Logic-Based Artificial Intelligence Algorithms Supporting Categorical Semantics

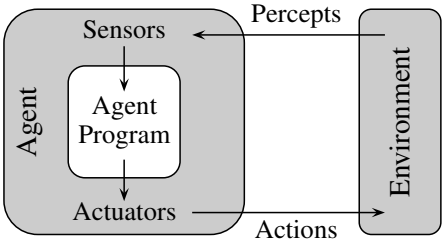
Ralph L. Wojtowicz
Shenandoah University
Department of Mathematics
Winchester, VA USA
rwojtowi@su.edu, ralphw@bakermountain.org

This paper seeks to apply categorical logic to the design of artificial intelligent agents that reason symbolically about objects more richly structured than sets. Using Johnstone's sequent calculus of terms- and formulae-in-context, we develop forward chaining and normal form algorithms for reasoning about objects in cartesian categories with the rules for Horn logic. We also adapt first-order unification to support multi-sorted theories, contexts, and fragments of first-order logic.

1 Introduction

The discovery of categorical logic is one of the 20th century's great contributions to mathematics. A facet of this field is characterizations of the semantic categories for classes of logical theories. First-order Horn, regular, coherent and intuitionistic theories, for example, are respectively interpreted in cartesian, regular, coherent, and Heyting categories [4, 10, 14]. Topological spaces and sheaves provide semantics for propositional and first-order S4 modal logics [2]. Cartesian-closed categories give semantics for the typed λ-calculus [14, 16]. Fragments of linear logic are interpreted in $*$-autonomous categories [3].

This paper grew from an effort to use the syntax and categorical semantics of [14] to reformulate the logic-based artificial intelligence (AI) methods of [24]. It is motivated by both pedagogy and applications. Over the past two years, we have taught undergraduate AI courses using this formulation and have found the sequent calculus of [14] (see 2.1.3 of this paper) to have several advantages. It is concise, has precise rules for substitution and equality, and can be introduced incrementally using fragments of first-order logic. Derived axioms serve as exercises. Categorical logic also clarifies the distinction between syntax and semantics. Truth table and Venn diagrams, for example, arise as propositional semantics in Set and Set/X. Directed graphs and other familiar objects illustrate semantics with multiple truth values.

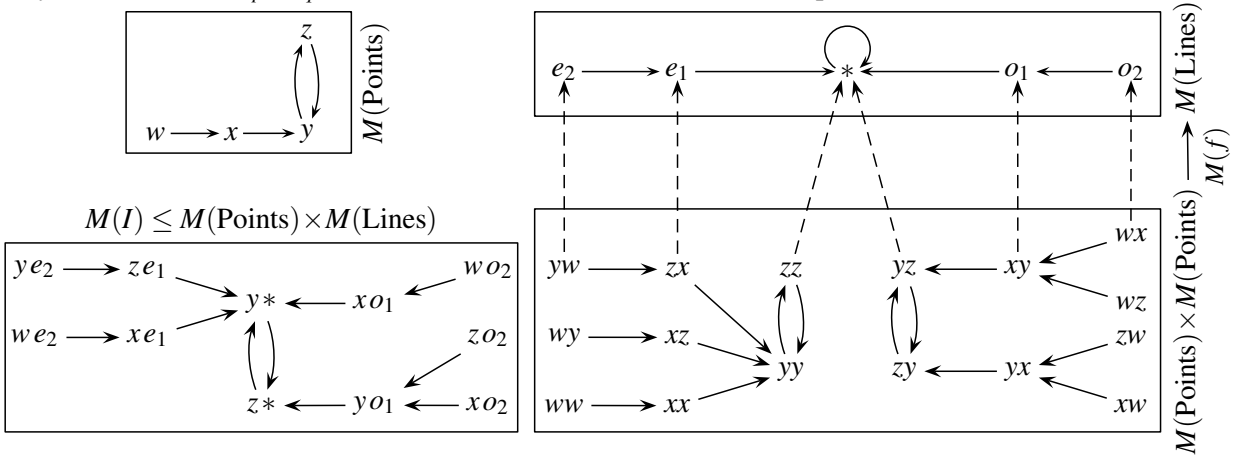


[24] formulates the field of AI in terms of agents. An *agent* is a system that can be viewed as perceiving its environment through sensors and acting upon the environment through actuators. The engineer's task is to design and implement an *agent program* that processes inputs then chooses appropriate actions. Agents may maintain representations of goals, history, and the environment. *Logic-based agents* use representations expressed as logical theories. Traditional presentations (e.g., [19, 23, 24]) rely on classical logic and semantics.

This paper seeks to apply categorical logic to the design of agents that reason symbolically about objects more richly structured than sets. Use of such abstraction barriers is an idiom that is frequently used to manage software complexity [1]. As an illustration, consider a theory $\mathbb{T}$ with two sorts, Points and Lines, a function symbol f : Points, Points $\to$ Lines, a relation symbol $I \rightarrowtail$ Points, Lines, and an axiom

Submitted to:
ACT 2023

$(\top \vdash_{x,y} (I(x,f(x,y)) \wedge I(y,f(x,y))))$. We define a model M of $\mathbb{T}$ in the category $\mathrm{Set}^{\circlearrowleft}$ ([18, 32]). Objects of $\mathrm{Set}^{\circlearrowleft}$ are *iterators* (discrete-time dynamic systems): pairs (X,g) with X a set and $g : X \to X$ a function. A map $\phi : (X,g) \to (Y,h)$ is a function that respects the dynamics: $\varphi \circ g = h \circ \varphi$. We interpret the sorts of $\mathbb{T}$ as iterators $M(\text{Points})$ and $M(\text{Lines})$, the function symbol as a map $M(f) : M(\text{Points}) \times M(\text{Points}) \to M(\text{Lines})$ and the relation symbol as a subobject $M(I) \leq M(\text{Points}) \times M(\text{Lines})$. For $x \in M(\text{Points})$, let t_p be the time for p to enter the 2-cycle. Points p and q are on line e_i if $t_p - t_q$ is even and $\max(t_p,t_q) = i$. They are on line o_i if $t_p - t_q$ is odd. x and y are on the line o_1, for example, while w and y are on e_2.



There is a vast literature on categories whose objects are used in AI. Categories of probability spaces can be traced to [6, 17]. A sample of other resources includes [5, 11, 13, 22, 25, 30]. Note that use of semantic categories of probabilistic objects differs from assigning probabilities to logical formulae [8, 26, 35]. Categories of fuzzy sets are well-studied [9, 12, 14, 27, 37]. Belief functions occur in [33, 34].

2 Categorical Logic

2.1 Syntax

The set Σ-Type of first-order *types* generated by a set Σ-Sort of *sorts* consists of finite lists $A_1, \ldots, A_n$ of sorts including the empty list which is written $[]$. A first-order *signature* Σ has: (1) a set Σ-Sort of sorts; (2) a set Σ-Fun of *function symbols* together with maps $\Sigma\text{-Fun} \to \Sigma\text{-Type}$ and $\Sigma\text{-Fun} \to \Sigma\text{-Sort}$ respectively assigning to each function symbol its *type* and *sort*; (3) a set Σ-Rel of *relation symbols* together with a map $\Sigma\text{-Rel} \to \Sigma\text{-Type}$ assigning to each relation symbol its type; (4) a set Σ-Var of *variables* together with a map $\Sigma\text{-Var} \to \Sigma\text{-Sort}$ assigning to each variable its *sort*. $f : A_1, \ldots, A_n \to B$ indicates that f is a function symbol with type $A_1, \ldots, A_n$ and sort B. If $n = 0$, then f is a *constant*. $R \rightarrowtail A_1, \ldots, A_n$ indicates that R is a relation symbol of type $A_1, \ldots, A_n$. If $n = 0$, then R is a *proposition*. $x : A$ indicates that x is a variable of sort A. We assume a countably infinite supply of variables of each sort.

2.1.1 Terms and Formulae

We recursively define the *terms* over a signature Σ together with the *sort* $t : A$ and the set $\mathrm{FV}(t)$ of *variables* of each term. (1) A variable $x : A$ is a term with $\mathrm{FV}(x) = \{x\}$. (2) If $f : A_1, \ldots, A_n \to B$ is a function symbol and $t_i : A_i$ are terms, then $f(t_1, \ldots, t_n) : B$ is a term with $\mathrm{FV}(f(t_1, \ldots, t_n)) = \bigcup \mathrm{FV}(t_i)$.

