Notes on Possible Implementation of PLN Factor Graphs in MM2+ / MORK

Ben Goertzel

July 20, 2025

Abstract

This note sketches out some crude thoughts about how to implement the "Quantale-Annotated PLN Factor Graph" approach from the *Weakness Is All You Need* draft at a low level in MTM2+ on top of MORK, leveraging MORK's pathmap indexing for hyperedges.

Contents

1 Introduction

This note presents a high-to-medium-level overview of a low-level design for implementing Probabilistic Logic Networks (PLN) as a "quantale-annotated" factor graph within the MM2$^+$/MORK framework. Building on the theoretical foundations laid out in the "Weakness Is All You Need" draft, we show how to:

- Define and combine two key quantales–the *logic-formula* quantale Q_{logic} and the *PLN truth-value* quantale Q_{PLN}–into a single product quantale $Q_{\text{comb}} = Q_{\text{logic}} \times Q_{\text{PLN}}$

- Encode each AtomSpace node (VariableAtom, FactorAtom, FactorVarLink) so that it carries both a symbolic formula component and an uncertainty component in Q_{comb}.
- Construct and manipulate the resulting factor graph via MORK's PathMap indexing, ensuring $O(1)$ or $O(\log N)$ neighbor lookups while preserving the full quantale semantics.
- Illustrate end-to-end inference with toy examples–including simple deduction, nested quantifiers with Skolemization, and background commonsense reasoning–to demonstrate how logical structure and probabilistic truth values propagate together.
- Analyze practical concerns of scalability, independence-assumption errors, and error-propagation, and propose heuristics (clustering, dependency factors, semantic normalization) that guarantee controlled, damped inference over large knowledge bases.

By unifying structural entailment and numeric uncertainty in the same algebraic framework, this approach enables scalable, incremental, and semantically faithful PLN inference suitable for both "background" and goal-directed reasoning in the AtomSpace.

2 The Needed Quantales

2.1 Logic-Formula Quantale

The space of logical formulas in a given first-order (or propositional) language can itself be endowed with a commutative quantale structure:

$$Q_{\text{logic}} = \big(\, \Phi/\!\equiv,\ \leq_{\text{entail}},\ \otimes_{\text{logic}},\ e,\ \bigvee_{\text{logic}} \big),$$

where:

- $\Phi/\!\equiv$ is the set of all well-formed formulas (WFFs), modulo logical equivalence.
- $\phi \leq_{\text{entail}} \psi$ iff $\phi \models \psi$ (logical entailment).
- $\phi \otimes_{\text{logic}} \psi := \phi \wedge \psi$ (conjunction), with unit $e = \mathsf{True}$.
- $\bigvee_{\text{logic}} \{\phi_i\} = \bigvee_i \phi_i$ (disjunction), the least upper bound under entailment.

Message-passing on this quantale corresponds to pure constraint-propagation:

$$\text{Var} \to \text{Fac}: \ m_{v \to f}(v) \ = \bigvee_{f' \in \text{Nbr}_V(v) \setminus \{f\}} m_{f' \to v}(v),$$

$$\text{Fac} \to \text{Var}: \ m_{f \to v}(v) = \exists_{V_f \setminus \{v\}} \Big(\phi_f \ \otimes_{\text{logic}} \bigotimes_{v' \neq v} m_{v' \to f}(v') \Big),$$

where $\exists$ is existential quantification over the eliminated variables

2.2 PLN Simple Truth Value Quantales

2.2.1 A Simple PLN Truth-Value Quantales

Let's assume PLN "simple truth values" for a start (pairs of (strength, confidence) values). Similar approaches can be taken for distributional truth values or other truth value species.

PLN associates each atom with a *Simple Truth Value* (STV), a pair $\langle s, c \rangle$ where $s \in [0,1]$ is the estimated probability and $c \in [0,1]$ the confidence. These STVs could most simply be said to live in the commutative quantale

$$V_{\text{PLN}} \;=\; [0,1] \times [0,1], \quad (s_1, c_1) \;\oplus\; (s_2, c_2) = (s_1 + s_2,\; c_1 + c_2), \quad (s_1, c_1) \;\otimes\; (s_2, c_2) = (s_1 s_2,\; c_1 c_2),$$

with unit $(1,1)$ and zero $(0,0)$:

2.2.2 A Better PLN Truth-Value Quantale

Section 5.10.6 of the original PLN Book develops a full *truth-value arithmetic* over the space of simple truth values

$$V_{\text{PLN}} \;=\; [0,1] \times [0,1],$$

where each pair (s, c) consists of a *strength* s (probability estimate) and a *confidence* c. These can be organized into a commutative quantale $(V_{\text{PLN}}, \oplus, \otimes)$ with the following operations:

Disjunction (Join) $\oplus$

$$(s_1, c_1) \;\oplus\; (s_2, c_2) \;=\; \big(s_1 + s_2 - s_1 s_2 \;,\; c_1 + c_2 - c_1 c_2\big).$$

This is the *probabilistic sum* of two pieces of evidence, combining likelihoods as $P(A \vee B) = P(A) + P(B) - P(A)P(B)$, and likewise aggregating confidences.

