 0 1 0 0 1 0 1 ? 0 1

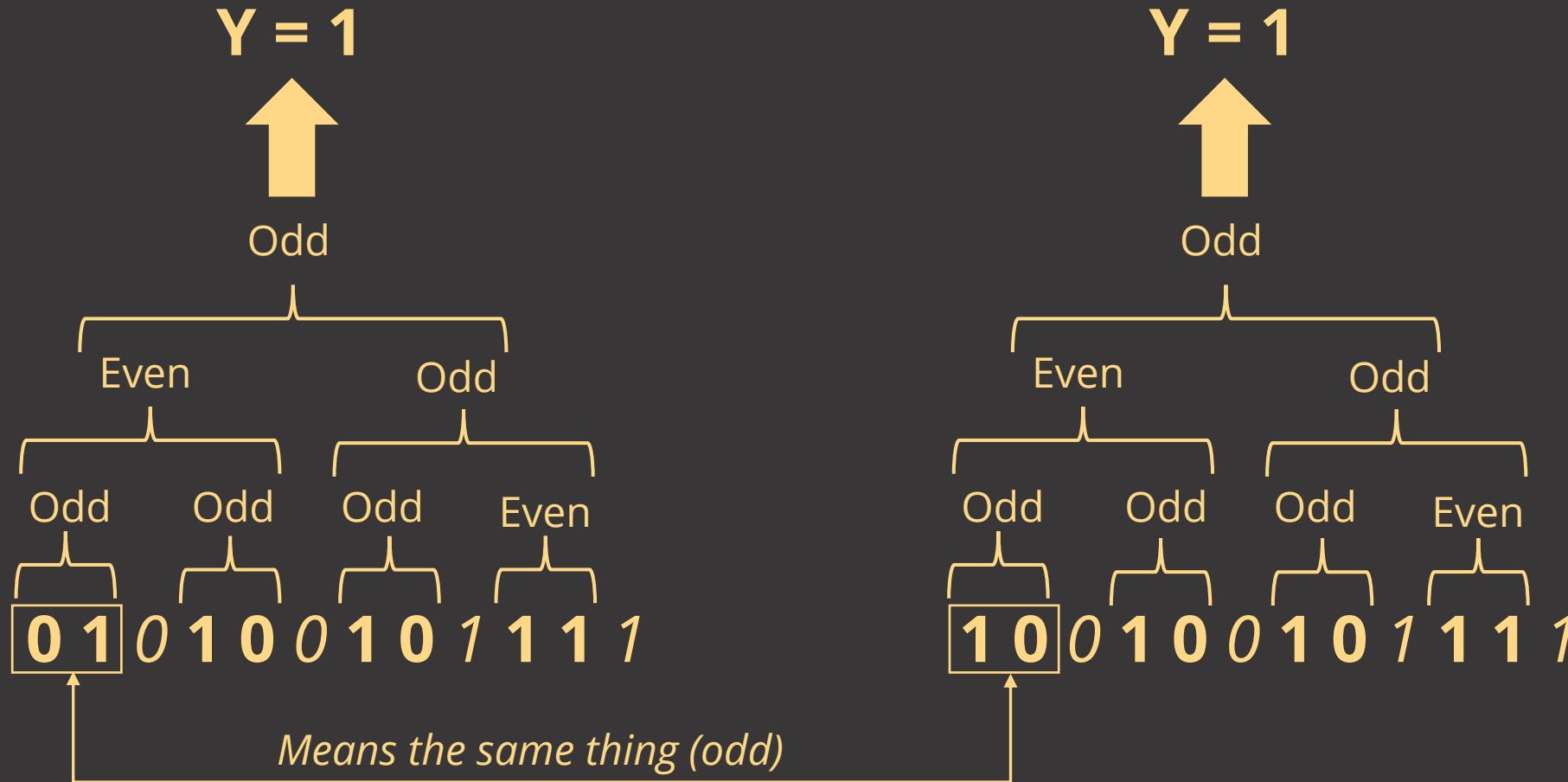
**Y**

?