We recursively define the *formulae* over a signature Σ together with the set $\mathrm{FV}(\varphi)$ of *free variables* of each formula. (1) If $R \rightarrowtail A_1, \ldots, A_n$ is a relation symbol and $t_i : A_i$ are terms, then $R(t_1, \ldots, t_n)$ is a formula with $\mathrm{FV}(R(t_1, \ldots, t_n)) = \bigcup \mathrm{FV}(t_i)$. (2) If $s : A$ and $t : A$ are terms, then $(s =_A t)$ is a formula with

$\mathrm{FV}(s =_A t) = \mathrm{FV}(s) \cup \mathrm{FV}(t)$. (3) $\top$ and $\bot$ are formulae. Neither has free variables. (4) If φ and ψ are formulae and $\star$ is a symbol in $\{\wedge, \vee, \Rightarrow\}$, then $\varphi \star \psi$ is a formula with $\mathrm{FV}(\varphi \star \psi) = \mathrm{FV}(\varphi) \cup \mathrm{FV}(\psi)$. (5) If φ is a formula, then so is $\neg\varphi$ with $\mathrm{FV}(\neg\varphi) = \mathrm{FV}(\varphi)$. (6) If φ is a formula, then $(\exists x : A)\,\varphi$ and $(\forall x : A)\,\varphi$ are formulae. Each has $\mathrm{FV}(\varphi)/\{x\}$ as its set of free variables. Formulae constructed using (1)–(2) are *atomic*. Those built with $\top$, $\wedge$ and atomic formulae are *Horn*; with $\exists$ and Horn are *regular*; and with $\bot$, $\wedge$ and regular are *coherent*. All are *first order*. A signature with no sorts is *propositional*.

A *context* is a finite list $\vec{x} = x_1, \ldots, x_n$ of distinct variables. Its *type* is $A_1, \cdots, A_n$ where $x_i : A_i$ and its *length* is n. The *concatenation* of contexts $\vec{x}$ and $\vec{y}$ is $\vec{x}, \vec{y}\,'$ where $\vec{y}\,' = \{y \in \vec{y} \,|\, y \notin \vec{x}\}$. A context is *suitable* for a term t if $\mathrm{FV}(t) \subset \vec{x}$. If t is a term and $\vec{x}$ is a context suitable for t, then $\vec{x}.t$ is a *term-in-context*. A context is *suitable* for a formula φ if $\mathrm{FV}(\varphi) \subset \vec{x}$. If φ is a formula and $\vec{x}$ is a context suitable for φ, then $\vec{x}.\varphi$ is a *formula-in-context*. A context suitable for a term t or formula φ may include variables that do not occur in t or φ (in addition to all those that do occur). A context is *suitable* for a list $\vec{s}$ of terms if it is suitable for each $s_i \in \vec{s}$ and similarly for a list of formulae. The *canonical context* for a term t or formula φ consists of the (free) variables of t or φ in order of occurrence. We write 1 for the empty context.

2.1.2 Substitution

A *substitution* $\theta = [\vec{s}/\vec{y}]$ consists of a context $\vec{y}$ and a list $\vec{s}$ of terms having the same length and type as $\vec{y}$. The empty substitution is $[\,]$. If $\vec{z}$ is a context, the *extension* of θ to $\vec{z}$ is $\theta^{\vec{z}} = [\vec{s}, \vec{z}\,'/\vec{y}, \vec{z}\,']$ where $\vec{z}\,' = \{z \in \vec{z} \,|\, z \notin \vec{y}\}$. For example, if $\theta = [f(y), u/x, w]$ and $\vec{z} = w, z$ then $\theta^{\vec{z}} = [f(y), u, z/x, w, z]$. A context is *suitable* for a substitution if it is suitable for $\vec{s}$. The *canonical context* for a substitution $\theta = [\vec{s}/\vec{y}]$ is the canonical context for $\vec{s}$. Application of a substitution to a term is:

$$t[\vec{s}/\vec{y}] = \begin{cases} x & \text{if } t = x \text{ and } x \notin \vec{y} \\ s_i & \text{if } t = y_i \text{ for some } y_i \in \vec{y} \\ f(t_1[\vec{s}/\vec{y}], \ldots, t_n[\vec{s}/\vec{y}]) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

The substitutions $[s_i/y_i]$ are performed simultaneously. For example, $f(x,y)[x,z/y,x] = f(z,x)$. In general, this differs from a sequential application: $\big(f(x,y)[x/y]\big)[z/x] = f(x,x)[z/x] = f(z,z)$. We need not assume that $\vec{y}$ is suitable for t. If a variables of t does not occur in $\vec{y}$, no substitution is applied to it. We apply a substitution $\theta = [\vec{s}/\vec{y}]$ to a formula φ by simultaneously applying θ to all terms of φ (D1.1.4 of [14]). In the case of quantified formuale, $\varphi[\vec{s}/\vec{y}] = (Q\,u')((\varphi_0\,[u'/u])\theta)$ if $\varphi = (Q\,u)\varphi_0$ and $Q \in \{\exists, \forall\}$ where u' is a variable of the same sort as u, u' does not occur in $\vec{s}$ or $\vec{y}$, and $[u'/u]$ is applied to φ_0 before θ is applied. For example, $\big((\exists x)\,R(x,y)\big)[x/y] = (\exists x')\,R(x',x)$. Formulae are *$\alpha$-equivalent* if they differ only in the names of their bound variables. For example, $((Qu)\varphi)$ and $((Qu')(\varphi[u'/u]))$ where Q is a quantifier, $u : U$, $u' : U$ and u' does not occur in φ.

First-order inference algorithms rely on unification 4. Unification constructs a substitution θ that, when applied to two lists of expressions, makes corresponding elements equal (or at least α-equivalent). Since we employ the sequent calculus of [14], unification must apply to terms- and formulae-in-context rather than mere terms and formulae. Moreover, unification constructs θ in stages. The fragment θ_i at stage i is built without awareness of the contexts occurring in later expressions. We must, therefore, be able to apply a substitution $[\vec{s}/\vec{y}]$ to an expression-in-context $\vec{z}.e$ without $\vec{y}$ being suitable for e and without eliminating sorts that occur in $\vec{z}$.

For a term-in-context, define $(\vec{z}.t)\theta = \vec{x}.(t\,\theta^{\vec{z}})$ where $\vec{x}$ is the canonical context for $\theta^{\vec{z}}$. For example: $(x,y\,.f(x))[g(z),w\,/x,w] = (x,y\,.f(x))[g(z),w,y\,/x,w,y] = z,w,y\,.\big(f(x)[g(z),w,y\,/x,w,y]\big) = z,w,y\,.f(g(z))$ Since $\vec{z}$ is suitable for t and $\vec{x}$ is suitable for $\theta^{\vec{z}}$, $\vec{x}$ is suitable for $t\,\theta^{\vec{z}}$. Moreover, if $z : Z$ is in $\vec{z}$, then either $z \in \vec{x}$ or there is a term $s : Z$ in $(\vec{z}.t)[\vec{s}/\vec{y}]$. So, $(\vec{z}.t)\theta$ does not use the converse of weakening 5.

For a formula-in-context define:

$$(\vec{z}.\varphi)[\vec{s}/\vec{y}] = \begin{cases} \vec{x}.\varphi & \text{if } \varphi = \top \text{ or } \varphi = \bot \\ \vec{x}.(t_1\theta^{\vec{z}} = t_2\theta^{\vec{z}}) & \text{if } \varphi = (t_1 = t_2) \\ \vec{x}.R(t_1\theta^{\vec{z}}, \ldots, t_n\theta^{\vec{z}}) & \text{if } \varphi = R(t_1, \ldots, t_n) \\ \vec{x}.((\varphi_0\theta^{\vec{z}}) * (\varphi_1\theta^{\vec{z}})) & \text{if } \varphi = (\varphi_0 * \varphi_1) \text{ and } * \in \{\wedge, \vee, \Rightarrow\} \\ \vec{x}.(\neg(\varphi_0\theta^{\vec{z}})) & \text{if } \varphi = \neg\varphi_0 \\ \vec{x}.\left((Qu')(\varphi_0[u'/u]\,\theta^{\vec{z}})\right) & \text{if } \varphi = (Qu)\,\varphi_0,\ u' \notin \vec{s}, \vec{y}, \vec{z}, \text{ and } Q \in \{\exists, \forall\} \end{cases}$$

where $\vec{x}$ is the canonical context for $\theta^{\vec{z}}$. For example, $\big(y,z,w.((\exists x)\,R(x,y,z))\big)\,[f(x)/y] = \big(y,z,w.((\exists x')\,R(x',y,z)\big)\,[f(x),z,w/y,z,w] = x,z,w.((\exists x')\,R(x',f(x),z))$.

Lemma 2.1. *Lemma: Let $\vec{z}.e_1$ and $\vec{z}.e_2$ be expressions-in-context with the same context and let $\theta = [\vec{s}/\vec{y}]$ be a substitution. Then $(\vec{z}.e_1)\,\theta$ and $(\vec{z}.e_2)\,\theta$ have the same context.*

Because: $(\vec{z}.e_i)\,\theta = \vec{x}.(e_i\,\theta^{\vec{z}})$ where $\vec{x}$ is the canonical context for $\theta^{\vec{z}}$ and is independent of i.

2.1.3 Deduction

A *sequent* is an expression $(\varphi \vdash_{\vec{x}} \psi)$ where φ and ψ are formulae and $\vec{x}$ is a context suitable for both φ and ψ. A *theory* over a signature Σ is a set of sequents. A theory is classified as Horn, regular, coherent or intuitionistic according to the classification of formulae that occur in it.

