III-Logics
Introduction into Propositional Logics

Foundations of Neuro-Symbolic AI

Alex Goessmann

University of Applied Science Würzburg-Schweinfurt

Summer Term 2026

Example: Reasons for a wet street

Let there be a system described by three binary variables (being True or False):

- Wet: Observation, that the street is wet.
- Rained: It had rained.
- Sprinkler: The sprinkler had been on.

Example of Reasoning

When we know that the formulas

- Rained $\Rightarrow$ Wet : Whenever it had rained, the street is wet.
- Rained: It had rained.

hold (i.e. = True), then it follows that the street is wet (i.e. Wet = True).

Necessary Notation for a Logic

Logic is build upon **Syntax**

- How to formulate logical formulas?
- Example: The formula Rained $\Rightarrow$ Wet represents the implication, that the street is wet whenever it had rained.
- Propositional Logics: The variables (or atomic formulas) are combined with logical connectives $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$ to build generic formulas.

and **Semantics**

- What do these formulas describe?
- Example: From the formula Rained $\wedge$ (Rained $\Rightarrow$ Wet) the formula Wet follows.
- Propositional Logics: Each formula describes possible worlds, which are called models of a formula. Comparing the models defines the **entailment relation**.

Model-theoretic semantics: Matrices represent formulas with two variables

We mark the models (the possible worlds) of a formula, and choose an order by

- Wet is False on the first column and True on the second
- Rained is False on the first row and True on the second

Then the model-theoretic semantics of Rained $\Rightarrow$ Wet is represented by the matrix:

$$(\text{Rained} \Rightarrow \text{Wet}) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{1}$$

Entailment by comparing the position of ones

We summarize

$$(\text{Rained} \Rightarrow \text{Wet}) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad \text{Rained} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \tag{2}$$

into

$$\text{Rained} \wedge (\text{Rained} \Rightarrow \text{Wet}) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} . \tag{3}$$

Comparing with

$$\text{Wet} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} . \tag{4}$$

we notice, that every model of Rained $\wedge$ (Rained $\Rightarrow$ Wet) is a model of Wet. We say that Wet is **entailed** and denote

$$\Big(\text{Rained} \wedge (\text{Rained} \Rightarrow \text{Wet})\Big) \models \text{Wet} . \tag{5}$$

Model-theoretic semantics: Tensors represent multiple variables

Tensors are a way to order all possible worlds in a system with multiple variables:

- Each variable is associated with an **axis**.
- For each assignment to a variable we find a **coordinate**

Example of three variables:
We associate the variable Sprinkler with a third direction

$$(\text{Rained} \Rightarrow \text{Wet}) \wedge \text{Sprinkler} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

The curse of dimensionality in propositional logics

Factored System

A factored system is a set of d variables X_k indexed by $k \in [d]$ each having 2 assignments, which interact with each other.

The total number of states of a factored system is

$$2^d \, .$$

Curse of dimensionality

When reasoning about a factored system with many variables (i.e. d is large), the naive enumeration of all models to formulas is infeasible.

The curse of dimensionality can be mitigated by smart distributed representations of the model marking tensors.

Decomposition into logical connectives

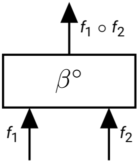
Strategy: Store the semantics of formulas f_1, f_2 and $f_1 \circ f_2$ at each decomposition of the formula.

Distributed Representation

The semantics of complex formulas is then stored by a set of semantics of the used logical connectives $\circ \in \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$.

A **graphical notation** depicts generic tensors:

- Tensors are rectangular boxes
- Each axis of a tensor is drawn by a leg to the box



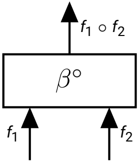
Encoding Logics using Tensors

Boolean states are represented by one-hot encodings

$$\epsilon_{\text{True}} = \epsilon_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \epsilon_{\text{False}} = \epsilon_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} .$$

The truth tables of logical connectives $\circ \in \{\wedge, \vee, \Rightarrow\}$ are represented by tensors $\beta^{\circ} \in \mathbb{R}^{2\times2\times2}$ defined as

$$\beta^{\circ} = \sum_{f_1, f_2 \in \{\text{True}, \text{False}\}} \epsilon_{f_1} \otimes \epsilon_{f_2} \otimes \epsilon_{f_1 \circ f_2}$$



Example: Tensor Core to logical disjunction $\vee$ has the coordinates

$$\beta^{\vee}_{1,:,:} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad \beta^{\vee}_{0,:,:} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} .$$

Encoding of Generic Formulas

The semantics of a formula f is a map

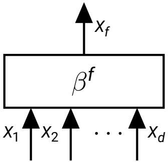
$$f : \bigtimes_{k \in [d]} [2] \to \{0,1\} \, .$$

Given a formula f we call a world index by $(x_0, \ldots, x_{d-1}) \in \bigtimes_{k\in[d]}[2]$ a model of f, if $f(x_0, \ldots, x_{d-1}) = 1$.

formulas are encoded by tensors

$$\beta^f = \sum_{x_0,\ldots,x_{d-1} \in [2]} \left(\bigotimes_{k\in[d]} \epsilon_{x_k} \right) \otimes \epsilon_{f(x_0,\ldots,x_{d-1})}$$

depicted as



.

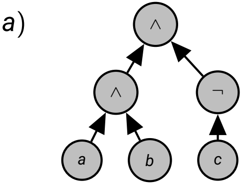
Tensor Calculus for Complex Formulas

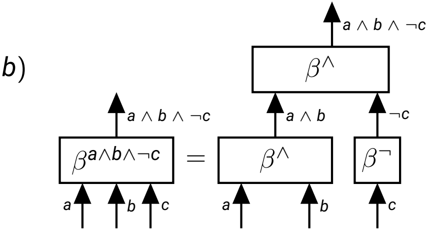
Representation of Complex Formulas

The semantics of complex formulas are retrieved by contractions of their connective semantics, which are summations of tensor coordinates among shared axes.