**Non-linear:** All information is lost if just a single informative bit is missing!



Captures stepwise abstraction (cf. Transformers)



Rewards generalization capability



# Solution with vanilla Tsetlin machine

## Clauses for Y = 1

$$C_1^+ = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_6 \wedge x_7 \wedge \neg x_8$$

$$C_2^+ = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_6 \wedge \neg x_7 \wedge x_8$$

$$C_3^+ = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge \neg x_6 \wedge x_7 \wedge x_8$$

$$C_4^+ = x_1 \wedge x_2 \wedge x_3 \wedge x_4 \wedge \neg x_5 \wedge x_6 \wedge \neg x_7 \wedge x_8$$

⋮

$$C_{128}^+ = x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge \neg x_5 \wedge \neg x_6 \wedge \neg x_7 \wedge \neg x_8$$

## Target model for $Y = 1$

$$C_1^+ \vee C_2^+ \vee C_3^+ \vee C_4^+ \vee \dots \vee C_{128}^+$$

Key learning aspect:  
Replace **OR** with **Addition**  
→ **Class vote sum**

$$C_1^+ \vee C_2^+ \vee C_3^+ \vee C_4^+ \vee \dots \vee C_{128}^+$$

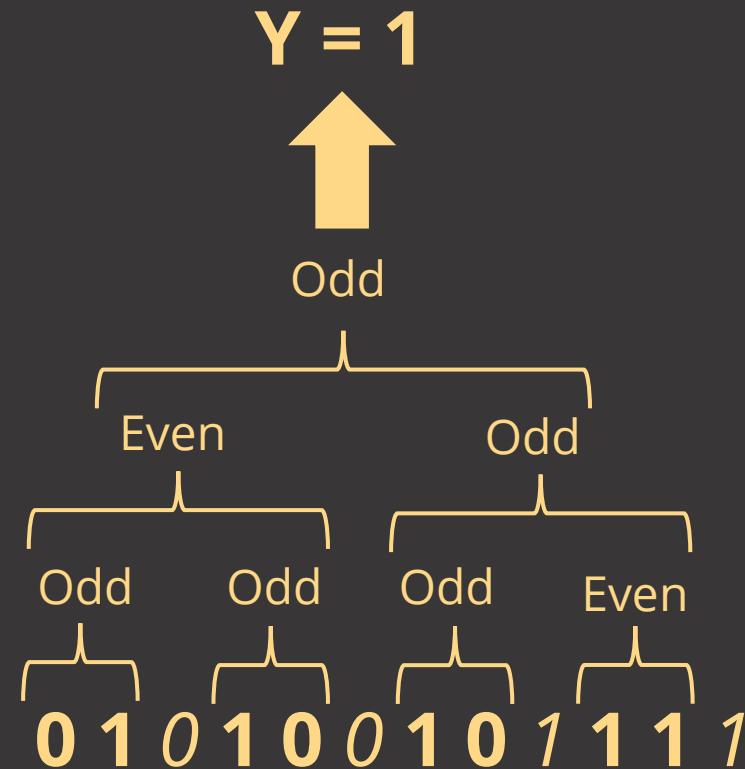


$$C_1^+ + C_2^+ + C_3^+ + C_4^+ + \dots + C_{128}^+$$

**Class vote sum favouring  $Y = 1$**



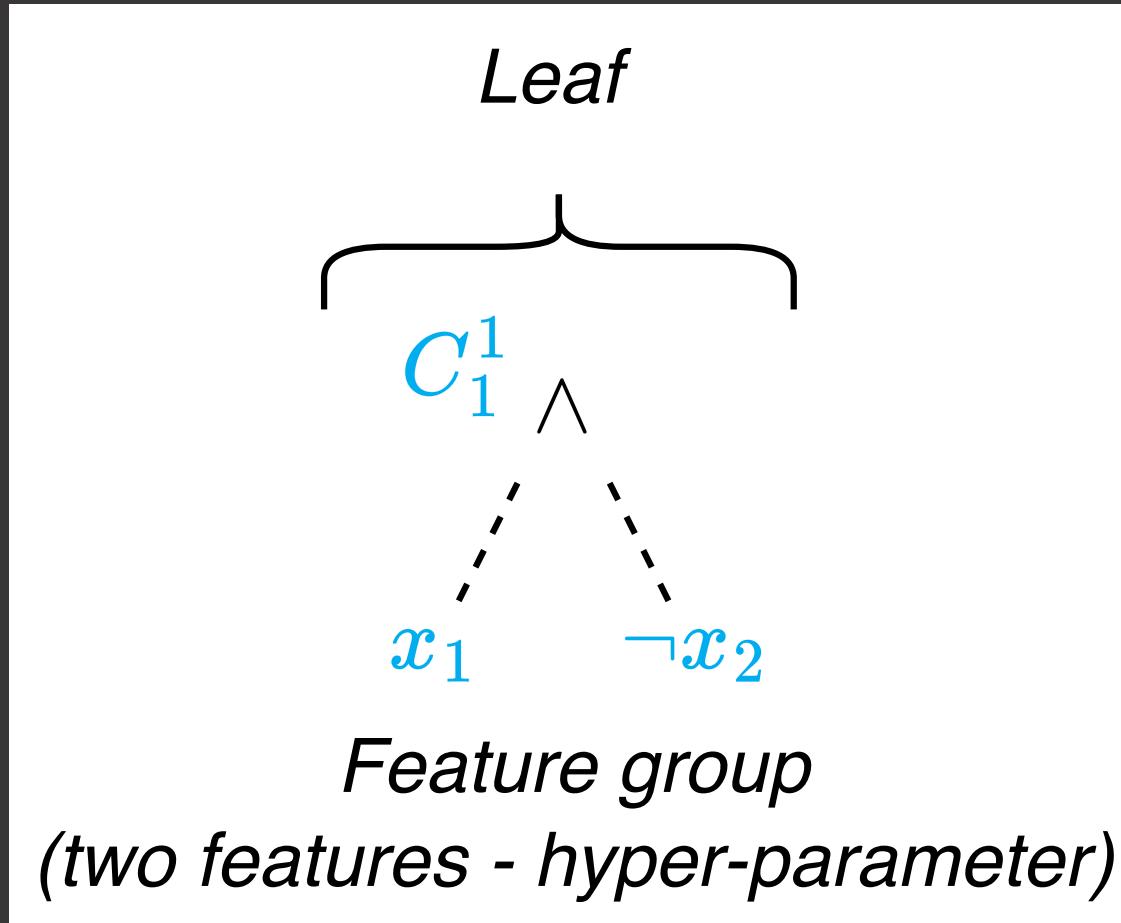
# Learning AND-OR hierarchies with nested clauses