For logical inference, we employ the sequent calculus of [14] shown below. Rules with a double-horizontal line may be used in either direction. We assume that a variable that occurs bound in a sequent does not also occur free in that sequent. The appendix (Section 7) includes proofs of derived rules. Fragments of classical logic are obtained by including different connectives and their sequent rules. Atomic logic has the identity, cut, substitution and Eq0. Horn logic adds Eq1 and the conjunction rules. Regular logic adds $\exists$ and the Frobenius Axiom. Coherent logic adds the disjunction and distributive rules. Intuitionistic logic adds $\Rightarrow$ and $\forall$. Classical logic adds EM. The distributive rule and Frobenius Axiom are derivable in intuitionistic logic (see 7.12 and 7.16) In regular logic we can derive the converse of Frobenius (7.13). In coherent logic we can derive the converse of the distributive rule 7.17.

Identity Rule (ID): $(\varphi \vdash_{\vec{x}} \varphi)$	Modus Ponens (Cut): $\frac{(\varphi \vdash_{\vec{x}} \psi)\ \ (\psi \vdash_{\vec{x}} \chi)}{(\varphi \vdash_{\vec{x}} \chi)}$
Substitution (Sub): $\frac{(\varphi \vdash_{\vec{x}} \psi)}{\left(\varphi[\vec{s}/\vec{x}] \vdash_{\vec{y}} \psi[\vec{s}/\vec{x}]\right)}$	Equality (Eq0): $(\top \vdash_x (x = x))$ (Eq1): $(((\vec{x} = \vec{y}) \wedge \varphi) \vdash_{\vec{z}} \varphi[\vec{y}/\vec{x}])$
True ($\top$): $(\varphi \vdash_{\vec{x}} \top)$ And Elimination ($\wedge E$): $((\varphi \wedge \psi) \vdash_{\vec{x}} \varphi)$ $((\varphi \wedge \psi) \vdash_{\vec{x}} \psi)$ And Rule ($\wedge$): $\frac{(\varphi \vdash_{\vec{x}} \psi)\ \ (\varphi \vdash_{\vec{x}} \chi)}{(\varphi \vdash_{\vec{x}} (\psi \wedge \chi))}$	False ($\bot$): $(\bot \vdash_{\vec{x}} \varphi)$ Or Introduction ($\vee I$): $(\varphi \vdash_{\vec{x}} (\varphi \vee \psi))$ $(\psi \vdash_{\vec{x}} (\varphi \vee \psi))$ Or Rule ($\vee$): $\frac{(\varphi \vdash_{\vec{x}} \chi)\ \ (\psi \vdash_{\vec{x}} \chi)}{((\varphi \vee \psi) \vdash_{\vec{x}} \chi)}$
Implication ($\Rightarrow$): $\frac{(\psi \vdash_{\vec{x}} (\varphi \Rightarrow \chi))}{((\varphi \wedge \psi) \vdash_{\vec{x}} \chi)}$	Distributive Rule: $((\varphi \wedge (\psi \vee \chi)) \vdash_{\vec{x}} (\varphi \wedge \psi) \vee (\varphi \wedge \chi))$
Existential ($\exists$) Quantification: $\frac{(\varphi \vdash_{\vec{x},y} \psi)}{((\exists y)\varphi \vdash_{\vec{x}} \psi)}$	Universal ($\forall$) Quantification: $\frac{(\varphi \vdash_{\vec{x},y} \psi)}{(\varphi \vdash_{\vec{x}} (\forall y)\psi)}$
Excluded Middle (EM): $(\top \vdash_{\vec{x}} (\varphi \vee \neg\varphi))$	Frobenius Axiom: $((\varphi \wedge (\exists y)\psi) \vdash_{\vec{x}} (\exists y)(\varphi \wedge \psi))$

2.1.4 Sequents vs Formulae

[19, 23, 24] use formulae not sequents to define theories, deduction, and algorithms for logic-based agents. Cut is defined using $\Rightarrow$ and contexts are replaced by universal quantifiers. To adapt the algorithms to less expressive fragments of first-order logic, we use the following.

Theorem 2.2. *(See D1.1.5 of [14]). In intuitionistic logic,* $(\varphi \vdash_{\vec{x}} \psi)$ *and* $(\top \vdash_1 ((\forall \vec{x})(\varphi \Rightarrow \psi)))$ *are provably equivalent. Consequently, in intuitionistic logic, we can replace sequents by formulae.*

Because:

Given a sequent:

1	$(\varphi \vdash_{\vec{x}} \psi)$	Hypothesis
2	$((\varphi \wedge \top) \vdash_{\vec{x}} \varphi)$	$\wedge E0$
3	$((\varphi \wedge \top) \vdash_{\vec{x}} \psi)$	Cut 2, 1
4	$(\top \vdash_{\vec{x}} (\varphi \Rightarrow \psi))$	$\Rightarrow 3$
5	$(\top \vdash_1 (\forall \vec{x})(\varphi \Rightarrow \psi))$	$\forall 4$

Given an implication formula:

1	$(\top \vdash_1 (\forall \vec{x})(\varphi \Rightarrow \psi))$	Hypothesis
2	$(\top \vdash_{\vec{x}} (\varphi \Rightarrow \psi))$	$\forall 1$
3	$((\varphi \wedge \top) \vdash_{\vec{x}} \psi)$	$\Rightarrow 2$
4	$(\varphi \vdash_{\vec{x}} \top)$	$\top$
5	$(\varphi \vdash_{\vec{x}} \varphi)$	ID
6	$(\varphi \vdash_{\vec{x}} (\varphi \wedge \top))$	$\wedge$ 4, 5
7	$(\varphi \vdash_{\vec{x}} \psi)$	Cut 6, 3

2.1.5 Horn Clauses vs Horn Sequents

In [23, 24], a clause is defined to be a disjunction of literals (φ or $\neg\varphi$ with φ atomic). It is a Horn clause if at most one is positive. If exactly one is positive, we can reformulate the idea using Horn sequents.

Theorem 2.3. *Given a sequent* $(\top \vdash_{\vec{x}} (\neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \psi))$ *in which* ψ *and each* φ_i *is atomic, we can derive* $((\varphi_1 \wedge \cdots \wedge \varphi_n) \vdash_{\vec{x}} \psi)$ *in intuitionistic logic.*

Because:

1	$(\top \vdash_{\vec{x}} (\neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \psi))$	Hypothesis
2	$((\neg\varphi_1 \vee \cdots \vee \neg\varphi_n) \vdash_{\vec{x}} \neg(\varphi_1 \wedge \cdots \wedge \varphi_n))$	Theorem 7.34
3	$((\neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \psi) \vdash_{\vec{x}} (\neg(\varphi_1 \wedge \cdots \wedge \varphi_n) \vee \psi))$	Theorem 7.15
4	$(\top \vdash_{\vec{x}} \neg(\varphi_1 \wedge \cdots \wedge \varphi_n) \vee \psi)$	Cut 1, 3
5	$(\neg(\varphi_1 \wedge \cdots \wedge \varphi_n) \vee \psi) \vdash_{\vec{x}} ((\varphi_1 \wedge \cdots \wedge \varphi_n) \Rightarrow \psi))$	Theorem 7.24 (no EM)
6	$(\top \vdash_{\vec{x}} ((\varphi_1 \wedge \cdots \wedge \varphi_n) \Rightarrow \psi))$	Cut 4, 5
7	$((\top \wedge (\varphi_1 \wedge \cdots \wedge \varphi_n)) \vdash_{\vec{x}} \psi)$	$\Rightarrow 6$
8	$((\varphi_1 \wedge \cdots \wedge \varphi_n) \vdash_{\vec{x}} \psi)$	Theorem 7.9

If $\psi = \neg\psi'$ we obtain $((\varphi_1 \wedge \cdots \wedge \varphi_n \wedge \psi) \vdash_{\vec{x}} \bot)$ or $((\varphi_1 \wedge \cdots \wedge \varphi_n) \vdash_{\vec{x}} \neg\psi')$ neither of which is Horn.

2.1.6 Normal Form for Horn Theories

Theorem 2.4. *(See D.1.3.10 of [14]). Any Horn sequent is provably equivalent to a list of sequents of the form* $((\varphi_1 \wedge \cdots \wedge \varphi_n) \vdash_{\vec{x}} \psi)$ *where each* φ_i *and* ψ *is either atomic or* $\top$.

Because: Given $((\varphi_1 \wedge \cdots \wedge \varphi_n) \vdash_{\vec{x}} (\psi_1 \wedge \cdots \wedge \psi_m))$, by $\wedge E$, we can derive $((\psi_1 \wedge \cdots \wedge \psi_m) \vdash_{\vec{x}} \psi_k)$ for $1 \leq k \leq m$. Cut then yields $((\varphi_1 \wedge \cdots \wedge \varphi_n) \vdash_{\vec{x}} \psi_k))$. Conversely, if we have a list of sequents in normal form, we can combine them into one sequent using $\wedge$. □

Applying the proof above to each sequent in a Horn theory, we obtain a Corollary: Every Horn theory is provably equivalent to a Horn theory in which every sequent is in normal form. Algorithm 13 in the appendix implements Theorem 2.4. The output sequents are distinct and all have the same context $\vec{x}$. We may convert a Horn theory to normal form by applying Algorithm 13 to its sequents and eliminating redundancies. See Algorithm 14.