Conjunction (Meet) $\otimes$

$$(s_1, c_1) \;\otimes\; (s_2, c_2) \;=\; \big(s_1 s_2 \;,\; 1 - (1 - c_1)(1 - c_2)\big).$$

Strengths multiply under the usual independence assumption $P(A \wedge B) = P(A)\,P(B)$, while confidences combine via the dual probabilistic-sum $1 - (1 - c_1)(1 - c_2)$, ensuring that high reliability in either estimate boosts the joint confidence.

Negation

$$\neg\,(s, c) \;=\; \big(1 - s,\; c\big),$$

so that only the strength is inverted, preserving the original confidence in the evaluation.

Implication and Residuation Implication is defined via the residuation of $\otimes$:

$$(s_A, c_A) \;\to\; (s_B, c_B) \;=\; \Big(\min\big(1,\, 1 - s_A + s_B\big) \;,\;\; f_{\to}(c_A, c_B)\Big),$$

where $f_{\to}$ is chosen so that $\alpha \otimes \beta \le \gamma$ iff $\alpha \le (\beta \to \gamma)$, ensuring the standard adjointness property.

Conditionalization Conditional truth values arise via a Bayes-style update:

$$(s_B \mid A) \;=\; \Big(\tfrac{s_A s_B}{s_A s_B + (1-s_A)(1-s_B)}\;,\;\; g(c_A, c_B)\Big),$$

with g reflecting the combined reliability of the antecedent and consequent.

Together, these operations endow $(V_{\mathrm{PLN}}, \oplus, \otimes)$ with a rich algebraic structure supporting *disjunction*, *conjunction*, *negation*, *implication*, and *conditionalization* directly in the truth-value domain, far beyond the simple product-and-sum quantale

2.3 Combined Quantale on Formulas and Truth-Values

To integrate uncertainty, one forms the product quantale

$$Q_{\mathrm{comb}} = Q_{\mathrm{logic}} \;\times\; Q_{\mathrm{tv}},$$

with carrier $\Phi/\!\equiv \times [0,1]^2$. Its operations are defined component-wise:

$$(\phi_1, \alpha_1) \;\otimes\; (\phi_2, \alpha_2) := \big(\phi_1 \wedge \phi_2 \;,\; \alpha_1 \otimes_{\mathrm{PLN}} \alpha_2\big),$$

$$(\phi_1, \alpha_1) \;\bigvee\; (\phi_2, \alpha_2) := \big(\phi_1 \vee \phi_2 \;,\; \alpha_1 \oplus \alpha_2\big),$$

where $\oplus$ and $\otimes_{\mathrm{PLN}}$ are the probabilistic-sum join and refined conjunction on $V_{\mathrm{PLN}} = [0,1] \times [0,1]$. Marginalization lifts similarly:

$$\exists_x \left(\phi(x), \alpha(x)\right) = \Big(\exists_x \phi(x),\; \bigoplus_x \alpha(x)\Big).$$

Inference via message-passing in Q_{comb} thus propagates both the *structure* of logical constraints and the *numerical* uncertainty of truth values in one unified algebra.

3 AtomSpace Encoding with the Product Quantale

We now encode each Atom in MORK so that it carries both a *logic-formula* value and a *PLN truth-value*, i.e. an element of the product quantale

$$Q_{\mathrm{comb}} = Q_{\mathrm{logic}} \times Q_{\mathrm{PLN}} = \big(\Phi/\!\equiv, \leq_{\mathrm{entail}}, \wedge, \vee\big) \;\times\; \big([0,1]^2, \oplus, \otimes_{\mathrm{PLN}}\big).$$

- **VariableAtom.** Represents a logical formula $\phi \in \Phi$. We store:
  - `FormulaValue`: the formula ϕ itself (element of Q_{logic}).
  - `TruthValue`: a pair $(s, c) \in [0,1]^2$ (element of Q_{PLN}).

  "'metta AtomSpace:setAttribute(v, "FormulaValue", ?) AtomSpace:setAttribute(v, "TruthValue", (s,c)) "'

- **FactorAtom.** Represents one instance of a PLN inference rule f. We store a *combined potential*

  $$\Phi_f : (Q_{\mathrm{logic}} \times Q_{\mathrm{PLN}})^k \;\longrightarrow\; Q_{\mathrm{logic}} \times Q_{\mathrm{PLN}},$$

  which maps k premises $\{(\phi_i, (s_i, c_i))\}$ to $\big(\phi_{\mathrm{out}}, (s_{\mathrm{out}}, c_{\mathrm{out}})\big)$. In metadata: "'metta AtomSpace:setAttribute(f, "CombinedPotential", ϕ_f) "'

- **FactorVarLink.** Connects each `FactorAtom` to its k argument `VariableAtoms`. MORK's `pathmap` then supports $O(1)$ neighbor lookup.