Captures stepwise abstraction (cf. Transformers)

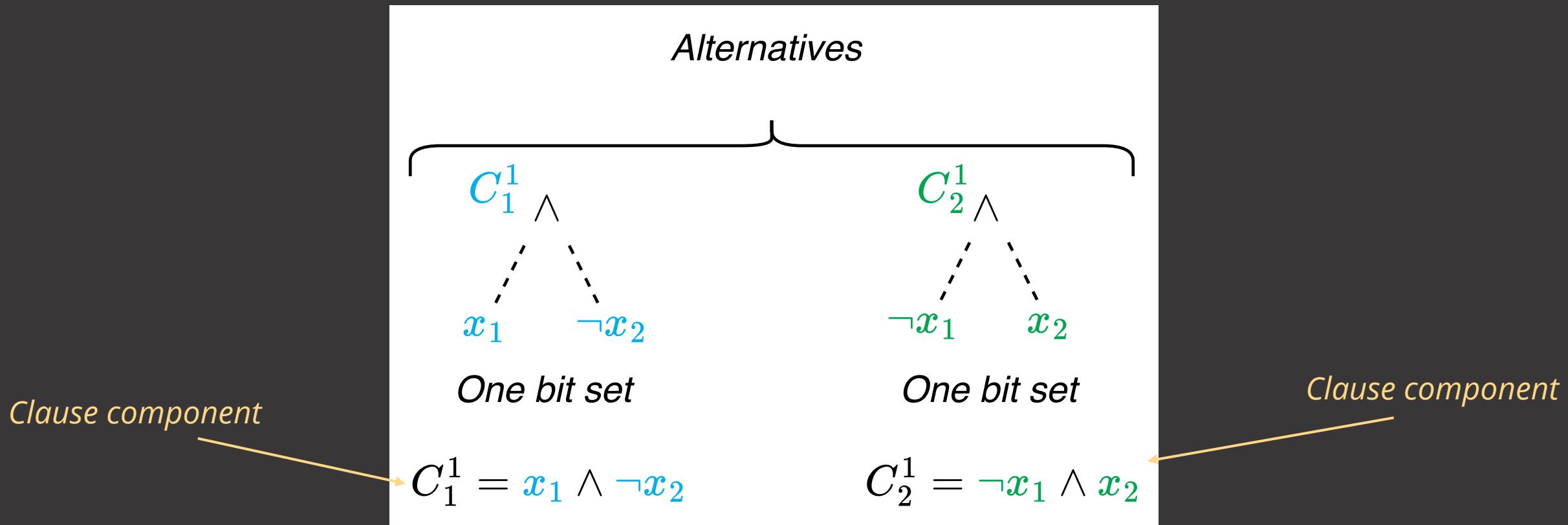
# *Hierarchical clause structure*

# Leaves: clause component

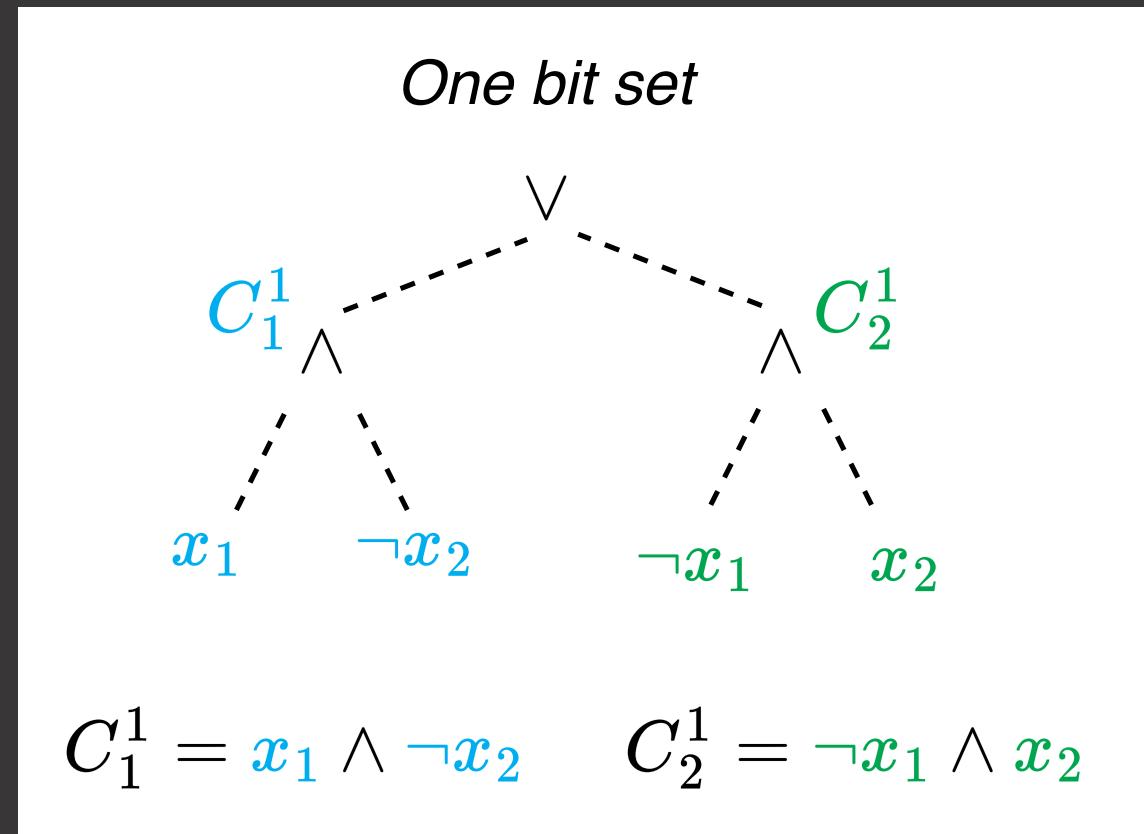


$C_1^1 = x_1 \wedge \neg x_2$   
*Clause Component*

# Alternatives

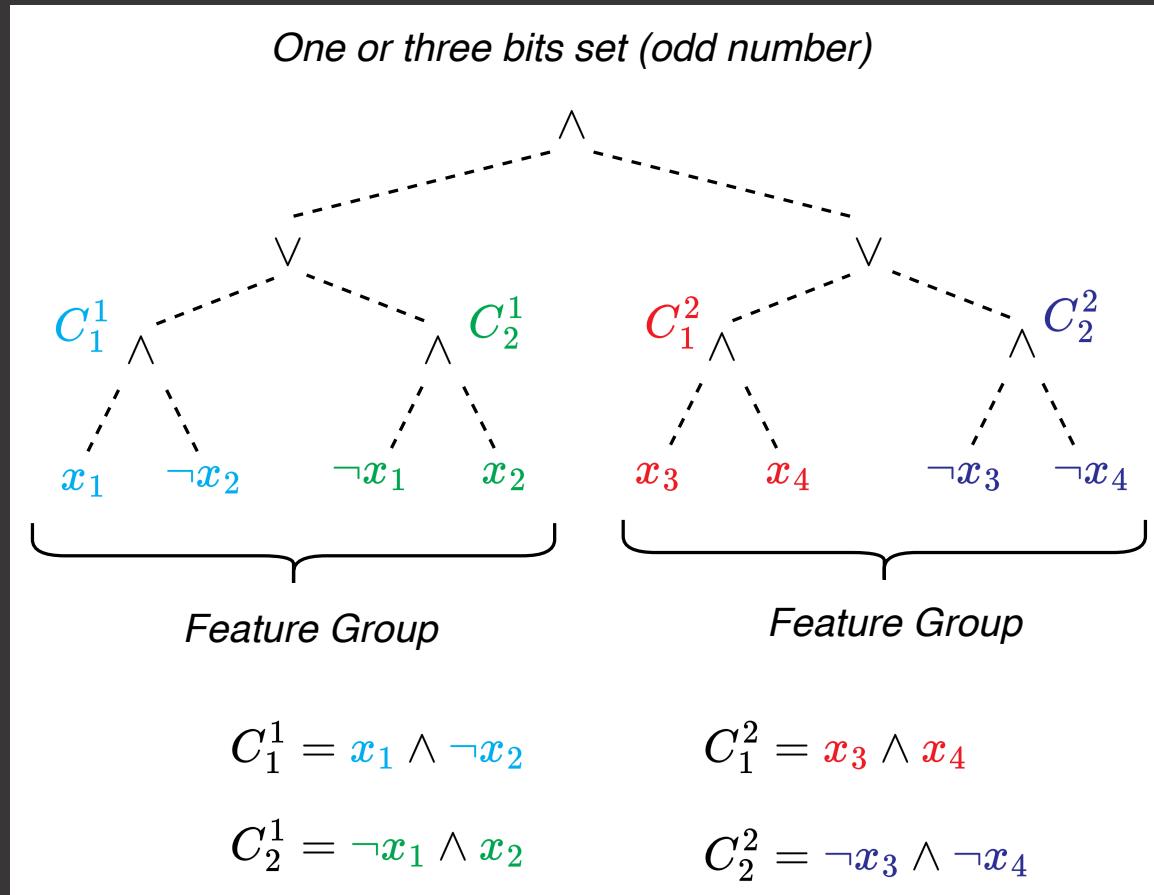