2.2 Semantics

Signatures and theories can be assigned interpretations in suitable categories [14].

2.2.1 Σ-Structures

Let Σ be a signature. A Σ-*structure* in a category $\mathscr{C}$ with finite products consists of functions assigning (1) an object $M(A)$ to each sort, (2) a morphism $M(f) : M(A_1) \times \cdots \times M(A_n) \to M(B)$ to each function symbol $f : A_1, \ldots, A_n \to B$, and (3) a subobject $M(R) \longmapsto M(A_1) \times \cdots \times M(A_n)$ to each relation symbol $R \rightarrowtail A_1, \ldots, A_n$. In particular, the empty type $[]$ has $M([]) =$ the terminal object and so a constant $f : [] \to B$ is interpreted as a *point* $1 \to M(B)$ and a proposition $R \rightarrowtail 1$ is a *truth value* $M(R) \longmapsto 1$.

2.2.2 Terms- and Formulae-in Context

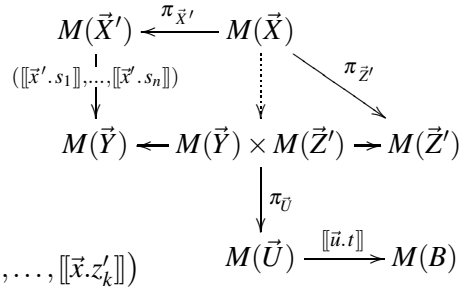
Σ-structures can be extended to terms- and formulae-in context (D.1.2.3 and D.1.2.7 of [14]). Given $\vec{x}.t$ with $\vec{x} : X_1, \ldots, X_m$ and $t : B$, then $[\![\vec{x}.t]\!] : M(X_1) \times \cdots \times M(X_m) \to M(B)$ is a morphism. If $\vec{x}.\varphi$ is a formula-in-context, then $[\![\vec{x}.\varphi]\!]_M \leq M(X_1) \times \cdots \times M(X_m)$ is a subobject. Semantics of the connectives are implemented via operations in suitable categories (D1.2.6 of [14]). Although the algorithms in this paper are syntactic, examples in 1 and 5 rely on the fact that $\wedge$ is computed as a pullback. A sequent $\sigma = (\varphi \vdash_{\vec{x}} \psi)$ is *satisfied* in M if $[\![\vec{x}.\varphi]\!]_M \leq [\![\vec{x}.\psi]\!]_M$. A theory $\mathbb{T}$ is *satisfied* in M if all its sequents are. D.1.3.2 and D.1.4.11 of [14] provide soundness and completeness theorems for categorical semantics.

2.2.3 Substitution

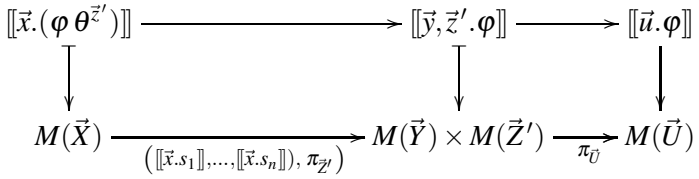
Semantics of substitution into terms and terms-in-context are computed by composition while semantics of substitution into formulae and formulae-in-context are computed by pullback. The properties for terms and formula (D1.2.4 and D1.2.7 of [14]) are included in the appendix of this paper 7.3.

Theorem 2.5. *Substitution Property for Terms-in-Context: If $\vec{z}.t$ is a term-in-context and $\theta = [\vec{s}/\vec{y}]$ is a substitution, then $[\![(\vec{z}.t)\theta]\!] = [\![\vec{x}.(t\,\theta^{\vec{z}})]\!] = [\![\vec{u}.t]\!] \circ \pi_U \circ \big(([\![\vec{x}'.s_1]\!], \ldots, [\![\vec{x}'.s_n]\!]) \circ \pi_{\vec{X}'},\ \pi_{\vec{Z}'} \big)$ where $\vec{z}' = \{z \in \vec{z} \,|\, z \notin \vec{y}\}$, $\vec{u}$ and $\vec{x}'$ are the canonical contexts for t and $\vec{s}$, and π_U and $\pi_{Z'}$ are projections.*

Because: The definition of substitution justifies the first equality of $[\![(\vec{z}.t)[\vec{s}/\vec{y}]]\!] = [\![\vec{x}.(t[\vec{s},\vec{z}'/\vec{y},\vec{z}'])]\!] = [\![\vec{y},\vec{z}'.t]\!] \circ \big([\![\vec{x}.s_1]\!], \ldots, [\![\vec{x}.s_n]\!], [\![\vec{x}.z'_1]\!], \ldots, [\![\vec{x}.z'_k]\!]\big)$. The Substitution Property for Terms gives the second. If $\vec{u} \subset \vec{y},\vec{z}'$ is the canonical context for t, then the Weakening Property (D1.2.4 of [14]) implies $[\![\vec{y},\vec{z}'.t]\!] = [\![\vec{u}.t]\!] \circ \pi_{\vec{U}}$. Similarly, if $\vec{x}'$ is canonical for $\vec{s}$, then $([\![\vec{x}.s_1]\!], \ldots, [\![\vec{x}.s_n]\!]) = ([\![\vec{x}'.s_1]\!], \ldots, [\![\vec{x}'.s_n]\!]) \circ \pi_{\vec{X}'}$. $[\![\vec{x}.z'_j]\!]$ is a projection, hence, $[\![\vec{y},\vec{z}'.t]\!] \circ \big([\![\vec{x}.s_1]\!], \ldots, [\![\vec{x}.s_n]\!], [\![\vec{x}.z'_1]\!], \ldots, [\![\vec{x}.z'_k]\!]\big)$ $= [\![\vec{u}.t]\!] \circ \pi_U \circ \big(\big([\![\vec{x}'.s_1]\!], \ldots, [\![\vec{x}'.s_n]\!]\big) \circ \pi_{\vec{X}'}, \pi_{\vec{Z}'} \big)$. □



Theorem 2.6. *Substitution Property for Formulae-in-Context: If $\vec{z}.\varphi$ is a formula-in-context and $\theta = [\vec{s}/\vec{y}]$ is a substitution, then $[\![(\vec{z}.\varphi)\theta]\!] = [\![\vec{x}.(\varphi\,\theta^{\vec{z}'})]\!]$ is computed as pullback where $\vec{u}$ is the canonical context for φ.*



Because: This follows from the Substitution Property for Formulae (D1.2.7 of [14]).

3 Forward Chaining for Propositional Horn Theories

Algorithm 1 determines if a Horn sequent $\sigma = ((R_1 \wedge \cdots \wedge R_k) \vdash_1 S)$ in normal form is derivable in a propositional Horn theory $\mathbb{T}$. It adapts the formula-based algorithm of [24] by replacing Horn clauses with Horn sequents (see 2.1.5). It maintains a queue $\mathscr{Q}$ of proposition symbols and makes successive passes through the $\mathbb{T}$-axioms. Symbols U that have occurred in $\mathscr{Q}$ are the right sides of derived sequents $((R_1 \wedge \cdots \wedge R_k) \vdash_1 U)$. Algorithm 1 and the formula-based algorithm of [24] differ in the way the queue is initialized. When using sequents, there are two sources for the initial queue. Each R_i is in the initial queue since $((R_1 \wedge \cdots \wedge R_k) \vdash_1 R_i)$ is derivable by $\wedge E$. The second source of symbols in the initial queue is right sides of $\mathbb{T}$-axioms of the form $(\top \vdash_1 U)$ since we can apply cut to the sequent rule $((R_1 \wedge \cdots \wedge R_k) \vdash_1 \top)$ and the axiom. In each pass, we have, in effect, derived $((R_1 \wedge \cdots \wedge R_k) \vdash_1 (\wedge\mathscr{Q}))$. We then consider each axiom $(\psi \vdash_1 P)$ of $\mathbb{T}$ and seek to apply $\wedge E$ to derive $((\wedge\mathscr{Q}) \vdash_1 \psi)$. If this is possible, we apply cut to derive $((R_1 \wedge \cdots \wedge R_k) \vdash_1 P)$. This adds P to $\mathscr{Q}$.