3.1 Local Potentials in the Product Quantale

Each factor's combined potential

$$\Phi_f\big((\phi_1, \alpha_1), \ldots, (\phi_k, \alpha_k)\big) = \big(\phi_{\text{out}},\, \alpha_{\text{out}}\big) \;\in\; Q_{\text{logic}} \times Q_{\text{PLN}},$$

splits into a *logic-part* and a *truth-value-part*:

$$\phi_{\text{out}} = \phi_f^{\text{logic}}\big(\phi_1, \ldots, \phi_k\big), \quad \alpha_{\text{out}} = \phi_f^{\text{tv}}\big(\alpha_1, \ldots, \alpha_k\big),$$

where

$$\phi_f^{\text{logic}}(\phi_1, \ldots, \phi_k) = \text{the deduced formula (e.g. head of the rule)},$$

and ϕ_f^{tv} is the PLN truth-value formula expressed in the enhanced quantale:

$$\phi_f^{\text{tv}}(\alpha_1, \ldots, \alpha_k) = \text{TV-rule}(\alpha_1, \ldots, \alpha_k) \quad \text{using } \oplus,\ \otimes_{\text{PLN}},\ \neg,\ \rightarrow, \ldots$$

Storing and Applying the Potential:

Pseudocode-ishly:

```
# retrieve the combined potential
cpot = AtomSpace:getAttribute(f, "CombinedPotential")

# apply it to all neighbors of f
(phi_out, (s_out, c_out)) = cpot([
    (AtomSpace:getAttribute(v_i, "FormulaValue"),
     AtomSpace:getAttribute(v_i, "TruthValue"))
    for each v_i in Nbr_F(f)
])

# store the results back on the new variable atom
AtomSpace:setAttribute(v_out, "FormulaValue", phi_out)
AtomSpace:setAttribute(v_out, "TruthValue", (s_out, c_out))
```

4 Constructing and Manipulating the Factor Graph with the Product Quantale

Algorithm 1: Build Combined-Quantale PLN Factor Graph in MM2$^+$

Input: Set of PLN formulas Φ, set of inference rule instances $\mathcal{F}$
Output: AtomSpace populated with VariableAtom, FactorAtom, and FactorVarLink aton
foreach $\varphi \in \Phi$ **do**
  $v \leftarrow$ AtomSpace:addAtom("VariableAtom", φ);
  AtomSpace:setAttribute(v, "FormulaValue", φ);
  AtomSpace:setAttribute(v, "TruthValue", initialSTV);

Building the factor

foreach *rule instance* $f \in \mathcal{F}$ *with inputs* $\varphi_1, \ldots, \varphi_k$ **do**
  $f_a \leftarrow$ AtomSpace:addAtom("FactorAtom", ruleType $= f$);
  AtomSpace:setAttribute$\big(f_a$, "CombinedPotential", $\Phi_f : (Q_{\text{logic}} \times Q_{\text{PLN}})^k \to Q_{\text{logic}} \times Q_{\text{PLN}}\big)$;
  foreach $i = 1, \ldots, k$ **do**
    $he \leftarrow$ AtomSpace:addAtom("FactorVarLink");
    AtomSpace:addLink(he, f_a);
    AtomSpace:addLink(he, v_i);

Notes:

- Each `VariableAtom` carries both `"FormulaValue"` $\in Q_{\text{logic}}$ and `"TruthValue"` $\in Q_{\text{PLN}}$.
- Each `FactorAtom` stores a `"CombinedPotential"` mapping $(\phi_i, (s_i, c_i))_{i=1}^k \mapsto (\phi_{\text{out}}, (s_{\text{out}}, c_{\text{out}}))$ in the product quantale $Q_{\text{logic}} \times Q_{\text{PLN}}$.
- `FactorVarLink` atoms link factors to their argument variables, enabling MORK's PathMap to index both logical-formula and truth-value messages in near-constant time.

4.1 Message-Passing with the Product Quantale

Let $\text{Nbr}_V(v)$ be the set of factor nodes adjacent to variable v, and $\text{Nbr}_F(f)$ the set of variable nodes adjacent to factor f.

Variable →Factor:

$$\text{varToFac}(v, f) \;=\; \bigvee_{f' \in \text{Nbr}_V(v) \setminus \{f\}} \text{msg}_{f' \to v},$$

where $\bigvee$ is the join in the combined quantale $Q_{\text{comb}} = Q_{\text{logic}} \times Q_{\text{PLN}}$:

$$(\phi_1, \alpha_1) \bigvee (\phi_2, \alpha_2) \;=\; \big(\phi_1 \vee \phi_2 \;,\; \alpha_1 \oplus \alpha_2\big),$$

and $\oplus$ is the probabilistic-sum join on V_{PLN}:

$$(s_1, c_1) \oplus (s_2, c_2) = \big(s_1 + s_2 - s_1 s_2,\; c_1 + c_2 - c_1 c_2\big).$$