# OR-structure for organizing alternatives



$$\text{Clause} = (C_1^1 \vee C_2^1)$$

# AND-structure for combining feature groups



$$\text{Clause} = (\underline{C}_1^1 \vee \underline{C}_2^1) \wedge (\underline{C}_1^2 \vee \underline{C}_2^2)$$

# *Inference*

Inference is exactly as for vanilla Tsetlin machine, except for clause evaluation...

# Clause evaluation

A clause provides multiple votes:

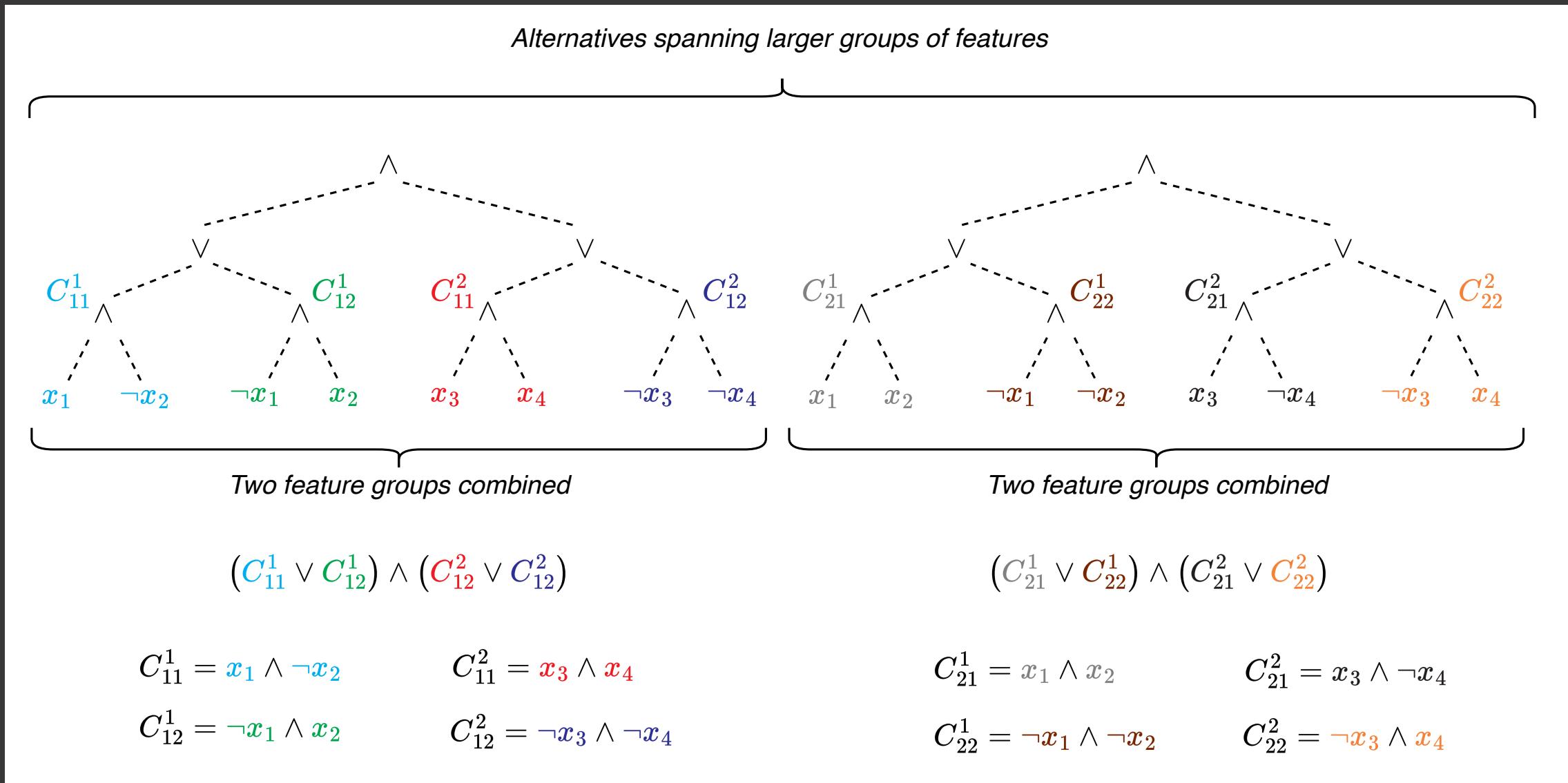
- Replace **OR** with **Addition**
  - Replace **AND** with **Multiplication!**
- **Clause votes**