Algorithm 1 Determine if $\sigma = ((R_1 \wedge \cdots \wedge R_k) \vdash_1 S)$ is derivable in a propositional Horn theory $\mathbb{T}$.

```
1:  procedure PROPOSITIONAL-FORWARD-CHAINING(𝕋, σ)
2:      n_s ← the number of sequents σ_i = ((P_1^i ∧ ··· ∧ P_{n_i}^i) ⊢_1 Q_i) in 𝕋
3:      n_p ← the number of proposition symbols in σ and in the axioms of 𝕋
4:      queue ← a queue containing R_1, ..., R_k and all Q_i for which (P_1^i ∧ ··· ∧ P_{n_i}^i) = ⊤
5:      inferred ← an array of size n_p with inferred[U] = false for all U
6:      count ← an array of size n_s with count[i] = the number of proposition symbols on the left of σ_i
7:      while queue is not empty do
8:          U ← pop(queue)
9:          if U = Q then return true
10:         if inferred[U] = false then
11:             inferred[U] ← true
12:             for i ← 0 to n_s − 1 do
13:                 if U occurs on the left side of of σ_i then
14:                     count[i] ← count[i] − 1
15:                     if count[i] = 0 then push U onto queue
16:     return false
```

Consider a theory with axioms (1) $((A \wedge B) \vdash_1 D)$ and (2) $((C \wedge D) \vdash_1 E)$. The figure below shows how Algorithm 1 yields a derivation of $((A \wedge B \wedge C) \vdash_1 E)$. Each horizontal line is an entry into the while loop. The $\mathscr{Q}$ history is written as a conjunction. The underlined symbol is no longer in $\mathscr{Q}$. U is the popped symbol. Count indicates the sequent σ_i for which count is decremented. Inferred is set to true for the underlined symbol in σ_i. The Derivation column indicates the derived sequents.

$\mathscr{Q}$ History	U	Count	Derivation	
$\underline{A} \wedge B \wedge C$	A	(1) $((\underline{A} \wedge B) \vdash_1 D)$		
$\underline{A} \wedge \underline{B} \wedge C$	B	(1) $((\underline{A} \wedge \underline{B}) \vdash_1 D)$	1. $((A \wedge B \wedge C) \vdash_1 (A \wedge B))$	$\wedge$ E
			2. $((A \wedge B) \vdash_1 D)$	Axiom 1
			3. $((A \wedge B \wedge C) \vdash_1 D)$	Cut 1, 2
$\underline{A} \wedge \underline{B} \wedge \underline{C} \wedge D$	C	(2) $((\underline{C} \wedge D) \vdash_1 E)$	4. $((A \wedge B \wedge C) \vdash_1 (A \wedge B \wedge C))$	Id
			5. $((A \wedge B \wedge C) \vdash_1 (A \wedge B \wedge C \wedge D))$	$\wedge I$ 4, 3
$\underline{A} \wedge \underline{B} \wedge \underline{C} \wedge \underline{D}$	D	(2) $((\underline{C} \wedge \underline{D}) \vdash_1 E)$	6. $((A \wedge B \wedge C \wedge D) \vdash_1 (C \wedge D))$	$\wedge$E
			7. $((C \wedge D) \vdash_1 E)$	Axiom 2

$\mathscr{Q}$ History	U	Count	Derivation	
			8. $((A \wedge B \wedge C \wedge D) \vdash_1 E)$	Cut 6, 7
			9. $((A \wedge B \wedge C) \vdash_1 E)$	Cut 5, 8

$\underline{A} \wedge \underline{B} \wedge \underline{C} \wedge \underline{D} \wedge \underline{E} \quad E$

4 Unification of Terms- and Formulae-in Context

A *unification* of lists $[\vec{x}_1.\alpha_1,\ldots,\vec{x}_n.\alpha_n]$ and $[\vec{y}_1.\beta_1,\ldots,\vec{y}_n.\beta_n]$ of terms-in-context is a substitution θ for which $(\vec{x}_i.\alpha_i)\theta = (\vec{y}_i.\beta_i)\theta$ for $1 \le i \le n$. For example, $\theta = [u,b,k,a,f(g(z)),g(z)\,/\,u,b,k,a,x,y]$ unifies $[u,x.g(x),\ a,y.f(y)]$ and $[b,x,y.g(f(y)),\ k,z.f(g(z))]$ since

$$\begin{aligned}
(u,x.g(x))\theta &= u,b,k,a,z.(g(x)\theta) &&= u,b,k,a,z.g(f(g(z)))\\
(b,x,y.g(f(y)))\theta &= u,b,k,a,z.(g(f(y))\theta) &&= u,b,k,a,z.g(f(g(z)))\\
(a,y.f(y)\theta &= u,b,k,a,z.(f(y)\theta) &&= u,b,k,a,z.f(g(z))\\
(k,z.f(g(z))\theta &= u,b,k,a,z.(f(g(z))\theta^z) &&= u,b,k,a,z.f(g(z))
\end{aligned}$$

A unification of lists of formulae-in-context is defined similarly. Unification is an essential subroutine for inference using fragments of first-order logic. We adapt the procedures of [24] and [28] to support (1) multi-sorted signatures and (2) terms- and formulae-in-context rather than terms and formulae without a context. We must take contexts into account in order to correctly apply the substitution rule. Unification algorithms taking into account only (1) are included in 7.1.9 and 7.1.10. Algorithms 7, 8, 9, 10, 11, and 12 of the appendix apply substitutions to terms, terms-in-context, formulae and formulae-in-context.

Theorem 4.1. *If Algorithm 2 returns a substitution θ, then $(\vec{x}_i.\,\alpha_i)\,\theta = (\vec{y}_i.\,\beta_i)\theta$ for $1 \le i \le\ n$.*

Because: First note that $(\vec{x}_1.\,\alpha_1)\theta_1 = (\vec{y}_1.\,\beta_1)\theta_1$:

Case A: Applying θ_1 concatenates $\vec{x}_1$ and $\vec{y}_1$ without x then transforms both terms-in-context to $\vec{z},x.x$.

Case B: Applying θ_1 concatenates $\vec{x}_1$ and $\vec{y}_1$ without x or y then transforms both terms to $\vec{z},y.y$.

Case C: The terms-in-context are $\vec{x}_1.x$ and $\vec{y}_1.g(\vec{t})$. $\vec{z}$ consists of all variables of $\vec{x}_1$ and $\vec{y}_1$ except x and $\mathrm{FV}(\vec{t})$. $\theta_1 = [\vec{z},g(\vec{t})/\vec{z},x]$. Since $(\vec{x}_1.x)\theta_1 = \vec{z},\mathrm{FV}(\vec{t}).(x[\vec{z},g(\vec{t}),\mathrm{FV}(\vec{t})/\vec{z},x,\mathrm{FV}(\vec{t})]) = \vec{z},\mathrm{FV}(\vec{t}).g(\vec{t})$ and $(\vec{y}_1.g(\vec{t}))\theta_1 = \vec{z},\mathrm{FV}(\vec{t}).(g(\vec{t})[\vec{z},g(\vec{t}),\mathrm{FV}(\vec{t})/\vec{z},x,\mathrm{FV}(\vec{t})]) = \vec{z},\mathrm{FV}(\vec{t}).g(\vec{t})$, we have $(\vec{x}_1.\alpha_1)\,\theta_1 = (\vec{y}_1.\beta_1)\,\theta_1$.

Case D: The procedure recursively calls itself with $[\vec{y}_1.\beta_1,\ldots]$ and $[\vec{x}_1.\alpha_1,\ldots]$ without removing terms or applying a substitution.

Case E: If the function symbols agree, the procedure recursively calls itself without removing terms or applying a substitution.

Consequently, $(\vec{x}_1.\,\alpha_1)\theta = (\vec{y}_1.\,\beta_1)\theta$. For each case A–E, if a substitution θ_i is appended to θ, then it is applied to all subsequent terms-in-context in the recursive call. Hence, when $\vec{x}_i.\,\alpha_i$ and $\vec{y}_i.\,\beta_i$ appear in the first terms in the argument lists, the algorithm seeks to unify $(\vec{x}_i.\,\alpha_i)\theta_1\cdots\theta_\ell$ and $(\vec{y}_i.\,\beta_i)\theta_1\cdots\theta_\ell$ for some $\ell \ge 0$. By induction, if θ_* is appended to θ, we have $(\vec{x}_i.\,\alpha_i)\theta_1\cdots\theta_\ell\,\theta_* = (\vec{y}_i.\,\beta_i)\theta_1\cdots\theta_\ell\,\theta_*$. It follows that $(\vec{x}_i.\,\alpha_i)\,\theta = (\vec{y}_i.\,\beta_i)\,\theta$. □

Lemma 4.2. *If Algorithm 2 returns a substitution θ, then $(\vec{x}_1.\,\alpha_1)\theta$ and $(\vec{x}_2.\,\alpha_2)\theta$ have the same context.*

Because: If u is a variable in $(\vec{x}_1.\alpha_1)\,\theta$ but not in $(\vec{x}_1.\alpha_1)\,\theta$, then there is a minimum j for which $u \in (\vec{x}_1.\alpha_1)\,\theta_1\cdots\theta_j$ but $u \notin (\vec{x}_2.\alpha_2)\,\theta_1\cdots\theta_j$ where $\theta = \theta_1\cdots\theta_k$. If $1 \le j \le k$, then $(\vec{x}_1.\alpha_1)\,\theta_1\cdots\theta_{j-1}$ and $(\vec{x}_2.\alpha_2)\,\theta_1\cdots\theta_{j-1}$ both contain u or neither does. But θ_j will either add u to both contexts or to neither. If $j = 0$, then $u \in (\vec{x}_1.\alpha_1)$ but $u \notin (\vec{x}_2.\alpha_2)$. If θ_1 is generated by Case A then θ_1 does not remove any variables, hence $u \in (\vec{x}_2.\alpha_2)$. Cases B and C: If θ_1 adds u to contexts, then $u \in (\vec{x}_2.\alpha_2)$. If θ_1 substitutes some y for u, then $u \notin (\vec{x}_1.\alpha_1)\theta_1$. Cases D and E: No substitution is applied. □