Factor →Variable:

$$\text{facToVar}(f, v) \;=\; \exists_{x \in V_f \setminus \{v\}} \Big(\Phi_f \;\otimes \bigotimes_{v' \in \text{Nbr}_F(f) \setminus \{v\}} \text{varToFac}(v', f) \Big),$$

where

$$(\phi_{f,\text{logic}}, \alpha_{f,\text{tv}}) \otimes (\phi', \alpha') = \big(\phi_{f,\text{logic}} \wedge \phi',\ \alpha_{f,\text{tv}} \otimes_{\text{PLN}} \alpha'\big),$$
$$(s_1, c_1) \otimes_{\text{PLN}} (s_2, c_2) = \big(s_1 s_2,\ 1 - (1 - c_1)(1 - c_2)\big),$$

and existential quantification $\exists$ in Q_{comb} is given by

$$\exists_x (\phi(x), \alpha(x)) = \Big(\exists_x \phi(x),\ \bigoplus_x \alpha(x)\Big).$$

Store each message as a combined-quantale element:

```
AtomSpace:setAttribute(v, "msgTo:"+f, varToFac(v,f))
AtomSpace:setAttribute(f, "msgTo:"+v, facToVar(f,v))
```

4.2 Orchestration Loop

Algorithm 2: PLN Belief-Propagation in Q_{comb}

Input: AtomSpace containing factor-graph links
Output: Combined-quantale messages on all edges
repeat
  foreach *hyperedge he with endpoints* (f, v) **do**
    $m_{v \to f} \leftarrow \text{varToFac}(v, f)$; // $(\phi, \alpha) \in Q_{\text{comb}}$
    AtomSpace:setAttribute(v, "msgTo:"+f, $m_{v \to f}$);
  foreach *hyperedge he with endpoints* (f, v) **do**
    $m_{f \to v} \leftarrow \text{facToVar}(f, v)$;
    AtomSpace:setAttribute(f, "msgTo:"+v, $m_{f \to v}$);
until *convergence or max_iterations reached*;

Final Marginals. For each variable v, the inferred combined-quantale marginal is

$$\text{Marginal}(v) = \bigvee_{f \in \text{Nbr}_V(v)} \text{msg}_{f \to v} \in Q_{\text{comb}}.$$

5 Toy Examples

5.1 Silly Little Toy Example

We illustrate end-to-end inference of Mammal(Lassie) using the combined quantale

$$Q_{\text{comb}} = Q_{\text{logic}} \times Q_{\text{PLN}}.$$

5.1.1 Knowledge Base

- **VariableAtom** vDog**:** FormulaValue = Dog(Lassie), TruthValue = $(0.9, 0.9)$.
- **VariableAtom** vMammal**:** FormulaValue = Mammal(Lassie), TruthValue = $(0, 0)$.
- **FactorAtom** fDed **(Deduction):** CombinedPotential Φ_{ded} maps

$$(\text{Dog}(x), (s, c)) \mapsto (\text{Mammal}(x),\ \text{DeductionSTV}(s, c)),$$

  where $\text{DeductionSTV}(s, c) = (0.8\, s,\ 0.8\, c)$.

5.1.2 Building the Factor Graph

```
vDog    = add VariableAtom("Dog(Lassie)")
set vDog.FormulaValue = Dog(Lassie)
set vDog.TruthValue   = (0.9,0.9)

vMammal = add VariableAtom("Mammal(Lassie)")
set vMammal.FormulaValue = Mammal(Lassie)
set vMammal.TruthValue   = (0.0,0.0)

fDed = add FactorAtom("Deduction")
set fDed.CombinedPotential = ?_ded

link1 = add FactorVarLink; link link1--fDed; link link1--vDog
link2 = add FactorVarLink; link link2--fDed; link link2--vMammal
```

Graph structure:

$$\text{vDog} \xleftrightarrow{\text{link1}} \text{fDed} \xleftrightarrow{\text{link2}} \text{vMammal}$$

5.1.3 Message-Passing Round

Variable → Factor Each variable seeds its outgoing message with its own value and joins any incoming messages:

$$m_{v\to f} = (\phi_v, \alpha_v) \ \vee \bigvee_{f' \in \mathrm{Nbr}_V(v)\setminus\{f\}} m_{f'\to v},$$

where $\vee$ is the join in Q_{comb}. Since there are no other incoming messages here:

$$m_{\text{vDog}\to fDed} = (\text{Dog(Lassie)}, (0.9, 0.9)), \quad m_{\text{vMammal}\to fDed} = (\text{Mammal(Lassie)}, (0, 0)).$$

Factor → Variable Each factor marginalizes out its other argument(s):

$$m_{f\to v} = \exists_{u\neq v}\Big(\Phi_f\big(m_{u\to f}\big)\Big),$$

where $\exists$ applies formula-disjunction and probabilistic-sum to the STV. Thus

$$m_{fDed\to vMammal} = \Phi_{\text{ded}}\big(\text{Dog(Lassie)}, (0.9, 0.9)\big) = \big(\text{Mammal(Lassie)}, (0.72, 0.72)\big).$$

We store:

```
set fDed.msgTo:Mammal(Lassie) = (Mammal(Lassie),(0.72,0.72))
```

5.1.4 Reading Out the Marginal

$$\text{Marginal}(v) = \bigvee_{f\in \mathrm{Nbr}_V(v)} m_{f\to v} \quad \in Q_{\text{comb}}.$$