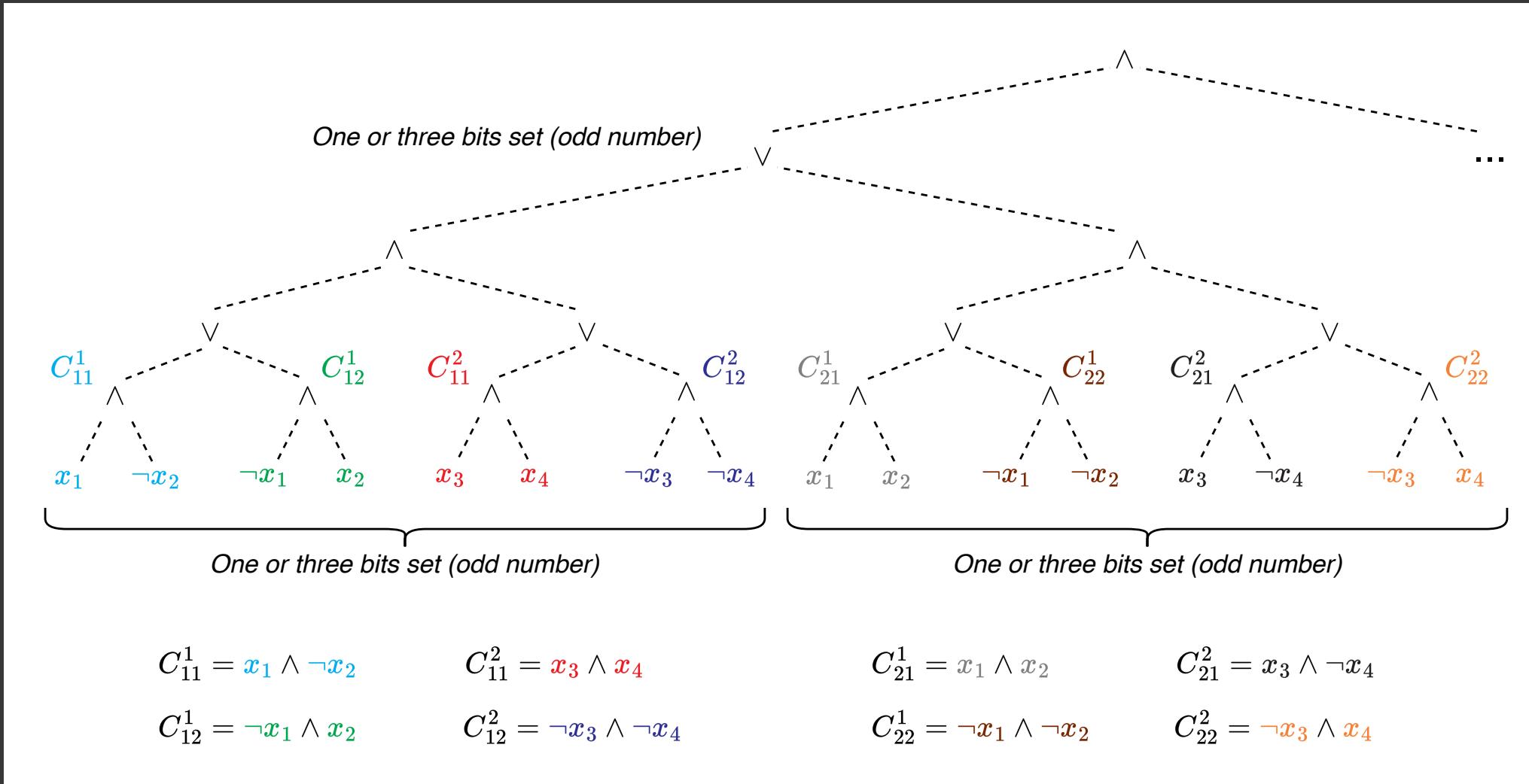
$$\text{Clause} = (C_1^1 \vee C_2^1) \wedge (C_1^2 \vee C_2^2)$$