Algorithm 2 Find a unification θ of two lists of terms-in-context

procedure UNIFY-TERMS-IN-CONTEXT($[\vec{x}_1.\alpha_1,\ldots,\vec{x}_m.\alpha_m]$, $[\vec{y}_1.\beta_1,\ldots,\vec{y}_n.\beta_n]$, $\theta=[]$)
  if $m\neq n$ **then return** Null
  else if $m=0$ **then return** θ
  else if $\mathrm{sort}(\alpha_1)\neq\mathrm{sort}(\beta_1)$ **then return** Null
  else if $\alpha=x:X$ is a variable **then**
    if $\beta_1=x$ **then** ▷ Case A
      $\vec{z}\leftarrow \mathrm{union}(\vec{x}_1,\vec{y}_1)/\{x\}$
      $\theta_1\leftarrow[\vec{z},x/\vec{z},x]$
      return UNIFY-TERMS-IN-CONTEXT($[(\vec{x}_2.\alpha_2)[\theta_1],\ldots,]$, $[(\vec{y}_2.\beta_2)[\theta_1],\ldots]$, $\theta\theta_1$)
    else if $\beta_1=y\neq x$ **then** ▷ Case B
      $\vec{z}\leftarrow \mathrm{union}(\vec{x}_1,\vec{y}_1)/\{x,y\}$
      $\theta_1\leftarrow[\vec{z},y/\vec{z},x]$
      return UNIFY-TERMS-IN-CONTEXT($[(\vec{x}_2.\alpha_2)[\theta_1],\ldots,]$, $[(\vec{y}_2.\beta_2)[\theta_1],\ldots]$, $\theta\theta_1$)
    else $\beta_1=g(t_1,\ldots,t_\ell)$ ▷ Case C
      if $\alpha_1=x$ occurs in β_1 **then return** Null
      else
        $\vec{z}\leftarrow \mathrm{union}(\vec{x}_1,\vec{y}_1)/\{x$ and the canonical context of $\vec{t}\,\}$
        $\theta_1\leftarrow[\vec{z},g(\vec{t})/\vec{z},x]$
        return UNIFY-TERMS-IN-CONTEXT($[(\vec{x}_2.\alpha_2)[\theta_1],\ldots,]$, $[(\vec{y}_2.\beta_2)[\theta_1],\ldots]$, $\theta\theta_1$)
  else $\alpha_1=f(s_1,\ldots,s_k)$
    if β_1 is a variable **then** ▷ Case D
      return UNIFY-TERMS-IN-CONTEXT($[\vec{y}_1.\beta_1,\vec{x}_2.\alpha_2,\ldots]$, $[\vec{x}_1.\alpha_1,\vec{y}_2.\beta_2,\ldots]$, θ)
    else $\beta_1=g(t_1,\ldots,t_\ell)$ ▷ Case E
      if $f\neq g$ **then return** Null
      else
        return UNIFY-TERMS-IN-CONTEXT($[\vec{x}_1.s_1,\ldots,\vec{x}_2.\alpha_2,\ldots]$, $[\vec{y}_1.t_1,\ldots,\vec{y}_2.\beta_2,\ldots]$, θ)

Lemma 4.3. *If Algorithm 2 returns θ, then $(\vec{x}_i.\,\alpha_i)\theta$ and $(\vec{x}_j.\,\alpha_j)\theta$ have the same context for all i, j.*

Because: We may assume without loss of generality that $i<j$. As an intermediate step in the procedure we will reach a recursive call to unify $[(\vec{x}_i.\,\alpha_i)\theta_*,\ (\vec{x}_{i+1}.\,\alpha_{i+1})\theta_*,\ \ldots]$ and $[(\vec{y}_i.\,\beta_i)\theta_*,\ (\vec{y}_{i+1}.\,\beta_{i+1})\theta_*,\ \ldots]$ where θ_* is the part of θ constructed so far. Since we assume the algorithm returns a substitution, the algorithm will proceed to construct the remaining part θ' of $\theta=\theta_*\,\theta'$. By Lemma 4.2, $(\vec{x}_i.\,\alpha_i)\theta_*\,\theta'$ and $(\vec{x}_{i+1}.\,\alpha_{i+1})\theta_*\theta'$ have the same context. The result follows by induction. □

Theorem 4.4. *If Algorithm 2 returns θ, then $(\vec{x}_i.\alpha_i)\,\theta$ and $(\vec{y}_j.\beta_j)\,\theta$ have the same context for all i, j.*

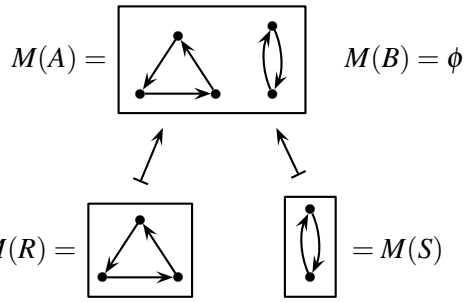
Because: By Lemma 4.3, all $(\vec{x}_i.\,\alpha_i)\,\theta$ have the same context. By Theorem 4.1, $(\vec{x}_i.\,\alpha_i)\,\theta=(\vec{y}_i.\,\beta_i)\,\theta$, hence, they have the same context. □

Consider, for example: UNIFY-TERMS-IN-CONTEXT$([x,u.\,g(x),\ a,y.f(y)],\ [b,x,y.\,g(f(y)),\ k,z.f(g(z))])$. Case E results in UNIFY-TERMS-IN-CONTEXT$([x,u.x,\ \ a,y.\,f(y)],[b,x,y.\,f(y),\ \ k,z.\,f(g(z))])$. Case C results in $\theta_1=[u,b,f(y)\,/\,u,b,x]$. Next call UNIFY-TERMS-IN-CONTEXT$([(a,y.f(y))\theta_1],\ [(k,z.f(g(z)))\theta_1])$. Evaluating the arguments yields: $(a,y.f(y))\theta_1=a,b,y,a.\,(f(y)[u,b,f(y),a,y\,/\,a,b,x,a,y])=u,b,y,a.\,f(y)$ and $(k,z.f(g(z)))\theta_1=u,b,y,k,z.\,(f(g(z))[u,b,f(y),k,z\,/\,u,b,x,k,z])=u,b,y,k,z.\,f(g(z))$. Case E results in UNIFY-TERMS-IN-CONTEXT$([u,b,y,a.y],\ [u,b,y,k,z.g(z)]$. Case C then yields $\theta_2=[u,b,k,a,g(z)\,/\,u,b,k,a,y]$. The resulting substitution is $\theta_1\,\theta_2=[u,b,k,a,f(g(z)),g(z)\,/\,u,b,k,a,x,y]$.

Algorithm 3 Find a unification θ of two lists of formulae-in-context.

procedure UNIFY-FORMULAE-IN-CONTEXT($[\vec{x}_1.\alpha_1,\ldots,\vec{x_m}.\alpha_m]$, $[\vec{y}_1.\beta_1,\ldots,\vec{y}_m\beta_n], \theta = []$))
  if $m \neq n$ **then return** Null
  else if $m = 0$ **then return** θ
  else if $\alpha_1 = (s_1 = s_2)$ and $\beta_1 = (t_1 = t_2)$ **then**
    $\theta' =$ UNIFY-TERMS-IN-CONTEXT($[\vec{x}_1.s_1,\ \vec{x}_1.s_2]$, $[\vec{y}_1.t_1, \vec{y}_1.t_2]$, θ)
    return UNIFY-FORMULAE-IN-CONTEXT($[(\vec{x}_2.\alpha_2)\,\theta', \ldots]$, $[(\vec{y}_2.\beta_2)\,\theta', \ldots]$, $\theta\,\theta'$)
  else if $\alpha_1 = Q(s_1,\ldots,s_k)$ and $\beta_1 = R(t_1,\ldots,t_\ell)$ **then**
    if $Q \neq R$ **then return** Null
    else
      $\theta' =$ UNIFY-TERMS-IN-CONTEXT($[\vec{x}_1.s_1,\ldots,\vec{x}_1.s_k]$, $[\vec{y}_1.t_1,\ldots,\vec{y}_1.t_k], \theta$)
      return UNIFY-FORMULAE-IN-CONTEXT($[(\vec{x}_2.\alpha_2)\theta', \ldots,]$, $[(\vec{y}_2.\beta_2)\,\theta', \ldots], \theta\,\theta'$)
  else if $\alpha_1 = \varphi * \varphi'$ and $\beta_1 = \psi * \psi'$ with $* = \wedge$, $\vee$ or $\Rightarrow$ **then**
    return UNIFY-FORMULAE-IN-CONTEXT($[\vec{x}_1.\varphi, \vec{x}_1.\varphi', \vec{x}_2.\alpha_2, \ldots]$, $[\vec{y}_1.\psi, \vec{y}_1.\psi', \vec{y}_2.\beta_2, \ldots]$, θ)
  else if $\alpha_1 = \neg\varphi$ and $\beta_1 = \neg\psi$ **then**
    return UNIFY-FORMULAE-IN-CONTEXT($[\vec{x}_1.\varphi, \vec{x}_2.\alpha_2, \ldots]$, $[\vec{y}_1.\psi, \vec{y}_2.\beta_2, \ldots]$, θ)
  else if $\alpha_1 = ((Q.x)\varphi)$ and $\beta_1 = ((Q.y)\psi)$ with $Q = \exists$ or $\forall$ **then**
    if sort$(x) \neq$ sort(y) **then return** Null
    else
      Let u_1, u_2, and u_3 be variables of sort(x) distinct from those in $\vec{x}_1$, $\vec{y}_1$, x, y, and each other.
      if $x = y$ **then**
        let $u_2 = u_3$.
      return UNIFY-FORMULAE-IN-CONTEXT(
          $[(\vec{x}_1, u_1.\varphi[u_1/x])[u_3/y]$, $(\vec{x}_2.\alpha_2)[u_2,u_3/x,y], \ldots]$,
          $[(\vec{y}_1, u_1.\psi[u_1/y])[u_2/x]$, $(\vec{y}_2.\beta_2)[u_2,u_3/x,y], \ldots]$, $\theta[u_2,u_3/x,y]$)
  return Null