Hence

$$\text{Marginal}(v_{\text{Mammal}}) = (\text{Mammal(Lassie)}, (0.72, 0.72)).$$

```
set vMammal.Marginal = (Mammal(Lassie),(0.72,0.72))
```

5.1.5 Incorporating New Evidence

To assert Mammal(Lassie) = $(0.95, 0.9)$:

```
set vMammal.TruthValue = (0.95,0.9)
re-enqueue links of vMammal
re-run local var->fac and fac->var updates
```

5.2 Example with Nested Quantifiers & Skolemization

We encode the formula

$$\forall y \left(\exists x \, \mathrm{Parent}(x, y)\right)$$

over $D = \{\mathrm{Anna}, \mathrm{Bob}\}$ with Skolem function $sk(y)$, and infer in $Q_{\mathrm{comb}} = Q_{\mathrm{logic}} \times Q_{\mathrm{PLN}}$.

5.2.1 AtomSpace Encoding

As before, each VariableAtom v carries

$$\texttt{FormulaValue}(v) \in Q_{\mathrm{logic}}, \quad \texttt{TruthValue}(v) \in Q_{\mathrm{PLN}}.$$

FactorAtoms $f_{\exists_A}, f_{\exists_B}, f_{sk_A}, f_{sk_B}, f_{\forall}$ carry the corresponding CombinedPotentials ExistsSTV, SkolemIdentity, ForAl

5.2.2 Message-Passing Round

Variable → Factor Every variable seeds its var→fac message with its own (ϕ_v, α_v) and joins any other incoming messages:

$$m_{v \to f} = (\phi_v, \alpha_v) \ \vee \bigvee_{f' \in \mathrm{Nbr}_V(v) \setminus \{f\}} m_{f' \to v}.$$

Hence, since no other messages exist initially:

$$m_{v_{PA} \to f_{\exists_A}} = (\mathrm{Parent}(\mathrm{Anna}, \mathrm{Alice}), (s_{PA}, c_{PA})), \quad m_{v_{PC} \to f_{\exists_A}} = (\mathrm{Parent}(\mathrm{Charlie}, \mathrm{Alice}), (s_{PC}, c_{PC})),$$

and similarly for v_{PB}, v_{PD} feeding $f_{\exists_B}$.

Existential Factors Each existential factor now receives the correct seeded inputs:

$$m_{f_{\exists_A} \to v_{\exists_A}} = \mathrm{ExistsSTV}\big(m_{v_{PA} \to f_{\exists_A}}, m_{v_{PC} \to f_{\exists_A}}\big) = \mathrm{ExistsSTV}\big((s_{PA}, c_{PA}), \ (s_{PC}, c_{PC})\big),$$

$$m_{f_{\exists_B} \to v_{\exists_B}} = \mathrm{ExistsSTV}\big((s_{PB}, c_{PB}), \ (s_{PD}, c_{PD})\big).$$

Skolem Factors

$$m_{f_{sk_A} \to v_{sk_A}} = \mathrm{SkolemIdentity}\big(m_{f_{\exists_A} \to v_{\exists_A}}\big), \quad m_{f_{sk_B} \to v_{sk_B}} = \mathrm{SkolemIdentity}\big(m_{f_{\exists_B} \to v_{\exists_B}}\big).$$

Universal Factor

$$m_{f_{\forall} \to v_{\forall}} = \mathrm{ForAllSTV}\Big(m_{f_{sk_A} \to v_{sk_A}}, m_{f_{sk_B} \to v_{sk_B}}\Big).$$

5.2.3 Final Marginal

The marginal on $v_\forall$ is the join in Q_{comb}:

$$\text{Marginal}(v_\forall) = \bigvee_{f \in \{\exists_A, \exists_B, sk_A, sk_B, \forall\}} m_{f \to v_\forall} = \big(\forall y\, \phi_{sk_y},\, \alpha_\forall\big).$$

6 Scalability and Applicability of PLN Factor-Graph Inference

These toy examples illustrate the mechanisms involved in using factor graphs for PLN logic, however they do not get across the actual point of using factor graphs against a MORK back-end, which is scalability.

In realistic PLN applications, knowledge bases will contain millions or billions of interrelated formulas and inference rules. By representing PLN as a logical factor graph, we decompose global inference into many small, localized message-passing updates. Each update touches only the factors and variables in a node's immediate neighborhood, so one full belief-propagation sweep costs $O(|E|)$ where $|E|$ is the number of factor-variable links.

Moreover, MORK's PathMap gives $O(1)$ or $O(\log N)$ retrieval of a node's neighbors, and shared subformula interning ensures that common expressions are updated only once. As a result, inference naturally parallelizes across cores or machines, supports incremental updates when evidence changes, and gracefully handles sparse, high-dimensional graphs–properties essential for scaling PLN from toy examples to real-world knowledge bases.

Factor Graph Methods vs Chaining We are not arguing that factor graphs will kick ass for every sort of inference control required for PRIMUS or narrow practical applications. Sometimes good old backward chaining will be best.