$$\text{Clause Votes} = (C_1^1 + C_2^1) \times (C_1^2 + C_2^2)$$

*Scaling up*





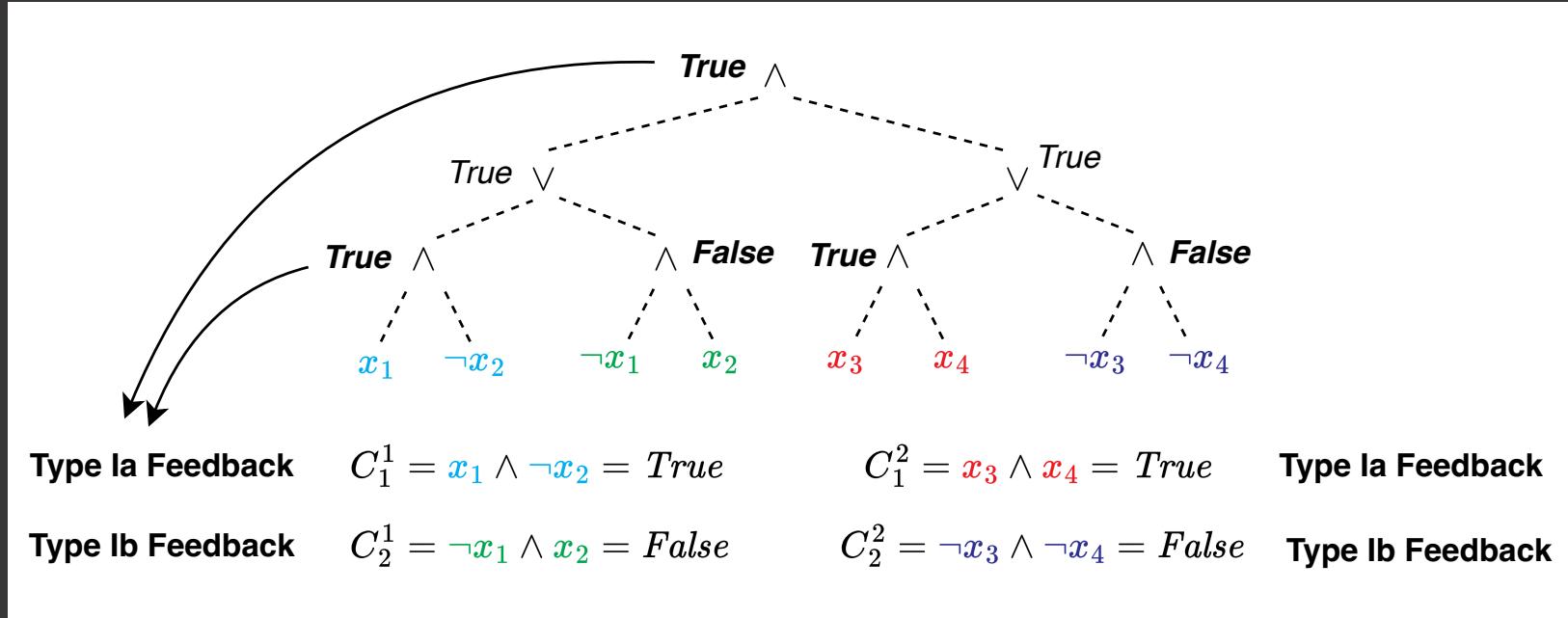
Clause =  $\left[ \left( (C_{11}^1 \vee C_{12}^1) \wedge (C_{11}^2 \vee C_{12}^2) \right) \vee \left( (C_{21}^1 \vee C_{22}^1) \wedge (C_{21}^2 \vee C_{22}^2) \right) \right] \wedge \dots$

$$\text{Clause} = [((C_{11}^1 \vee C_{12}^1) \wedge (C_{11}^2 \vee C_{12}^2)) \vee ((C_{21}^1 \vee C_{22}^1) \wedge (C_{21}^2 \vee C_{22}^2))] \wedge \dots$$

$$\text{Clause Votes} = [((C_{11}^1 + C_{12}^1) \times (C_{11}^2 + C_{12}^2)) + ((C_{21}^1 + C_{22}^1) \times (C_{21}^2 + C_{22}^2))] \times \dots$$