5 Detecting Closed Sorts



The substitution rule includes the weakening rule: from $(\varphi \vdash_{\vec{x}} \psi)$ derive $\left(\varphi \vdash_{\vec{y}} \psi\right)$ where $\vec{y}$ contains all the variables of $\vec{x}$ and possibly others. That is, we are free to introduce new variables into a context. Since $\vec{x}$ must be suitable for φ and ψ, no new variable of $\vec{y}$ is free in either formula. The converse of weakening is not a permitted inference rule. There are two exceptions. Let $y : A$ be a variable that occurs in $\vec{y}$ but not $\vec{x}$. (1) If $x : A$ occurs in $\vec{x}$, then we may substitute $[x/y]$ into $\left(\varphi \vdash_{\vec{y}} \psi\right)$. (2) If there is a closed term $k : Y$, then we may substitute $[k/y]$ into $\left(\varphi \vdash_{\vec{y}} \psi\right)$. In either case, we eliminate y from $\vec{y}$.

As a counterexample to the converse of weakening, let $\mathbb{T}$ be the Horn theory with sorts A and B, relations $S \rightarrowtail A$ and $R \rightarrowtail A$ and axiom $(R(a) \vdash_{a,b} S(a))$ where $a : A$ and $b : B$. Construct a model M of $\mathbb{T}$ in Set$^{\circlearrowleft}$ as shown above. $[\![a,b.R(a)]\!]$ and $[\![a,b.S(a)]\!]$ are subobjects of $M(A) \times M(B) = \phi$, hence, both equal ϕ. Consequently, $M \models (R(a) \vdash_{a,b} S(a))$. However, $[\![a.R(a)]\!] \cong M(R)$ and $[\![a.S(a)]\!] \cong M(S)$ in $M(A)$. Since there is no Set$^{\circlearrowleft}$ morphism from a 3-cycle to a 2-cycle, $M \not\models (R(a) \vdash_a S(a))$. If $(R(a) \vdash_a S(a))$ were derivable in $\mathbb{T}$ this would contradict the Soundness Theorem (Proposition D.1.3.2 of [14]).

An application of the special cases of the converse to weakening arises in Algorithm 5. If $y : A$ is in $\vec{y}$, it is simple to determine if there is an $x : A$ in $\vec{x}$. Algorithm 4 determines if there is a closed term $k : A$.

Algorithm 4 Determine if a sort A of a signature Σ is closed.

```
1:  procedure SORT-CLOSED?(Σ, A)
2:      n_s ← the number of sorts in Σ
3:      n_f ← the number of function symbols in Σ
4:      queue ← a queue containing the sorts S of Σ for which Σ has a constant k : 1 → S
5:      closed ← an array of size n_s with closed[i] = false
6:      count ← an array of size n_f with count[i] = the arity of f_i
7:      while queue is not empty do
8:          S ← pop(queue)
9:          if S = A then
10:             return true
11:         if closed[S] = false then
12:             closed[S] ← true
13:             for i ← 0 to n_f − 1 do
14:                 if S occurs in the type of f_i then
15:                     count[i] ← count[i] − 1
16:                     if count[i] = 0 then push S onto queue
17:     return false
```

Algorithm 4 is similar to Algorithm 1 : identify sorts in the former with proposition symbols in the latter, constants with axioms $(\top \vdash_1 R)$ and function symbols with sequents.

6 Forward Chaining for First-Order Horn Theories

First-order inference involves applying sequent rules for the relevant fragment of logic and discovering substitutions that allow the inference to proceed. Before applying cut to derive of $(\top \vdash_{x_3} C(x_3))$ from axioms $(\top \vdash_{x_1} A(x_1))$ and $(A(x_2) \vdash_{x_2} B(x_2))$, for example, we apply $[x_3/x_1]$ and $[x_3/x_2]$ to the axioms. We may also derive $(\top \vdash_{x_3,w} C(x_3))$ by applying the weakening substitution $[x_3,w/x_3,w]$ regardless of the sort of w. If, however, the first axiom were $(\top \vdash_{x_1,y} A(x_1))$, it is not clear that we can answer the original query $(\top \vdash_{x_3} C(x_3))$ since y must be eliminated.

Algorithm 5 performs inference in first-order Horn theories. It uses unification to discover substitutions and, if necessary, attempts to use the methods of Section 5 to eliminate variables.

Consider a theory $\mathbb{T}$ with axioms (1) $(A(x_1) \vdash_{x_1,y} B(x_1))$, (2) $((B(x_2) \wedge C(x_2)) \vdash_{x_2,w_1} D(x_2,w_1))$, and (3) $(\top \vdash_{x_3} A(x_3))$. We seek to derive the sequent $(C(x_4) \vdash_{x_4,w_2,z} D(x_4,w_2))$ where $w_i : W$, $x_i : X$, $y : Y$ and $z : Z$. In applying Algorithm 5, the initial queue has $x_3.A(x_3)$ and $x_4,w_2,z.C(x_4)$. The first pass through the while loop unifies $[x_1,y.A(x_1)]$ and $[x_3.A(x_3)]$ by discovering $\theta_A = [y,x_3/y,x_1]$. This adds $y,x_3.B(x_3)$ to the queue. The second pass unifies $[x_2,w_1.B(x_2),\ x_2,w_1.C(x_2)]$ and $[y,x_3.B(x_3),\ x_4,w_2,z.C(x_4)]$ by discovering $\theta_1 = [w_1,y,x_3/w_1,y,x_2]$ then $\theta_2 = [w_1,y,z,w_2,x_4/w_1,y,z,w_2,x_3]$. These compose to give the substitution $\theta_B = [w_1,y,x_4,z,w_2,x_4/w_1,y,x_2,z,w_2,x_3]$. This adds $w_1,y,x_4,z,w_2.D(x_4,w_1)$ to the queue. Since line 17 discovers that this unifies with the goal $x_4,w_2,z.D(x_4,w_2)$, we seek to reconcile the formula-in-context with the goal. w_1 and w_2 have the same sort so the substitution $[w_2/w_1]$ leaves only y as a variable in the derived context that does not occur in the goal. We then apply Algorithm 4 to determine

if Y is a closed sort in which case we could eliminate y using a substitution $[k/y]$ with k a constant.

Algorithm 5 Determine if a Horn sequent $\sigma = \big((R_1 \wedge \cdots \wedge R_k) \vdash_{\vec{y}} S\big)$ in normal form is derivable in a propositional Horn theory $\mathbb{T}$ in normal form with sequents $\big((P_1{}^i \wedge \cdots \wedge P^i_{n_i}) \vdash_{\vec{x}^i} Q^i\big)$ for $1 \leq i \leq m$.

```
1:  procedure FORWARD-CHAINING (𝕋, σ)
2:      queue ← a queue containing y⃗.R_1, ..., y⃗.R_k and all x⃗^i.Q^i for which (P_1^i ∧ ··· ∧ P_{n_i}^i) = ⊤
3:      new ← true
4:      while new = true do
5:          new ← false
6:          for i ← 1 to m do
7:              for each list u⃗_1.φ_1, ..., u⃗_{n_i}.φ_{n_i} from the queue
8:                  θ ← UNIFY([u⃗_1.φ_1, ..., u⃗_{n_i}.φ_{n_i}], [x⃗^i.P_1^i, ..., x⃗^i.P_{n_i}^i]) do
9:                  if θ ≠ Null then
10:                     v⃗.Q' ← (x⃗^i.Q^i)θ
11:                     for u⃗.φ in the queue do
12:                         γ ← UNIFY([u⃗.φ], [v⃗.Q'])
13:                         if γ ≠ Null then
14:                             add v⃗.Q' to the queue
15:                             new ← true
16:                     if new = true then
17:                         δ ← UNIFY([y⃗.S], [v⃗.Q'])
18:                         if δ ≠ null then
19:                             z⃗.S = (v⃗.Q')δ
20:                             reconcilable ← true
21:                             for z : Z with z ∈ z⃗ and z ∉ y⃗ do
22:                                 if Z is not a closed sort and y⃗ has no y : Z then
23:                                     reconcilable ← false
24:                             if reconcilable = true then
25:                                 return true
26:     return false
```

The example discussed above generates the following derivation.