Roughly speaking, we could say that chaining–whether forward or backward–excels whenever the inference task is highly goal-driven or involves only a small, localized portion of the knowledge base. In such scenarios, one can (hope to) dynamically select and apply only those PLN rules and knowledge items that directly contribute toward the target conclusion, thereby avoiding the setup and maintenance of a full factor graph over every variable and rule. As a result, chaining incurs cost roughly proportional to the number of rules actually fired, rather than to the total number of factor-variable links $|E|$. Furthermore, chaining easily accommodates heuristic guidance–such as rule-priority orderings, admissible cost estimates, and early stopping once the goal is reached–whereas factor-graph belief propagation proceeds in a uniform, undirected fashion and must typically run to convergence across the entire graph. Consequently, for sparse, focused, or hierarchical reasoning problems, chaining often achieves faster response times and lower memory usage than global factor-graph message-passing.

Goal-directed inference is certainly highly important for PRIMUS and various applications, so that working toward scalable heuristics for backward chaining is very important. However, broad-based "background inference" aimed at bringing out knowledge implicit in the overall Atomspace is also very important, and this seems to be where factor graph approaches would excel.

Hand-Wavy Examples of Background Inference As rough evocative examples of how continual background inference over the AtomSpace (such as a factor graph representation may efficiently support) can surface rich, implicit knowledge, let's consider some illustrative scenarios...

1. **Science question answering.** Suppose the user asks: "Why does adding salt to water lower its freezing point?" A broad-based factor-graph sweep can in parallel combine:

- the rule for freezing-point depression proportional to molality,
- the definition of molality as moles of solute per kilogram of solvent,
- Raoult's law for vapor-pressure lowering,
- the relation between vapor pressure and phase-change temperature.

This precomputes support for intermediate concepts (for example, salt dissociates into ions) and highlights low-confidence links for further targeted reasoning.

2. **Humanoid-robot dialogue.** Imagine a robot companion overhears: "It's a bit drafty in here." Background inference can:

   - infer that a drafty room implies reduced thermal comfort,
   - link thermal comfort to actions like closing a window or adjusting the thermostat,
   - consult the user's past temperature preference (e.g. 72 deg F),
   - propagate confidence levels to decide whether to suggest: "Shall I close the window for you?"

   A single factor-graph pass activates these related beliefs without any explicit chaining goal.

3. **Multi-turn context management.** During a science tutorial, the user asks: "How does photosynthesis adjust when light intensity changes?" Background inference can maintain a factor graph over:

   - light intensity versus chlorophyll absorption,
   - stomatal opening versus CO2 uptake,
   - enzyme activity versus temperature,
   - feedback loops in ATP and NADPH production.

   If the conversation then shifts to "What about under water?" one can incrementally update the same graph (for example by adding refractive-index effects) instead of rebuilding it from scratch.

7 A Scalable Heuristic for Minimizing the Damage from Independence Assumptions

We now dig a bit into the question of independence assumptions that are made implicitly in the mathematics used above, and how to best minimize and manage their incorrectness.

7.1 Independent Conjunction in the Product Quantale

Let us first clarify the nature of the problem. According to the above factor-graph math we incur an independence assumption whenever two premises are conjoined. The *monoid* operation in Q_{comb} is now

$$(\phi_1, (s_1, c_1)) \otimes (\phi_2, (s_2, c_2)) = \Big(\phi_1 \wedge \phi_2,\ (s_1 s_2,\ 1 - (1 - c_1)(1 - c_2))\Big),$$

where:

- $\phi_1 \wedge \phi_2$ is the conjunction in the logic-formula quantale Q_{logic}.
- $(s_1 s_2,\ 1 - (1 - c_1)(1 - c_2))$ is the refined conjunction in the PLN truth-value quantale Q_{PLN}.

and the probabilistic independence assumption is plain to see.

Justification of the above

Structural meet: Conjoining the formulas ϕ_1, ϕ_2 yields $\phi_1 \wedge \phi_2$, capturing the logical constraint that both premises hold.

Strength (s): Under independence, $P(A \wedge B) = P(A)\,P(B)$, so we multiply strengths $s_1 s_2$.

Confidence (c): We combine confidences via the dual probabilistic-sum $1 - (1 - c_1)(1 - c_2)$, ensuring that high reliability in either premise raises the joint confidence.

Implementation In MeTTa-IL or native code, the factor-to-variable "product" message now becomes:

```
quantale.times(
  (phi1, (s1,c1)),
  (phi2, (s2,c2))
) -> (
  phi1 /\ phi2,
  (s1*s2, 1 - (1-c1)*(1-c2))
)
```

While residual independence errors must still be managed (e.g. via clustering or dependency factors), this combined-quantale conjunction ensures that both the *structure* of the logic and the *uncertainty* of the truth values are aggregated in a single principled operation.

7.2 Heuristic Adjustment of the Combined-Quantale Factor Graph

To properly handle non-independence in the product quantale

$$Q_{\text{comb}} = Q_{\text{logic}} \times Q_{\text{PLN}},$$

one potentially useful approach seems to be to run a periodic restructuring pass over the AtomSpace. This leverages MORK's `PathMap` for fast neighbor lookups and either (a) inserts explicit dependency

factors or (b) clusters strongly dependent variables into a macro-variable.