- Choose distributed representation to avoid contractions
- Only execute those contractions required by reasoning

Contractions can be depicted graphically by a Tensor Network:





Logical Inference: Entailment

The semantics of logics enable the calculation of logical consequences, framed logical inference.

Definition (Entailment)

A propositional formula f is entailed by a formula $\mathcal{KB}$, if for each state $x_0, \ldots, x_{d-1}$ with $\mathcal{KB}(x_0, \ldots, x_{d-1}) = 1$ we have $f(x_0, \ldots, x_{d-1}) = 1$. We denote in this case

$$\mathcal{KB} \models f\,.$$

Approaches to automate the decision of entailment:

- Proof-theoretic approaches: Inference rules such as Modus ponens
- Model-theoretic approaches: Check whether $\mathcal{KB} \wedge \neg f$ has a model.

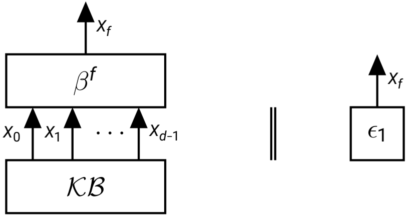
Deciding entailment by Contraction

Theorem (Contraction Criterion for Entailment)

We have $\mathcal{KB} \models f$ (respectively $\mathcal{KB} \models \neg f$) if and only if

$$\left\langle \mathcal{KB}, \beta^f \right\rangle_{[X_f]} \parallel \epsilon_1 \quad \textit{(respectively } \left\langle \mathcal{KB}, \beta^f \right\rangle_{[X_f]} \parallel \epsilon_0\textit{)}\,.$$

We depict this condition by



.

Advantages of the Network Decomposition

Model-theoretic Semantics by Tensor Networks

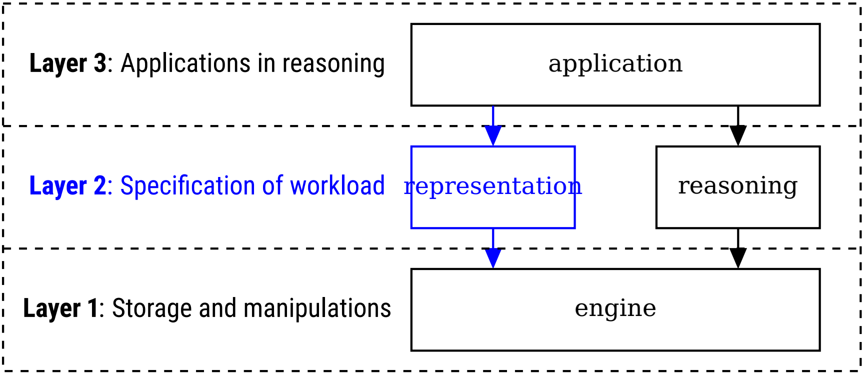
- **Tensors encode the models** of propositional formulas by their nonzero coordinates
- **Networks eliminate redundancies** when representing similar formulas

This is an implementation of the **neural paradigm**, resulting in

- Effective representation: Latent variables by logical formulas
- Differential parametrization: Multilinearity with respect to each core tensor

Implementations in tnreason: Subpackage representation

The subpackage representation is dedicated to the implementation of logics.



Representation of Propositional Formulas

Propositional formulas have three representations specified in the subpackage $\mathrm{representation}$:

- Syntax of formulas f is stored in a script language σ^f based on nested lists of strings
- Semantics of formulas is stored by tensor networks of connective cores
- Random variable representing the formula satisfaction

Representation of Syntax

In $\mathrm{tnreason}$, propositional syntax is represented by nested lists of strings, for example:

["and", ["not", "Rained"], ["imp", "Rained", "Wet"]]

Connective have a defined representation as strings:

Unary connective $\circ$	σ°
$\neg$	*"not"*
$()$	*"id"*

Binary connective $\circ$	σ°
$\wedge$	*"and"*
$\vee$	*"or"*
$\Rightarrow$	*"imp"*
$\oplus$	*"xor"*
$\Leftrightarrow$	*"eq"*

Any other string is interpreted by an atomic formula (a so-called propositional symbol).

Representation of Syntax

Having strings representing

- Connectives (by defined representations)
- Atomic Formulas (all other strings)

we represent formulas $f_1 \circ, f_2$

$$[\sigma^\circ, \sigma^{f_1}, \sigma^{f_2}]$$

where we apply the conventions

- Connectives are at the 0th position in each list
- Further entries are either atoms as strings or encoded formulas itself

Examples of representations

Atomic variable Rained by

σ^{Rained} = "Rained"

Negative literal $\neg$Rained by

["not","Rained"]

Horn clause (Rained $\Rightarrow$ Wet) by

["imp","Rained","Wet"]

Knowledge Base ($\neg$Rained) $\wedge$ (Rained $\Rightarrow$ Wet) by

["and", ["not", "Rained"], ["imp", "Rained", "Wet"]]

Semantics

The recursive structure of the nested lists σ^f is exploited in finding tensor network representations of f. The function

encoding.create_raw_cores()

creates the connective cores to σ^f by

- When σ^f a list, create $\beta^{\sigma^f[0]}$ and add to the tensor networks created by recursive calls to the subformulas $\sigma^f[1:]$
- Return empty list, when σ^f is a string

Then we have

$$f = \left\langle encoding.create\_raw\_cores(\sigma^f)\} \cup \{\epsilon_1 \right\rangle_{\left[f^0,\dots,f^{d-1}\right]}$$