1	$(C(x_4) \vdash_{x_4,w_1,z} C(x_4))$	Id
2	$(A(x_3) \vdash_{x_3,y} B(x_3))$	Apply θ_A to Axiom 1
3	$(\top \vdash_{x_3,y} B(x_3))$	$\top$
4	$(C(x_4) \vdash_{y,w_1,x_4,w_2,z} C(x_4))$	Apply θ_B to 1
5	$((B(x_4) \wedge C(x_4)) \vdash_{y,w_1,x_4,w_2,z} D(x_4,w_1))$	Apply θ_B to Axiom 2
6	$(C(x_4) \vdash_{y,w_1,x_4,w_2,z} \top)$	$\top$
7	$(\top \vdash_{y,w_1,x_4,w_2,z} B(x_4))$	Apply θ_B to 3
8	$(C(x_4) \vdash_{y,w_1,x_4,w_2,z} B(x_4))$	Cut 6, 7
9	$(C(x_4) \vdash_{y,w_1,x_4,w_2,z} (B(x_4) \wedge C(x_4)))$	$\wedge$ 8, 4
10	$(C(x_4) \vdash_{y,w_1,x_4,w_2,z} D(x_4,w_1))$	Cut 9, 5
11	$(C(x_4) \vdash_{y,w_2,x_4,z} D(x_4,w_2))$	10 $[w_2/w_1]$

If $y : Y$, Y is a closed sort, and $k : Y$ is constant term, then the substitution $[k/y]$ yields the goal sequent.

An implementation of the algorithms discussed in this paper will be available at [36]. In our AI courses we have used the Prover9 [21] and Vampire [29] classical resolution-based theorem provers.

References

[1] H. Abelson, G. J. Sussman & J. Sussman (1996): *Structure and Interpretation of Computer Programs*, second edition. MIT Press.

[2] S. Awodey & K. Kishida (2008): *Topoology and Modality: The Topological Interpreteation of First-Order Modal Logic*. The Review of Symbolic Logic. Available at `https://www.andrew.cmu.edu/user/awodey/preprints/FoS4.pdf`.

[3] M. Barr (1991): **-Autonomous Categories and Linear Logic*. Mathematical Structures in Computer Science 1, pp. 159–178. Available at `https://www.math.mcgill.ca/barr/papers/scatll.pdf`.

[4] J. L. Bell (2005): *Set Theory: Boolean-Valued Models and Independence Proofs*, third edition. Oxford University Press.

[5] R. Blute, J. Desharnais, A. Edalat & P. Panangaden (1997): *Bisimulation for Labeled Markov Processes*. In: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science, IEEE, pp. 149–158.

[6] N. N. Čencov (1982): *Statistical Decision Rules and Optimal Inference*. Translations of Mathematical Monographs 53, American Mathematical Society.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest & C. Stein (2009): *Introduction to Algorithms*, third edition. MIT Press.

[8] P. Domingos, S. Kok, D. Lowd, H. Poo, M. Richardson & P. Singla (2008): *Markov Logic*. In L. D. Raedt, P. Frasconi, K. Kersting & S. Muggleton, editors: Probabilistic Inductive Logic Programming, Springer-Verlag.

[9] M. Eytan (1981): *Fuzzy Sets: a Topos Logical Point of View*. Journal of Fuzzy Sets and Systems 5, pp. 47–67.

[10] P. Freyd & A. Scedrov (1990): *Categories, Allegories*. North-Holland.

[11] M. Giry (1985): *A Categorical Approach to Probability Theory*. In: Categorical Aspects of Topology and Analysis, Lecture Notes in Mathematics 915, Springer-Verlag, pp. 68–85.

[12] J. A. Goguen (1967): *L-Fuzzy Sets*. Journal of Mathematical Analysis and its Applications 18, pp. 145–174.

[13] M. Jackson (2006): *A Sheaf Theoretic Approach to Measure Theory*. Ph.D. thesis, University of Pittsburgh.

[14] P. T. Johnstone (2002): *Sketches of an Elephant: A Topos Theory Compendium*. Oxford University Press.

[15] B. W. Kernighan & D. M. Ritchie (1988): *The C Programming Language*, second edition. Prentice Hall PTR.

[16] J. Lambek & P. J. Scott (1986): *Introduction to Higher-Order Categorical Logic*. Cambridge Studies in Advanced Mathematics 7, Cambridge University Press.

[17] F. W. Lawvere (1986): *Taking Categories Seriously*. In: Revista Colombiana de Mathematicas, XX, pp. 147–178. Available at `http://www.tac.mta.ca/tac/reprints/index.html`.

[18] F. W. Lawvere & S. Schanuel (2009): *Conceptual Mathematics*, second edition. Cambridge University Press.

[19] G. F. Luger (2005): *Artificial Intelligence Structures and Strategies for Complex Problem Solving*, fifth edition. Addison Wesley.

[20] E. G. Manes (1982): *A Class of Fuzzy Theories*. Journal of Mathematical Analysis and Applications 85, pp. 409–451.

[21] W. McCune (2009): *Prover9 Automated Theorem Prover*. Available at `https://www.cs.unm.edu/~mccune/prover9/`.

[22] X. Meng (1992): *The Categories of Convex Sets and of Generalized Metric Spaces with Applications in Statistical Decision Theory, Stochastic Dynamic Programming, and Related Areas*. M.S. Thesis. SUNY Buffalo.

[23] T. M. Mitchell (1997): *Machine Learning*. McGraw-Hill.

[24] S. Russell & P. Norvig (2021): *Artificial Intelligence a Modern Approach*, fourth edition. Pearson Education.

[25] I. Schiopu (1973): *A Categorical Approach to Probability Theory*. M.S. Thesis. Dalhousie.

[26] D. Scott & P. Krauss (1966): *Assigning Probabilities to Logical Formulas*. In J. Hintikka & P. Suppes, editors: *Aspects of Inductive Logic*, North-Holland.

[27] L. N. Stout (1991): *A Survey of Fuzzy Set and Topos Theory*. *Journal of Fuzzy Sets and Systems* 42, pp. 3–14.

[28] A. S. Troelstra & H. Schwichtenberg (2000): *Basic Proof Theory*, second edition. Cambridge University Press.

[29] A. Voronkov et al. (1994): *The Vampire Automated Theorem Prover*. Available at `https://vprover.github.io`.

[30] M. Wendt (1994): *The Category of Disintegration*. *Cahiers de Topologie et Géométrie Différentielle Catégoriques* XXX-4, pp. 291–296.

[31] R. L. Wojtowicz (2002): *On Categories of Cohesive-Active Sets and Other Dynamic Systems*. Ph.D. thesis, University of Illinois at Urbana-Champaign. Available at `http://www.adjoint-functors.net/wojtowicz-thesis02.pdf`.

[32] R. L. Wojtowicz (2004): *Symbolic Dynamics and Chaos Defined by Right Adjointness*. In D. Dubois, editor: *American Institute of Physics Conference Proceedings*, 4 edition, pp. 268–281. Available at `http://www.adjoint-functors.net/aipcasys2.pdf`.

[33] R. L. Wojtowicz (2005): *Categorical Logic as a Foundation for Reasoning Under Uncertainty and as a Guide to Machine Learning Algorithm Development*. SBIR Phase I Final Report.

[34] R. L. Wojtowicz (2008): *On Transformations Between Belief States*. In: *Soft Methods for Handling Variability and Imprecision*, Advances in Soft Computing, Springer-Verlag, pp. 313–320. Available at `http://www.adjoint-functors.net/belief.pdf`.

[35] R. L. Wojtowicz (2009): *Non-Classical Markov Logic and Network Analysis*. In: *IEEE 12th International Conference on Information Fusion. Seattle, WA*. Available at `http://www.adjoint-functors.net/f2009pp.pdf`.

[36] R. W. Wojtowicz: *Source Code for Logic-Based Artificial Intelligence Algorithms Supporting Categorical Semantics*. Available at `http://www.adjoint-functors.net/cai`.

[37] O. Wyler (1994): *Fuzzy Logic and Categories of Fuzzy Sets*. In: *Non-Classical Logics and their Applications to Fuzzy Subsets, Theory and Decision Library* 32, Springer-Verlag, pp. 291–296.