Algorithm 3: Dependency-Handling via PathMap in Q_{comb}

```
Input: DEPENDENCE_THRESHOLD ∈ [0,1], USE_CLUSTERING ∈ {false, true}
Data: AtomSpace, pathmap
Result: Graph restructured with clusters or explicit dependency factors
vars ← AtomSpace:getAtomsByTag("VariableAtom");
foreach v in vars do
    F_v ← pathmap.getFactorsForVariable(v);
    sharedCount ← empty map;
    foreach f in F_v do
        foreach u in pathmap.getVariablesForFactor(f) if u ≠ v do
            sharedCount[u]++;
    deg_v = |F_v|;
    foreach (u, c) in sharedCount do
        F_u ← pathmap.getFactorsForVariable(u);
        deg_u = |F_u|;
        jaccard = c/(deg_v + deg_u − c);
        if jaccard > DEPENDENCE_THRESHOLD then
            if USE_CLUSTERING then
                cid ← mergeClusterIds(v.ClusterId, u.ClusterId);
                AtomSpace:setAttribute(v, "ClusterId", cid);
                AtomSpace:setAttribute(u, "ClusterId", cid);
            else
                f_dep ← AtomSpace:addAtom("FactorAtom","DependencyFactor");
                AtomSpace:setAttribute(f_dep, "CombinedPotentialRoutine", "DepJointCombined");

                foreach w ∈ {v, u} do
                    link ← AtomSpace:addAtom("FactorVarLink");
                    AtomSpace:addLink(link, f_dep);
                    AtomSpace:addLink(link, w);
```

Complexity. Each variable v visits its adjacent factors once (total $O(|E|)$) and uses PathMap lookups in $O(1)$ or $O(\log N)$, so the whole pass runs in $O(|E|)$.

After the pass.

- If clustering was selected, all `VariableAtom`s sharing a `ClusterId` are treated as one macro-variable. Its
  - *FormulaValue* is the conjunction $\bigwedge_i \phi_i$ in Q_{logic}.
  - *TruthValue* is the PLNs-conjunction $\bigotimes_{\text{PLN}}\{(s_i, c_i)\}$ in Q_{PLN}.
- If dependency factors were inserted, each new `FactorAtom` ‘"DependencyFactor"‘ has a `CombinedPotentialRoutine` ‘"DepJointCombined"‘ which implements

$$(\phi_A, \alpha_A) \otimes (\phi_B, \alpha_B) = (\phi_A \wedge \phi_B,\ \text{DepJointSTV}(\alpha_A, \alpha_B)),$$

i.e. the exact joint-formula conjunction in Q_{logic} plus the true joint STV in Q_{PLN}.

This restructuring cannot entirely eliminate independence errors in Q_{comb}, but it significantly reduces them by ensuring that any remaining $\otimes$ operations occur only over (approximately) independent clusters or through explicit joint-STV factors. The same pass also benefits chaining-based inference by grouping correlated formulas before rule-firing.

8 Estimating and Managing Error Propagation

We now return to the issue of managing independence assumptions, trying to understand what must be done to prevent the errors that will inevitably remain after preventative measure are taken, from multiplying and dominating results.

In the combined quantale

$$Q_{\text{comb}} = Q_{\text{logic}} \times Q_{\text{PLN}},$$

each message is a pair (ϕ, α), where $\phi \in Q_{\text{logic}}$ is a formula and $\alpha = (s, c) \in Q_{\text{PLN}}$ its truth-value. Independence-assumption errors enter only in the numeric component α, but propagate through the structure via the combined monoid:

$$(\phi_1, \alpha_1) \ \otimes \ (\phi_2, \alpha_2) = \big(\phi_1 \wedge \phi_2 \, , \ \alpha_1 \otimes_{\text{PLN}} \alpha_2 \big).$$

8.1 Local Error in Q_{comb}

For two variables A, B with true joint message (ϕ_{AB}, α_{AB}) but combined via

$$(\phi_A, \alpha_A) \otimes (\phi_B, \alpha_B) = \big(\phi_A \wedge \phi_B, \ \alpha_A \otimes_{\text{PLN}} \alpha_B\big),$$

the *numeric* local error is still

$$\varepsilon_{AB} = \big\| \, \alpha_{AB} \ - \ (\alpha_A \otimes_{\text{PLN}} \alpha_B) \big\|,$$

while the logical part incurs no numeric error (conjunction is exact).

8.2 Error Propagation on Trees

If the factor graph is a tree, numeric errors along a path of length d in the α-component multiply (or add in log-space). After d combined-quantale $\otimes$ operations each with relative error δ, the strength may deviate by up to

$$(1 + \delta)^d.$$

8.3 Damping via Marginalization

Each variable-to-factor or factor-to-variable marginalization uses the $\bigvee$-join in Q_{comb}:

$$\bigvee (\phi_i, \alpha_i) = \Big(\bigvee_{\text{logic}} \phi_i, \ \bigoplus \alpha_i \Big),$$

where $\bigoplus$ is the probabilistic-sum on Q_{PLN}. This re-normalization in the numeric component dampens accumulated errors, since the denominator in the probabilistic-sum prevents unbounded growth.