*Learning*

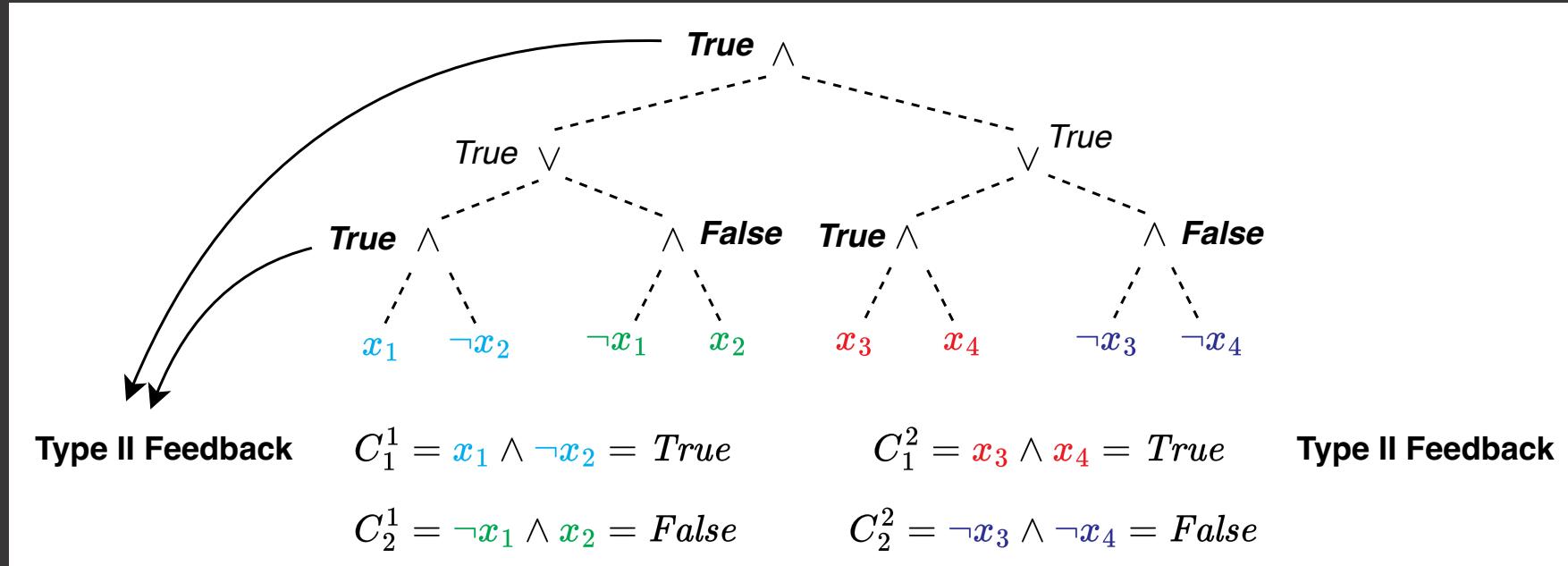
A **clause component** is updated  
exactly as a standard **clause**,  
apart from...



**Input:**  $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$       **Target:**  $y = 1$

### Type I Feedback (Target Class)

- Record truth value of all AND-sub-expressions from leaf (**clause component**) to root (**clause**)
- If all AND-sub-expressions on path are True  $\rightarrow$  Give **Type Ia feedback** to **clause component**
- Otherwise, give it **Type Ib feedback**



**Input:**  $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$       **Target:**  $y = 0$  (noise)

Type II Feedback (Other Classes)

- Record truth value of all AND-sub-expressions from leaf (**clause component**) to root (**clause**)
- If all AND-sub-expressions on path are True → Give **Type II feedback** to **clause component**

# Further work

- Full implementation in PyTsetlinMachineCUDA, TMU, PyTsetlinMachine, and GraphTsetlinMachine
- Reuse/sharing of clause components
- Flexible architecture composition (1D, 2D, 3D, graphs, ...)
- Proof of concept applications (image understanding, natural language understanding, signal processing, etc.)
- Theoretical analysis