8.4 Loop Effects and Fixed-Point Behavior

In loopy propagation, the structural ϕ-updates (conjunctions and disjunctions) are exact, while the numeric α-updates converge to a fixed point under repeated $\otimes_{\mathrm{PLN}}$ and $\oplus$. Empirically, small local errors ε_{AB} tend to be averaged out by the normalization in $\oplus$, yielding a damped bias.

8.5 Empirical Error-Bounding

Because the propagation of numeric error depends on graph topology and the interaction of $\otimes_{\mathrm{PLN}}$ and $\oplus$, the practical way to estimate error is to:

1. Measure local numeric errors ε_{AB} on a gold-standard subgraph.
2. Run one round of combined-quantale message-passing and observe the amplification or damping in the (s, c) component.
3. Extrapolate that factor to your full graph to predict overall bias.

Takeaways

- On sparse, tree-like regions, numeric errors multiply along chains unless $\oplus$ normalizations are frequent enough.
- On dense or loopy regions, the join in Q_{comb} typically dampens numeric errors back toward a stable fixed point.
- The logical-formula component is unaffected by numeric error and remains exact under entailment.

8.6 Formal Sketch under Log-Normal Model

Model each factor's numeric error $\Delta_f = \alpha_{\mathrm{true}}/(\alpha_A \otimes_{\mathrm{PLN}} \alpha_B)$ as i.i.d. log-normal with $\log \Delta_f \sim (0, \sigma^2)$. If each variable normalizes over m incoming numeric messages, then after d $\otimes_{\mathrm{PLN}}$-updates and d $\oplus$-normalizations the expected numeric error is

$$E[\varepsilon_d] = \exp\big(d(\tfrac{1}{2}\sigma^2 - \log m)\big),$$

so that $\log m > \frac{1}{2}\sigma^2$ guarantees exponential damping of numeric error in Q_{PLN}, and hence in the combined quantale.

9 Ontologies FTW

The main lesson from these heuristic calculations seems to be: To cause error not to propagate too badly, we should introduce cluster atoms and explicit-dependency atoms, just as was suggested above for the purpose of reducing independence-assumption errors in the first place.

Among other things this seems to be an argument for **semantic normalization of knowledge into a standard ontology of relatively independent concepts**. To the extent one does this, one can reduce independence-assumption errors in PLN as well as in the quantale truth value math underlying factor graphs, and also ensure errors generally dampen each other rather than multiplying.

9.1 Conclusions in the Product Quantale

The main lesson from our heuristic analysis in the combined quantale

$$Q_{\text{comb}} = Q_{\text{logic}} \times Q_{\text{PLN}}$$

that, to prevent error from spiraling out of control, we must absorb strong dependencies up-front by either clustering or explicit dependency factors–applied to both the *formula* and *truth-value* components.

Semantic Normalization. By clustering strongly dependent variables A and B into one macro-variable X, we replace several repeated

$$(\phi_A, \alpha_A) \otimes (\phi_B, \alpha_B) = (\phi_A \wedge \phi_B,\ \alpha_A \otimes_{\text{PLN}} \alpha_B)$$

operations with a single retrieval of the true joint message

$$(\phi_X, \alpha_X) = \text{JointCombined}((\phi_A, \alpha_A), (\phi_B, \alpha_B)),$$

where

$$\phi_X \ = \ \phi_A \wedge \phi_B, \qquad \alpha_X \ = \ \text{TrueJointSTV}(\alpha_A, \alpha_B).$$

Because (ϕ_X, α_X) is stored once on X, its local independence-error Δ_X is absorbed at cluster creation rather than multiplied at each propagation step. Moreover, X inherits the sum of the fan-ins of A and B, increasing its normalization factor m_X, so that if $\log m_X > \frac{1}{2}\,\sigma_X^2$ we achieve exponential damping of any residual errors.

Explicit Dependency Factors. Alternatively, inserting a `DependencyFactor` between A and B uses a combined-quantale potential

$$\Phi_{\text{dep}}\big((\phi_A, \alpha_A), (\phi_B, \alpha_B)\big) = \big(\phi_A \wedge \phi_B,\ \text{TrueJointSTV}(\alpha_A, \alpha_B)\big).$$

Messages then flow

$$(\phi_A, \alpha_A) \ \longrightarrow \ \text{DependencyFactor} \ \longrightarrow \ (\phi_B, \alpha_B),$$

ensuring the exact joint formula conjunction and STV are used, and leaving only approximately independent edges elsewhere. Every remaining conjunction in Q_{comb} therefore has $\Delta_f \approx 1$, guaranteeing $\log m > \frac{1}{2}\,\sigma^2$ and hence stable, damped inference.

In practice, this product-quantale approach integrates structural and numeric normalization: clustering or dependency factors both collapse logical correlations (via $\wedge$) and absorb numeric dependence (via TrueJointSTV*), so that – if all goes well – subsequent message-passing in Q_{comb} enjoys exponential error damping rather than uncontrolled amplification.*